

# Dynamic Routing of Locally Restorable Bandwidth Guaranteed Tunnels using Aggregated Link Usage Information

Murali Kodialam T. V. Lakshman

Bell Laboratories  
Lucent Technologies  
101 Crawfords Corner Road  
Holmdel, NJ 07733, USA

{muralik, lakshman}@bell-labs.com

*Abstract*—This paper presents new algorithms for dynamic routing of locally restorable bandwidth guaranteed paths. Dynamic routing implies routing of requests that arrive one-by-one with no a priori knowledge of future arrivals, and so necessitating use of on-line algorithms. *Local restorability* means that upon a link or node failure, the first node upstream from the failure must be able to switch the path to an alternate preset outgoing link so that path continuity with bandwidth guarantees is restored by a strictly local decision. The motivation for use of local restoration is that it is much faster than path restoration because failure information does not have to propagate to the source. Local restoration implies that to successfully route a path set-up request an active (primary) path, and a bypass backup path for every link and node used by the active path must be determined. This locally restorable on-line routing problem is becoming particularly important in optical networks and in MPLS (Multi Protocol Label Switching) based networks due to the trend toward dynamic provisioning of bandwidth guaranteed or wavelength paths. To prevent excessive resource usage for backup paths, and to satisfy the implicit service provider requirement of optimizing network resource utilization so as to increase the number of potential future demands that can be routed, it is desirable to judiciously share backup paths while still maintaining local restorability. The best sharing performance is achieved if the routing of every path in progress in the network is known to the routing algorithm at the time of a new path set-up. However, this requires maintenance of non-aggregated or per-path information which is not often desirable particularly when distributed routing is preferred. We show that a partial information scenario which uses only aggregated and not per-path information provides sufficient information for efficient dynamic routing of locally restorable bandwidth guaranteed paths. In this partial information scenario the routing algorithm only knows what fraction of each link's bandwidth, is currently used by active paths, and is currently used by backup paths. Obtaining this information is feasible using proposed traffic engineering extensions to routing protocols. We develop efficient dynamic routing algorithms for bandwidth guaranteed paths that are locally restorable under single link or node failure. The routing is done using a sequence of shortest path computations, and it permits sharing of backup paths between requests as well as between the backup paths for different network elements for the same request. We compare the routing performance of our algorithm to other known restoration schemes. Our partial information based locally restorable algorithm performs very well in terms of the number of rejected requests and total bandwidth usage.

## I. INTRODUCTION

The locally restorable dynamic routing problem considered in this paper is motivated by trends in backbone and transport networks toward dynamic provisioning of bandwidth guaranteed paths with fast restoration capability. An important context in which such fast restoration has been proposed is in Multi-Protocol-Label-Switching (MPLS) [3], [1],[7],[6],[8] and in this paper, for ease of exposition, we mostly focus

on Multi-Protocol-Label-Switching (MPLS) or MPLS-related applications. However, the presented algorithms are also usable in other networking applications requiring dynamic restorable bandwidth provisioning.

In MPLS [3] packets are encapsulated, at ingress points, with labels that are then used to forward the packets along label switched paths (LSPs). These LSPs can be thought of as virtual traffic trunks that carry flow aggregates generated by classifying the packets arriving at the edge or ingress routers of an MPLS network into "forwarding equivalence classes" [3]. This classification into flow aggregates combined with explicit routing of bandwidth guaranteed LSPs enables service providers to traffic engineer their networks [1] and to dynamically provision bandwidth guaranteed paths. Recently, proposals [7], [8], [11] have been made to incorporate restoration mechanisms in MPLS. These restoration mechanisms allow backup paths, onto which traffic can be quickly redirected upon failure detection, to be setup simultaneously with the active path thus ensuring that an LSP if set-up is quickly restorable upon failure.

The incorporation of restoration leads to new QoS routing problems. One possibility is to dynamically route both an active path and a backup path in order to satisfy a request to set-up a restorable bandwidth guaranteed LSP. Routing algorithms for this scenario are presented in [10]. Simultaneous routing of both paths ensures that sufficient resources will be available upon failure for successful LSP restoration. However, with this scheme the restoration actions of active path failure detection and backup activation are performed by the source node. Delays entailed in propagation of failure information to the source node may preclude this scheme, known as path restoration, from achieving restoration times comparable to the 50 ms restoration associated with SONET rings.

When MPLS restoration times must be comparable to SONET restoration times, the proposed MPLS fast restoration mechanism [1] is a faster alternative to path restoration from the source. In this paper, we use the more representative term local restoration instead of fast restoration. *Local restorability* means that upon a link or node failure, the first node upstream from the failure must be able to switch the path to an alternate preset outgoing link so that path continu-

ity with bandwidth guarantees is restored by a strictly local decision. Local restoration implies that to successfully route a path set-up request an active path, and a bypass backup path for every link and node used by the active path must be determined. This QoS routing problem of on-line routing of locally restorable bandwidth guaranteed paths has not been well studied.

In this paper, we present new algorithms for this problem of setting up locally restorable bandwidth guaranteed tunnels. Since we are focusing on the MPLS application, we use the terms LSP and tunnels synonymously in the rest of the paper. We concentrate on bandwidth routing because this is the most likely traffic engineering use for setting up QoS guaranteed paths. If QoS constraints such as delays and losses are to be incorporated in service level agreements (SLA), one way of accommodating this, given the traffic descriptor and SLA, is to convert such an SLA into an effective bandwidth requirement for the LSPs (with the queueing delays and losses primarily restricted to the network edges) which can then be routed through the MPLS network as a constant-bit-rate stream encountering only negligible or predictable queueing delays in the MPLS core network. Routing taking delay and loss metrics directly into account is difficult computationally and requires information difficult to acquire such as nodal load versus delay characteristics. The problem is further compounded when backup paths have to be routed as well.

Note that an approach essentially similar to routing of bandwidth guaranteed paths can be used for dynamic wavelength-path set-up in optical networks (particularly when wavelength conversion is permitted at each optical crossconnect). Here a wavelength can be thought of as the outermost (non-stackable) label in the MPLS label stack. These functional similarities between setting-up wavelength switched paths and setting-up MPLS label-switched paths have been pointed out in [2] as a basis for integrating the optical layer control plane and Multi Protocol Label Switched (MPLS) control plane (as also the fact that integration permits more efficient network resource allocation). Given these similarities and the possible standardization effort toward integrated control protocols [2], we present our routing algorithms in a more general setting and use the term LSP to denote either a bandwidth-guaranteed MPLS label-switched path or a wavelength ( $\lambda$ ) switched path. The rest of the paper discusses LSP routing, with LSPs as defined above. Our algorithms can be used for routing wavelength paths in optical networks as well as for routing bandwidth-guaranteed label-switched paths. The restoration feature is particularly important in optical networking.

#### A. Online Routing

Since we are interested in dynamic routing of LSPs, we cannot use offline algorithms that assume that all the restorable demands that are to be routed are known a priori or assume that any number of existing LSPs can be rerouted

to accommodate a new LSP. Instead, the LSP requests that arrive one-by-one have to be routed by an on-line algorithm that routes both the active path and the backup path for each link or node while meeting the service provider traffic engineering requirement of optimizing network resource utilization so as to increase the number of potential future demands that can be routed.

We assume that requests for LSP set-up arrive one at a time. Each request has an ingress node, an egress node and an associated bandwidth (for wavelength switched paths, the bandwidth is just the wavelength capacity or unit bandwidth if all wavelengths have identical capacity). For every request, the objective of the routing algorithm is to compute an active path and a backup path for each node or link used by the active path. If sufficient bandwidth is not available to set up either the active path or the backup paths then the connection set-up request is rejected.

We only consider the case of protection against single link (node) failures. This is because the backup path is likely to be used only for a short time until a new active path is set-up. Secondly, protection against multiple failures requires multiple backups and this is too expensive in backup resource requirements.

## II. RESTORATION OPTIONS AND FEATURES

We discuss in more detail some of the features of different restoration models relevant to the problem considered in the paper. We first contrast local versus end-to-end or path restoration. We then discuss the failure modes that we protect against and the information about the network that the algorithm needs.

#### A. Restoration Model

As mentioned before, two options for restoration are *end-to-end or path restoration* and *local restoration*. In the case of end-to-end restoration, the idea is to provide a backup path from the source to the destination for each request. This backup path is (link or node) disjoint from the active path. However the drawback with this approach is that when there is a link or node failure, this information has to propagate back to the source which in turn switches from the active to the secondary path for all the demands that use the link. Note that the link failure information has to be potentially propagated to all nodes in the network. End-to-end restoration along with the information transfer on failure is illustrated in Figure 1 and in Figure 2.

As noted before, the time taken for this information propagation to the source may not be acceptable for many applications. Therefore we consider a local restoration model. In the case of local restoration, the backup paths are preset locally and therefore failure information does not have to propagate back to the source before connections are switched to the backup path.

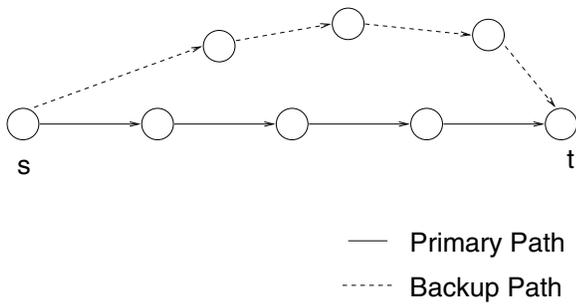


Fig. 1. Paths for End-to-End Restoration

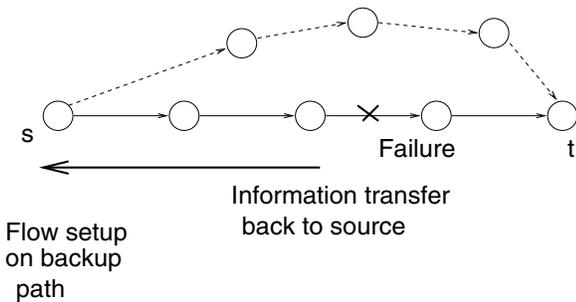


Fig. 2. Information Transfer on Link Failure

### B. Failure Modes

We consider two different failure modes that we protect against. The first is against *single link failures*. In this case, we have to protect the active path against all single link failures. The second is the *single element (node or link) failures*. The amount of resources needed to provide backup for the second mode will be greater since a node failure results in multiple link failures. In terms of detecting failures, we assume the following: In the single link failure model, when a link fails, the two nodes that are at the end point of the link know that the link has failed and they immediately switch all the demands that go on this link to the alternate path. Note that when a node fails, all the links incident on this node fail and in this case the backup path designed to protect against link failures may also fail. In the single *element failure* model, when a node fails we assume that all the link interfaces at that node fail and therefore all links that are incident on the node fail. This is detected as in the link failure case and the demands are routed across the failed node. Note that in the case of single element failures, there has to be a backup path for every node, instead of every link, traversed by the active path. We do not provide a backup if the source or the destination of the traffic fails. Taking care of single node failures almost handles all the link failures also except for the last link on the active path. This is illustrated in Figure 3. Therefore for the single element failure case, we protect against all single node failures and the failure of the last link.

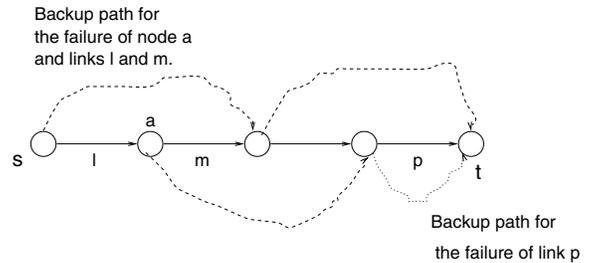


Fig. 3. Backup Path for Single Element Failure

### C. 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 include link  $(i, j)$ . This backup path for link  $(i, j)$  can include any link including any links on the active path for the current demand (apart from link  $(i, j)$ ), as well as any links that are used in the backup path for other active links for this demand. For single element failures, the backup path for the failure of node  $k$  involves doing the following: First determine the links  $(j, k)$  and  $(k, l)$  in the active 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 along this backup path. Note that the backup path for the failure of node  $k$  has to avoid all links incident on node  $k$ .

### D. Sharing Backup Links

Clearly, capacity on the active path cannot be shared. However, the capacity in the backup path can be shared in two ways: Inter-demand sharing and intra-demand sharing. *Inter-demand sharing* refers to sharing of the backup bandwidths belonging to different demands whose active paths are link disjoint. For example, if two equal demands between a given source and destination do not share any links in common on the active path, then the backup path for these two demands can be shared completely. This is an extreme case. However, even if the two demands on the primary path share some links in common, it may still be possible to share capacity on the backup path.

*Intra-demand sharing* is illustrated using Figure 4. In the figure note that link  $(8, 4)$  is used to backup links  $(2, 3)$  and  $(3, 4)$ . This means that backup capacity is shared on this link for backups belonging to the same demand and is an example of intra-demand sharing. The algorithms that we develop exploit both inter-demand and intra-demand sharing in order to minimize the amount of bandwidth used. Figure 5 illustrates sharing backups for the single element failure case.

The amount of sharing that is achievable depends on the amount of link-usage information that is available to the routing algorithm. We discuss this next.

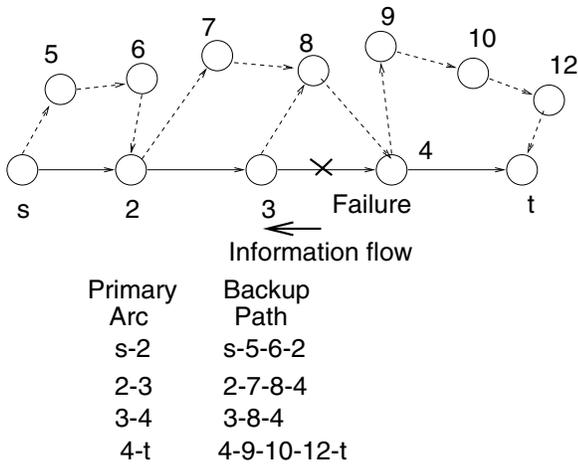


Fig. 4. Forward and Backup Path for Single Link Failure

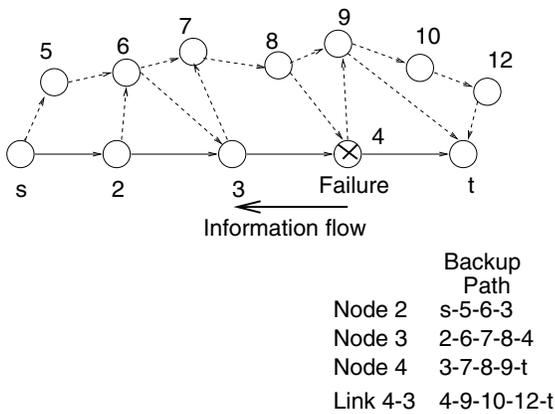


Fig. 5. Forward and Backup Path for Single Element Failure

### E. None, Complete, and Partial Information for Routing

The first scenario that we consider is what we call the *no information* case. In this scenario, we assume that the only information that the routing algorithm has about the network is the residual (available) bandwidth on each link. The residual bandwidth is defined as the difference between the link capacity and the amount of bandwidth already taken by the active and the backup paths traversing the link. This information is obtainable from routing protocol extensions similar to those in [4], [9], [12]. However, note that in this scenario, for each link the amount of bandwidth utilized separately by the active and the backup paths is not known. Only the total used bandwidth is known.

In the second scenario, we assume that the routing algorithm has *complete information*, i.e., it knows the routes for the active and backup paths of all the connections currently in progress. This is too much information to make it feasible for availability via routing protocol extensions. If routing is done in a centralized manner this information can be maintained by the routing algorithm. However, if the computation is distributed, then it would be very difficult to disseminate this information to all the nodes. The amount of information needed for the complete information model is very large.

In the third *partial information* scenario, the information available to the routing algorithm is slightly more than that in the no information scenario. The additional information in this scenario is that for each link instead of knowing only the total bandwidth usage, we now separately know the total bandwidth used by active paths, and the total bandwidth used by backup paths. This incremental information is very useful and it is possible to disseminate it in a distributed manner. It is feasible to obtain this information from traffic engineering extensions to routing protocols provided the backup and active paths are grouped into separate classes for which link bandwidth usage is distributed.

In the first no information scenario, it is not possible to do any inter-demand sharing of the backup paths since the relevant information on backup bandwidth usage is not available. Intra-demand sharing is still possible since the source node has the backup bandwidth usage information for the current demand. The second complete information scenario permits the best sharing but is not always practical. So it is mainly useful only for comparison purposes. The third partial information scenario is fairly modest in terms of the amount of information to be maintained. Because only aggregate information is needed and no per-LSP information is needed it is easy to maintain and use this information in a distributed fashion. Therefore on-line routing of bandwidth guaranteed active and backup paths for local restoration under the partial information model is the main focus of this paper.

III. PROBLEM DEFINITION

We consider a network of  $n$  nodes (switches/routers) and  $m$  links. All the links are assumed to be directional. Each bandwidth demand arrives at a route server or edge-router which determines the explicit-route for the active and the backup paths for that demand. The request either arrives directly to the route server or may first arrive at an ingress node which then queries the route server to generate the explicit route. We leave out the details of protocols that may be used for this interaction for the sake of conciseness. The explicit route is then communicated back to the ingress router which then uses a signaling mechanism such as PNNI in ATM networks or RSVP/CR-LDP in IP/MPLS networks to set-up the path to the egress and to reserve bandwidth on each link on the path. We consider the request for tunnel  $k$  to be defined by a triple  $(o_k, t_k, b_k)$ . The first field  $o_k$ , specifies the ingress router, the second field  $t_k$  specifies the egress router and the third  $b_k$  specifies the amount of bandwidth required for tunnel  $k$ . For each tunnel request, an *active path* and a set of *backup paths* have to be set up. In the single link failure case, we assume that there will be local restoration on any single link failure in the network. In the single element failure case, there will be local restoration in the case of single link or single node failure. If we determine that there is not sufficient

bandwidth in the network to either set up the active path or any of the backup paths for a tunnel request, then this request is rejected. In [10], the restoration model, is to have a backup path between the source and the destination of the demand that is disjoint from the active path. In this paper we consider the case where the restoration has to be local, i.e., the switch to a bypass backup path has to be done at the first upstream node and not at the source node.

Tunnel requests are assumed to come one at a time. For ease of notation, assume that the current tunnel request is for  $b$  units of bandwidth between source node  $s$  and destination node  $t$ . If this tunnel request is accepted, then note that all links on its active path will reserve  $b$  units of bandwidth for this tunnel.

We would like to determine the amount of sharing that can be achieved under the different information scenarios. Towards this end we define the following notation. Let  $A_{ij}$  represent the set of demands that use link  $(i, j)$  in their active paths and  $B_{ij}$  represent the set of demands that use link  $(i, j)$  for backup. Let  $F_{ij}$  represent the total amount of bandwidth reserved for the demands that use link  $(i, j)$  on the active path. Let  $G_{ij}$  represent the total amount of bandwidth reserved for backup by links whose backup paths use link  $(i, j)$ .

$$F_{ij} = \sum_{k \in A_{ij}} b_k$$

and

$$G_{ij} = \sum_{k \in B_{ij}} b_k.$$

Let  $R_{ij} = C_{ij} - F_{ij} - G_{ij}$  represent the residual bandwidth of link  $(i, j)$  which is all that is known in the *no information* case. For the *complete information* case we assume that each node knows the sets  $A_{ij}$  and  $B_{ij}$  for all links  $(i, j)$  in the network. 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 demands 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*. If no restoration is needed, i.e., we just have to determine one path, this bandwidth minimization objective leads to min-hop routing. 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 routing of demands that are currently active in the network.

#### IV. 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 these integer programming problems. Therefore we design a heuristic algorithm to solve the

restoration problem. In this section, we outline the key ideas used to the developed algorithm. In the next section we give a formal description of the algorithm.

First we consider the single link failure case, ignoring the intra-demand sharing of backup bandwidth, under the partial and complete information models. Analysing this case gives us the overall design of the algorithm. If link  $(i, j)$  is used in the active path, then there has to be a backup path bypassing link  $(i, j)$  so as to backup any failure of this link. Note that the backup path has to start at node  $i$  but can terminate at any node between node  $j$  and the destination  $t$  on the active path. For now we ignore this, and consider the case where the backup path for link  $(i, j)$  has to start at  $i$  and terminate at node  $j$ . In this case, the overall bandwidth needed when link  $(i, j)$  is used in the active path is the sum of the bandwidth for using it in the active path and the bandwidth used for backing up the link. The bandwidth needed if link  $(i, j)$  is used on the active path is  $b$ . The bandwidth needed to backup up link  $(i, j)$  can be computed as the shortest path from node  $i$  to node  $j$  after removing link  $(i, j)$ .

The cost of the links depends on the information model used. In the case where there is complete information, the sets  $A_{ij}$  and  $B_{ij}$  are known for all links  $(i, j)$ . Since we are assuming robustness under single link failures, it is possible to share backup paths between demands whose active paths do not share the same link. In order to formulate this problem, we first define the quantity  $\theta_{ij}^{uv}$  for each link pair  $(i, j)$  and  $(u, v)$ . This quantity  $\theta_{ij}^{uv}$  is the cost of using link  $(u, v)$  on the backup path if link  $(i, j)$  is used in the active path. In order to compute the value of  $\theta_{ij}^{uv}$  we first define the set  $\phi_{ij}^{uv} = A_{ij} \cap B_{uv}$ . This is the set of demands that use link  $(i, j)$  on the active path and link  $(u, v)$  on the backup path. Let the sum of all the demand values in the set  $\phi_{ij}^{uv}$  be represented by  $\delta_{ij}^{uv} = \sum_{k \in \phi_{ij}^{uv}} d_k$ . Recall that the current demand is for  $b$  units of bandwidth between nodes  $s$  and  $d$ . Now  $\theta_{ij}^{uv}$  is defined as follows:

$$\theta_{ij}^{uv} = \begin{cases} 0 & \text{if } \delta_{ij}^{uv} + b \leq G_{uv} \\ & \text{and } (i, j) \neq (u, v) \\ \delta_{ij}^{uv} + b - G_{uv} & \text{if } \delta_{ij}^{uv} + b > G_{uv} \text{ and} \\ & R_{uv} \geq \delta_{ij}^{uv} + b - G_{uv} \\ & \text{and } (i, j) \neq (u, v) \\ \infty & \text{Otherwise} \end{cases}$$

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

$$\theta_{ij}^{uv} = \begin{cases} 0 & \text{if } F_{ij} + b \leq G_{uv} \text{ and} \\ & (i, j) \neq (u, v) \\ F_{ij} + b - G_{uv} & \text{if } F_{ij} + b > G_{uv} \text{ and } R_{uv} \geq \\ & F_{ij} + b - G_{uv} \text{ and } (i, j) \neq (u, v) \\ \infty & \text{Otherwise} \end{cases}$$

This is based on the observation that

$$\delta_{ij}^{uv} \leq F_{ij} \quad \forall (i, j) \quad \forall (u, v).$$

Note that since links  $(i, j)$  and  $(u, v)$  cannot be on both the active and the backup paths, the value of  $\theta_{ij}^{uv}$  is set to infinity if  $(i, j) = (u, v)$ . The quantity  $\delta_{ij}^{uv}$  represents the amount of backup capacity on link  $(u, v)$  that is needed backup already routed demands that use link  $(i, j)$ , and hence the amount that cannot be used to backup the current demand if it too were to use link  $(i, j)$  in the active path. If  $\delta_{ij}^{uv} + b \leq G_{uv}$  then note that the current demands can be backed up on link  $(u, v)$  without reserving any additional bandwidth. For the partial information scenario, since  $\delta_{ij}^{uv}$  is not known to the routing algorithm, we use  $F_{ij}$  as a conservative approximation for  $\delta_{ij}^{uv}$ .

Next we determine the cost of using link  $(i, j)$ . This cost is the sum of the cost of using link  $(i, j)$  on the active path and the cost of its bypass path. To determine the cost of bypassing link  $(i, j)$ , we compute the shortest path from  $i$  to  $j$  (excluding link  $(i, j)$ ) where the cost of each link  $(u, v)$  in the path is given by  $\theta_{ij}^{uv}$ . Let the length of this shortest path between  $i$  and  $j$  be  $\phi_{ij}$ . Then the cost of using link  $(i, j)$  on the active path is  $b + \phi_{ij}$ , i.e., the sum of the bandwidth usage on link  $(i, j)$  and bandwidth usage for bypass of link  $(i, j)$ . Once usage costs are associated with each link in the network (using a total of  $m$  shortest path computations), we now compute the shortest path between  $s$  and  $t$  using  $b + \phi_{ij}$  as the cost of link  $(i, j)$ . This gives the minimum amount of bandwidth without intra-demand sharing taken into account. Therefore we totally solve  $m + 1$  shortest path problems. This leads to the first design idea: *The cost of backup paths can be determined by solving shortest path problems, one for each link in the network.*

In the above discussion, we ignored the fact that the backup path for link  $(i, j)$  starts at  $i$  but can end at any node on the path from  $j$  to  $t$  (including  $j$  and  $t$ ). Handling this case is facilitated by executing the shortest path algorithm backwards from the destination to the source.

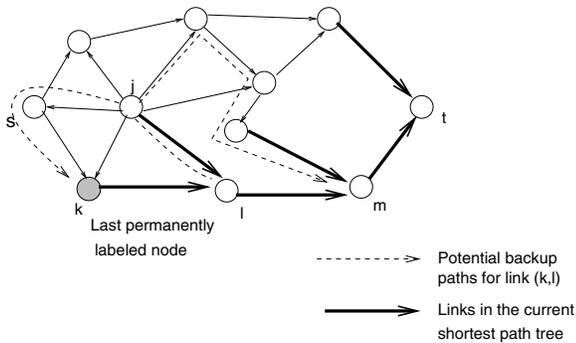


Fig. 6. Computing Backup Cost

This facilitation is illustrated by Figure 5 which shows a step in the backwards execution of the algorithm. The dark lines in the graph represent the shortest path tree when Dijkstra's algorithm is executed backwards from the destination. For every node that is permanently labeled in the shortest path tree there is a unique path from that node to the sink. Consider a node  $k$  that is permanently labeled when we are constructing

the shortest path tree from the sink. Associated with node  $k$  is the path  $P(k) = \{l, m, t\}$  along the shortest path tree from node  $k$  to the destination. Consider link  $(k, j)$  in the network. The cost of using link  $(k, j)$  in the active path is the sum of bandwidth currently being routed and the cost of backing up link  $(k, j)$ . The dotted lines in Figure 5 illustrate three different paths to backup link  $(k, j)$ . In the last section, the cost of backing up link  $(k, j)$  was computed as the shortest path from  $k$  to  $j$  with  $\theta_{kj}^{uv}$  as the cost of link  $(u, v)$ . Now instead of computing the shortest path from  $j$  to  $k$  we compute instead the shortest path from  $k$  to any node in  $P(k)$ . This can be done easily by running Dijkstra's algorithm from  $j$  using  $\theta_{jk}^{uv}$  on link  $(u, v)$ , and terminating the algorithm when any node in the set  $P(k)$  is permanently labeled by the algorithm. This examples illustrates the reasoning leading to the second design principle: *It is necessary to execute the shortest path (Dijkstra's) algorithm backwards starting at the sink.*

To derive the next key idea, we consider the single link failure case where we also take into account intra-demand sharing of backup bandwidth. Intra-demand sharing of bandwidth occurs when the link  $(i, j)$  uses link  $(u, v)$  for a backup and reserves a bandwidth of  $w$  on link  $(u, v)$ . When some other link  $(k, l)$  on the active path wants to use link  $(u, v)$  for its backup then, in addition to any inter-demand sharing it can use the already reserved bandwidth of  $w$  for free. Recall that the shortest path algorithm is to be run backward from the destination. In order to keep track of how much bandwidth is reserved at each link, we introduce a  $m$ -vector  $\lambda^u$  corresponding to node  $u$  in the network. (Recall that the number of links in the network is  $m$ .)  $\lambda_{ij}^u$  represents the amount of bandwidth reserved by the current demand for all the backup paths for all the links leading from node  $u$  to the destination  $t$ . This bandwidth reservation for the current demand can be used to save bandwidth, by intra-demand sharing, when backing up the links from  $u$  to 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  $\lambda^j$  represents the reservation made for this demand for all the backup path from node  $j$  to the sink. This path is known since there is a unique path from node  $j$  to the sink in the shortest path tree. Consider a link  $(m, n)$  in the network. Define

$$\kappa_{mn} = F_{kj} + b - B_{mn} - \lambda_{mn}^j.$$

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

$$l_{mn} = \begin{cases} 0 & \text{if } \kappa_{mn} \leq 0 \\ \delta_{mn} & \text{if } 0 \leq \kappa_{mn} \leq b \text{ and } R_{mn} \geq \kappa_{mn} \text{ and} \\ & (m, n) \neq (k, j) \\ \infty & \text{Otherwise} \end{cases}$$

The cost in the case of the complete information case can also be modified similarly. Therefore the above procedure gives us a method for accounting for the intra-demand sharing. This

gives the third design principle: *Maintaining the m-vector at each node that gives us the amount of bandwidth reserved for the current demand for backing up all links from the the given node to the destination can be used to account for intra-demand sharing.*

To get node bypass paths, the procedure is almost the same as the edge bypass path case. There are two main differences. The first is that when we want to determine the cost of including link  $(i, j)$  in the active 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 sink, the successors of all the nodes that are permanently labeled by Dijkstra's algorithm are already known since a path has been established from that node to the destination. The second important difference is that when a node fails, all the links incident on the node fails. Therefore the cost of the backup has to account for all the links failing simultaneously. We only consider the outgoing links from the node. For example, when computing the cost of using link  $(i, j)$  in the active path, we have to consider the cost of backing up demands that use link  $(j, l)$  for  $l \in V$ . Therefore the cost of all links (without considering intra-demand savings) have to be modified as follows.

Cost of using link  $(u, v)$  for complete information case:

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

Cost of using link  $(u, v)$  for partial information case:

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

The next algorithm design idea is then the following: *Modifying the cost of the links in the computation of the backup costs can be used to account for node failures.* These ideas are now integrated, and in the next section we give a formal description of the algorithm.

## V. FORMAL DESCRIPTION OF THE ALGORITHM

In this section we describe the algorithm for the single link failure case in the partial information case. The modifications that have to be made for the other cases are straightforward as outlined in the preceding section.

### LOCAL\_EDGE\_DISJOINT( $s, t$ )

- **INITIALIZATION**
  - 1: Reverse all links in the network.
  - 2:  $T = V, P = \emptyset, \phi_t = 0, \phi_j = \infty \forall j \neq t$   
 $\lambda_{mn}^d = 0 \forall (m, n) \in E, Q(t) = \emptyset.$
- **ITERATIVE STEP**
  - 2:  $k = \text{Arg min}_{j \in T} \phi_j$ . If  $k = s$  go to Step 6.
  - 3:  $T = T \setminus \{k\}$  and  $P = P \cup \{k\}$ .
  - 4: For each  $j \in T, (k, j) \in E$ 

$$w_{kj} = \text{ALT\_PATH\_COST}(k, j)$$

$$\text{if } (\phi_j \geq w_{kj} + \phi_k)$$

$$\phi_j = \phi_k + w_{kj}$$

$$Q(j) = k$$
  - 5: Go to Step 2.
- **TERMINATION**
  - 6: Exit.

Let  $s \in V$  represent the source and  $t \in V$  represent the destination. Let  $(i, j) \in E$  denote a directed edge (link) in the graph. Let  $d$  represent the current demand size. Let  $F_{ij}, B_{ij}$  and  $R_{ij}$  represent the total amount of active bandwidth, backup bandwidth and residual capacity on link  $(i, j)$ . We use repeated invocations of Dijkstra's algorithm to generate the path. In the description of the algorithm given below the following notation is used. This is a summary of the notation for all the three routines that are used.  $T$  and  $T'$  represent the set of temporarily labeled nodes and  $P$  and  $P'$  the set of permanently labeled nodes. The node labels during the execution of Dijkstra's algorithm will be represented by  $\phi$  and  $\gamma$ . We use node sized vectors  $Q$  and  $Q'$  to represent the predecessor array along the shortest path tree. With node  $u$  in the graph we associate an arc length array  $\lambda^u$  where  $\lambda_{ij}^u$  represents that amount of bandwidth reserved on link  $(i, j)$  for the current demand on all the links on the path from node  $u$  to the destination  $t$ . All invocations of Dijkstra's algorithm will be from the destination node. Therefore at any given point in the algorithm, for any permanently labeled node  $u$  there is a unique path from  $u$  to the destination  $t$  and hence the value of  $\lambda^u$  is known for any permanently labeled node. We use a link length array  $\beta$  to temporarily store the value of  $\gamma^u$ . The main routine that we use is the LOCAL\_EDGE\_DISJOINT(). This routine calls the routine ALT\_PATH\_COST() which determines the cost of providing a link with a local backup. This is done via several invocations of SHORT\_PRED\_PATH(). The reason for multiple invocations of SHORT\_PRED\_PATH() is the fact that the backup path for a link can terminate at any point on the path from that link to the destination. Since the amount of intra-demand saving is a function of the node where the backup path ends, the shortest path algorithm has to be exe-

cuted from each node in the path from the current link to the destination. Note that both LOCAL\_EDGE\_DISJOINT() as well as SHORT\_PRED\_PATH() are modified versions of Dijkstra's algorithm.

#### ALT\_PATH\_COST( $k, j$ )

- **INITIALIZATION**
  - 1:  $u = k, \text{MIN} = \infty$ .
- **ITERATIVE STEP**
  - 2: If  $u = \emptyset$  go to Step 6.
  - 3:  $\alpha = \text{SHORT\_PRED\_PATH}(k, u, j)$ .
  - 4: if ( $\alpha \leq \text{MIN}$ )
    - $\text{MIN} = \alpha$
    - $\lambda_{mn}^j = \beta_{mn} \quad \forall (m, n) \in E$
  - 5:  $u = Q(u)$  Go to Step 2.
- **TERMINATION**
  - 6: Exit.

#### SHORT\_PRED\_PATH( $k, u, j$ )

- **INITIALIZATION**
  - 1:  $\delta_{mn} = F_{kj} + b - B_{mn} - \lambda_{mn}^u \quad \forall (mn) \in E$ .
  - 2:
$$l_{mn} = \begin{cases} 0 & \text{if } \delta_{mn} \leq 0 \\ \delta_{mn} & \text{if } 0 \leq \delta_{mn} \leq b \text{ and } R_{mn} \geq \delta_{mn} \text{ and } (m, n) \neq (k, j) \\ \infty & \text{Otherwise} \end{cases}$$
  - 3:  $T' = V, P' = \emptyset, \gamma_u = 0, \gamma_j = \infty \quad \forall j \neq u$   
 $\lambda_{mn}^d = 0 \quad \forall (m, n) \in E$
- **ITERATIVE STEP**
  - 4:  $w = \text{Arg min}_{j \in T'} \omega_j$ . If  $w = k$  go to Step 9.
  - 5:  $T' = T' \setminus \{w\}$  and  $P' = P' \cup \{w\}$ .
  - 6: For each  $i \in T', (w, i) \in E$ 
    - if ( $\gamma_i \geq l_{wi} + \gamma_w$ )
      - $\gamma_i = \gamma_w + l_{wi}$
      - $Q'(i) = w$
  - 7: Go to Step 2.
- **TERMINATION**
  - 8: Set  $\beta_{mn} = \lambda_{mn}^u \quad \forall (mn) \in E$ , if arc  $(mn)$  is on the shortest path from  $u$  to  $j$  set  $\beta_{mn} = \lambda_{mn}^u + l_{mn}$ .
  - 9: Exit.

## VI. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of our locally restorable routing algorithm that uses only aggregate information. We also evaluate the performance obtained using the three information models. The experimental set up is the following: We performed experiments on a graph with 15 nodes and 56 links. The test network is shown in Figure 7 below. Each undirected link in Figure 7 represents two directed links. For the first set of experiments, the capacity of the links are set to infinity. For the second set of experiments, the light links have a capacity of 12 units in each direction and the dark links have a capacity of 48 units in each direction. The objective of

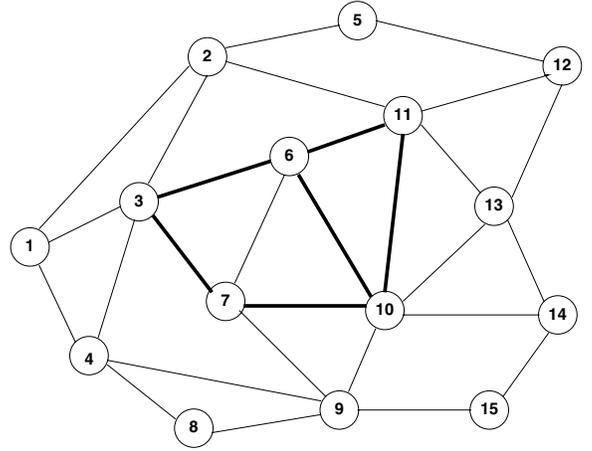


Fig. 7. Test Network

the experiment is to determine the amount of bandwidth savings that we (aggregated) can get by just using partial information. We study the performance of restoration algorithms for the following scenarios:

1. End-to-end restoration with partial information (SPI). For this case, we use the algorithm presented in [10].
2. Local restoration for single link failure with no information (LLNI).
3. Local restoration for single link failure with partial information (LLPI).
4. Local restoration for single link failure with complete information (LLCI).
5. Local restoration for single element failure with no information (LENI).
6. Local restoration for single element failure with partial information (LEPI).
7. Local restoration for single element failure with complete information (LECI).

We measure the effectiveness of the different schemes by doing two experiments.

### A. Network Loading Experiments

In the first experiment, the capacity of the links are set to infinity. Demands arrive one at a time to the network. The source and the destination nodes are picked up at random from

among the nodes in the network. The bandwidth is uniformly distributed between 1 and 6 units. Each demand has to be allocated an active path which is restorable. For restorability, the backup is either an end-to-end path backup, or is a set of node or link bypass paths depending on whether the failure protection is against single element or single link failures. For the no information case, only the residual bandwidth of each link is known to the routing algorithm. For the partial information case, we separately know the total bandwidth used by active paths, the total bandwidth used by backup paths, and the residual bandwidths on each link. With complete information, we know the routes for the active and backup paths of all the connections currently in progress.

The objective of the first set of experiments is to determine the bandwidth efficiency for the different scenarios. After 50 demands have been loaded on to the network, the total amount of bandwidth consumed by the demands (for both active and backup) is determined for each of the information models. We performed 10 experiments with different random seeds. Figure 8 shows the performance of the three different information models in the different information models for the single link failure case and Figure 9 shows the same results for the single element failure case. Clearly having complete information is useful. However, the practical case of having partial information is also useful and the incremental information about active and backup path usage clearly improves performance in comparison to the no information case where only residual bandwidths are available. Comparing the bandwidth usage in the two figures, it is evident that the penalty for protection against element failures is not prohibitively high. Hence, local restorability against single element failures using partial information is the most appealing case.

Since the partial information case is the most appropriate for distributed implementation we compare the partial information case for end-to-end restoration, single link failure and the single element failure cases in Figure 10.

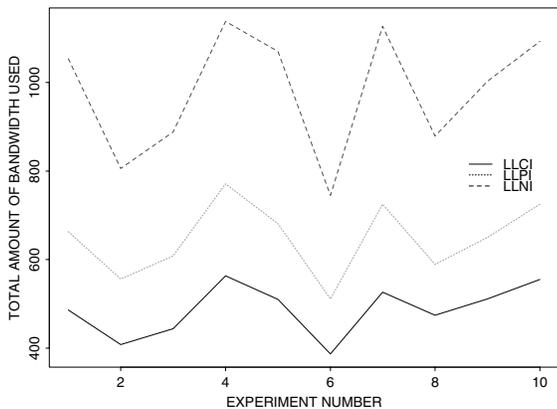


Fig. 8. Total amount of bandwidth consumed after 50 demands for the single link failure backup case

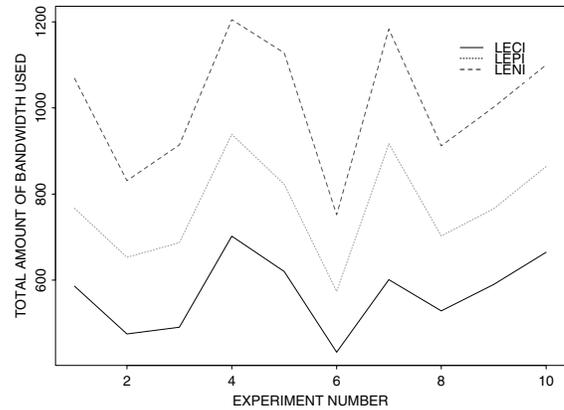


Fig. 9. Total amount of bandwidth consumed after 50 demands for the single element failure backup case

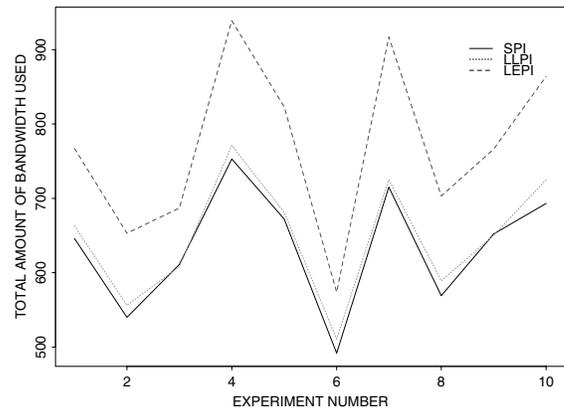


Fig. 10. Total amount of bandwidth consumed after 50 demands for the partial information case

### B. Experiments with dropped demands

The second set of experiments were performed to study the behavior of the algorithms with respect to the number of demands dropped when there is a overloading of the network. In these experiments, the links have finite capacity. Demands arrive one at a time. Each demand is uniformly distributed between 1 and 6 units. The source and the destination for the demands are picked at random. If there is no capacity for either the active or the backup path, then the demand is assumed to be rejected. We performed 10 experiments. We count the number of rejected demands after 40 demands are loaded on to the network. As in the case of the amount of bandwidth consumed, Figure 11 shows the the number of rejected demands (out of 40) for the single link failure backup case for all the three information models, Figure 12 gives the same plot for the single element failure case and Figure 13 gives the number rejected under the partial information scenario for the three different failure models. Again, we see that having

complete information yields the best performance, and that having partial information gives a considerable performance improvement over the no information model.

## VII. CONCLUDING REMARKS

We considered a new QoS routing problem which requires the on-line routing of a bandwidth guaranteed path along with the setting up of bypass paths for every link or node traversed by the primary active path. The bypass paths are used for fast local restoration where upon a link or node failure, the first upstream node re-establishes path continuity (with bandwidth guarantees) by switching to the bypass path for the failed node or link. The routing objective is to minimize the bandwidth usage for each connection so as optimize use of network resources while protecting against single node or link failure. Bandwidth efficiency is achieved by exploiting the potential for inter-demand and intra-demand backup bandwidth sharing. We develop a new algorithm for this routing problem which only uses aggregated link usage information (total bandwidth consumed on each link by active paths, total bandwidth consumed on each link by backup paths, and the residual bandwidths) that is easily obtainable by proposed routing protocol extensions. We show that the algorithm performs well in terms of the number of rejected requests and the total bandwidth used. The main use of this algorithm is for MPLS network routing and for wavelength routing in optical networks with wavelength conversion.

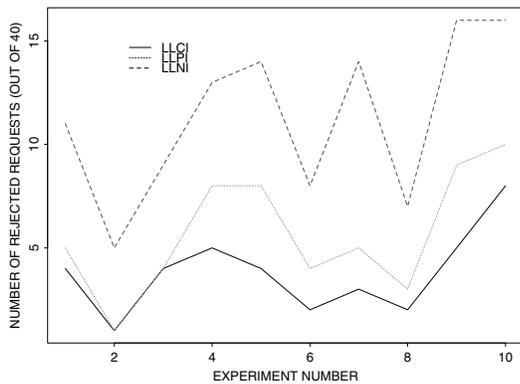


Fig. 11. Number of rejected demands out of 40 for the single link failure backup case

## REFERENCES

- [1] D. O. Awduche, L. Berger, D. Gan, T. Li, G. Swallow, V. Srinivasan, "Extensions to RSVP for LSP Tunnels", *Internet Draft draft-ietf-mpls-rsvp-lsp-tunnel-04.txt*, September 1999.
- [2] D. O. Awduche, Y. Rekhter, J. Drake, R. Coltun, "Multi-Protocol Lambda Switching: Combining MPLS Traffic Engineering Control with Optical Crossconnects", *Internet Draft draft-ietf-awduche-mpls-te-optical-00.txt*, October 1999.
- [3] 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.

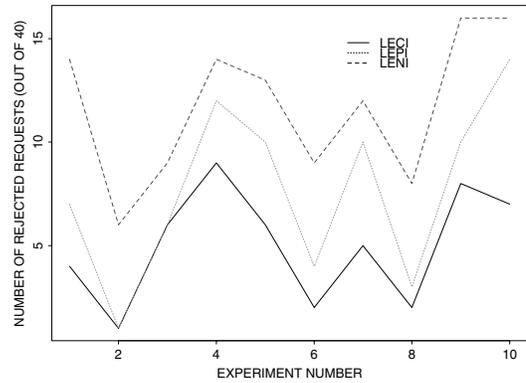


Fig. 12. Number of rejected demands out of 40 for the single element failure backup case

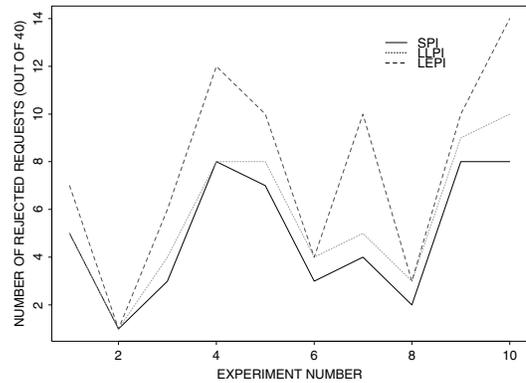


Fig. 13. Number of rejected demands out of 40 for the partial information case

- [4] R. Guerin, D. Williams, A. Przygienda, S. Kamat, A. Orda, "QoS Routing Mechanisms and OSPF Extensions", *Internet Draft draft-guerin-qos-routing-ospf-04.txt*, December 1998.
- [5] R. Guerin, D. Williams, A. Orda, "QoS Routing Mechanisms and OSPF Extensions", *Proceedings of Globecom 1997*.
- [6] R. Goguen, G. Swallow, "RSVP Label Allocation for Backup Tunnels", *work in progress, Internet Draft draft-swallow-rsvp-bypass-label-00.txt*, October 1999.
- [7] D. Haskin, R. Krishnan, R. Boyd, A. Hannan, "A Method for Setting an Alternative Label Switched Paths to Handle Fast Reroute", *work in progress, Internet Draft draft-haskin-mpls-fast-reroute-00.txt*, June 1999.
- [8] R. Krishnan, D. Haskin, "Extensions to RSVP to Handle Establishment of Alternate Label-Switched-Paths for Fast Reroute", *work in progress, Internet Draft draft-krishnan-mpls-reroute-rsvpext-00.txt*, June 1999.
- [9] D. Katz, D. Yeung, "Traffic Engineering Extensions to OSPF", *work in progress, Internet Draft*, 1999.
- [10] M. Kodialam, T. V. Lakshman "Dynamic Routing of Bandwidth Guaranteed Paths with Restoration" *Proceedings of Infocom 2000*, March 2000.
- [11] S. Makam, V. Sharma, K. Owens, C. Huang, "Protection/Restoration of MPLS Networks" *work in progress, Internet Draft draft-makam-mpls-protection-00.txt*, June 1999.
- [12] H. Smit, T. Li, "IS-IS Extensions for Traffic Engineering", *work in progress, Internet Draft*, 1999.