Routing Restorable Bandwidth Guaranteed Connections using Maximum 2-Route Flows

Koushik Kar† Murali Kodialam* T. V. Lakshman*

† ECE Department
University of Maryland
College Park
Maryland, MD 20742

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

***Abstract*— Routing with service restorability is of much importance in Multi-Protocol Label Switched (MPLS) networks, and is a necessity in optical networks. For restoration, each connection has an active path and a disjoint backup path. The backup path enables service restoration upon active path failure. For bandwidth efficiency backups may be shared. This requires that at least the aggregate backup bandwidth used on each link be distributed to nodes performing route computations. If this information is not available, sharing is not possible. Also, one scheme in use for restorability in optical networks is for the sender to transmit simultaneously on the two disjoint paths and for the receiver to choose data from the path with stronger signal. This has the advantage of fast receiver-initiated recovery upon failure but it does not allow backup sharing.**

In this paper, we consider the problem of efficient dynamic routing of restorable connections when backup sharing is not allowed. Our objective is to be able to route as many connections as possible for one-at-a-time arrivals and no knowledge of future arrivals. Since sharing cannot be used for achieving efficiency, the goal is to achieve efficiency by improved path selection. We show that by using the minimum-interference ideas used for non-restorable routing, we can develop efficient algorithms that outperform previously proposed algorithms for restorable routing such as routing with the min-hop like objective of finding two disjoint paths with minimum total hop-count. We present two new and efficient algorithms for restorable routing without sharing, and one of them requires only shortest path computations. We demonstrate that both algorithms perform very well in comparison to previously proposed algorithms.

I. Introduction

Restoring service after failures is an important issue in both MPLS [5], [13] and optical networks [6]. For restoration, each connection is routed along two disjoint paths: a primary (active path) and a secondary (backup) path. The backup path is used for restoring connectivity if the active path fails. The backup path can possibly be shared for bandwidth efficiency. However, this may not be always possible. For sharing to be possible, the nodes performing route computations must know the amount of bandwidth on each link that is currently being used for providing backup. While this can be disseminated by simple extensions to link-state routing protocols, current extensions only disseminate link status and the bandwidth used for carrying active paths [10]. Sharing is also not possible when the sender simultaneously transmits on both paths and the receiver chooses to receive data from the path with the stronger signal. This scheme has been used in optical networks despite its bandwidth inefficiency because it permits very quick and simple restoration since only the receiver needs to detect and act upon failure.

In this paper, we consider the problem of dynamic routing of restorable connections. This is similar to the problem studied in [12]. However, in [12] the emphasis is on the development of algorithms that permit efficient backup sharing while using only aggregate information on active and backup link usage. The no sharing case is studied in [12] to provide an upper bound on efficiency, and the algorithm used computes two disjoint paths that minimize the total link costs, i.e., it is the restorable routing analog of min-hop routing. The focus of this paper is only on the no sharing case. Since we cannot achieve efficiency by sharing, our objective here is to improve performance by improved path selection. For some alternative approaches to this problem in the context of optical networks, see [3] and the references therein.

For dynamic routing of non-restorable connections, a recently proposed algorithm that performs very well is minimum interference routing [9]. Here the idea is that a newly routed connection's path must not interfere too much with paths that might be critical to satisfy future demands (as explained in more detail later). In this paper, we develop two new algorithms for routing restorable connections where the objective is to improve performance by routing using the minimum interference criteria. Note that even though we use the minimum interference criteria, the developed algorithms are new and are not simple extensions of the algorithms in [9]. However, as with the algorithms in [9], the only information needed for routing is the link residual bandwidths and knowledge of network ingress-egress pairs is exploited to achieve performance improvements. No assumptions are made regarding knowledge of future arrivals and connections arrive one-at-a-time to the network. Clearly, we would like to accept as many connections as possible and minimize the number of rejected requests.

The paper is organized as follows. We first review the basic notions pertinent to minimum interference routing. We then formally state the minimum-interference routing problem for restorable connections. Sections IV & V present two new algorithms for min-interference routing of connections with backup. Note that one of them involves only shortest path computations. Performance comparisons demonstrating the efficiency of the

new algorithms are in Section VI. Concluding remarks are in Section VII.

II. Minimum Interference Routing: Basic Ideas

We describe the basic minimum interference ideas as is applicable to the routing of two disjoint paths. A complete description of the motivation for the minimum interference criteria and its use for routing a single path are in [9]. As will be seen below, routing two paths changes the routing problem substantially requiring the development of new routing algorithms.

The key idea in min-interference routing is to pick paths that do not interfere too much with potential future demands between different ingress-egress pairs. To define interference concretely for the restorable routing problem, we require the concept of *maximum 2-route flows* [11] [1], which we describe below.

A. Maximum 2-route Flow

Recall that routing of a restorable demand has to be done over a pair of link-disjoint paths (one for the active and the other for the backup). An *elementary 2-flow* is defined to be a flow of one unit along a pair of link-disjoint paths. Consider the network in Figure 1 of 7 nodes and 8 links, shared by two ingress-egress pairs (S_1, D_1) and (S_2, D_2). In this figure, for example, a flow of one unit on each of the paths 1-4-3 and 1-2-3 together constitute an elementary 2-flow for the pair (S_1, D_1).

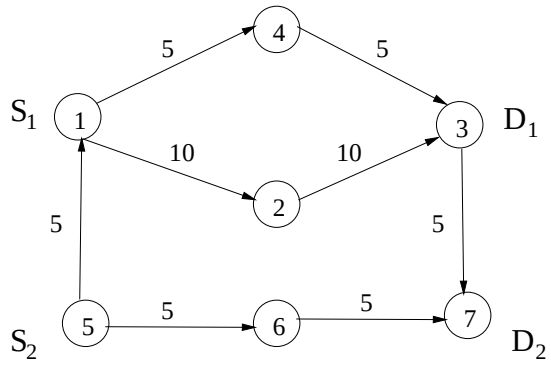

Fig. 1. Illustrative Example

A *2-route flow* is any flow such that can be expressed as a non-negative linear sum of elementary 2-flows. Therefore, a flow of 5 units on both 1-4-3 and 1-2-3 constitute a 2-route flow for (S_1, D_1). Similarly, a flow of 2 units on both 5-1-4-3-7 and 5-1-2-3-7 together with 4 units on 5-6-7, constitute a 2-route flow (it can be decomposed into 2 times an elementary 2-flow along the edge-disjoint path pair 5-1-4-3-7 and 5-6-7, and 2 times an elementary 2-flow along the edge-disjoint path pair 5-1-2-3-7 and 5-6-7). Note that the sum of several 2-route flows is also a 2-route flow. Since every demand with backup routed in a network is a 2-route flow, the bandwidth routed (including both active and backup paths) between any ingress and egress constitute a 2-route flow.

The *value* of a 2-route flow is the total amount of flow (including both active and backup paths) that enters the destination node for that flow. Therefore, for the 2-route flow that represents the bandwidth routed between an ingress-egress pair, the value is twice the total amount of bandwidth routed on the active paths (or the backup paths). Depending on the context, the term 2-route flow will be used to mean the value of the flow as well as the flow itself. The *maximum 2-route flow* between two nodes is a 2-route flow with the maximum value that can be sent between the two nodes, without exceeding the link capacities. The maximum 2-route flow for (S_1, D_1) is 10 units (5 units on both of the paths 1-4-3 and 1-2-3). Similarly, it is easy to see that the maximum 2-route flow for (S_2, D_2) is also 10 units.

The maximum 2-route flow for an ingress-egress pair is an upper-bound on the total amount of restorable bandwidth that can be routed between that ingress-egress pair. Therefore, the maximum 2-route flow for an ingress-egress pair can be considered as a measure of the "open capacity" between that pair. When we route a demand, the maximum 2-route flow for one or more ingress-egress pairs could possibly decrease. In min-interference routing, we try to route demands such that this reduction in the maximum 2-route flow for the different ingress-egress pairs is as small as possible.

B. Maximum Flow versus Maximum 2-route Flow

Now we demonstrate the difference between maximum flow (maxflow) [2] and maximum 2-route flow by an example.

Consider the pair (S_2, D_2) in the network of Figure 1. The maxflow for this pair is 10 units, which is the same as the maximum 2-route flow for this pair. Now consider the pair (S_1, D_1). Note that the maxflow for this pair is 15 units, whereas the maximum 2-route flow for the pair is only 10 units. This simple example demonstrates the intuitively obvious result that maximum 2-route flow is always upper-bounded by the maxflow. In general, the maximum 2-route flow could be less than the maxflow, as in the case of the pair (S_1, D_1).

In the case of non-restorable demands the maxflow for an ingress-egress pair can be thought of as a measure "open capacity" between that particular ingress and egress [9], since it represents the maximum amount of traffic that can be routed between that ingress-egress pair. For example, we can route 15 units of non-restorable flow between S_1 and D_1. However, only 5 units of restorable demands can be routed and therefore the total amount of bandwidth used by the active and the backup path is only 10 units.

Clearly, maxflow in general is not a good indicator of the maximum amount of restorable flow that can be routed between an ingress-egress pair, and instead we need to determine maximum 2-route flow.

C. 2-Critical Links

The idea of 2-critical links is central to the development of the routing algorithms in this paper.

Definition 1: A link is defined to be 2-critical for an ingress-egress pair if the value of maximum 2-route flow for the ingress-egress pair decreases when the capacity of the link is decreased infinitesimally.

This definition is analogous to the definition of 1-critical links in the the case of non-restorable demands. These 1-critical links are links that belong to some mincut. (This follows directly from the maxflow- mincut theorem). By the definition

of 2-critical links, if the current restorable connection (active or backup path) is along a 2-critical link then the maximum 2-route flow between a particular ingress-egress pair will be reduced.

A link could be 2-critical for several different ingress-egress pairs. Since in general we are interested in keeping the maximum 2-route flows between all ingress-egress pairs as high as possible, we would like to avoid routing connections (active as well as backup paths) over these 2-critical links, to the extent possible. We now illustrate the importance of using 2-critical links for routing restorable flows. Consider the ingress-egress pair (S_1, D_1) in the network in Figure 1. Note that all of the links 1-4, 4-3, 1-2 and 2-3 are 1-critical for this pair, since the maxflow for this pair decreases if a connection is routed on any of these links. However, it is easy to see that only the links 1-4 and 4-3 are 2-critical for this ingress-egress pair, the links 1-2 and 2-3 are not.

Now consider a demand of 1 unit between S_2 and D_2, which requires an active as well as a backup path. Clearly, one amongst the active or the backup path has to be routed over 5-6-7, whereas the other one can be routed either over 1-4-3 or over 1-2-3. If we use the path 1-4-3, the total amount of demand that can be routed between S_1 and D_1 drops down by 1 unit. However, if the path 1-2-3 is used, the total amount of demand that can be routed between S_1 and D_1 remains the same. Therefore, we would like to use the path 1-2-3 and not 1-4-3. If we route based on 2-critical links for (S_1, D_1), we would pick the path 1-2-3 (recall that only the links 1-4 and 4-3 are 2-critical for (S_1, D_1), and we are routing so as to avoid the 2-critical links). However, if we route based on 1-critical links, both the paths 1-2-3 and 1-4-3 are equivalent (since all the links 1-4, 4-3, 1-2, 2-3 are 1-critical). Therefore, if routing is done on the basis of the 1-critical links, it is possible that the path 1-4-3 is chosen, and not 1-2-3. Therefore, as this example demonstrates, routing on the basis of 1-critical links may not provide good routes in the case of connections with backups. The reason for this is that 1-critical links are computed on the basis of maxflow which is, in general, not an indicator of the amount of demand that can be routed between an ingress-egress pair, as we have argued before.

In Sections IV & V, we will present efficient algorithms to determine the 2-critical links for any ingress-egress pair.

D. Path Computation

Once the 2-critical links are identified, we would like to avoid routing connections on 2-critical links as much as possible. It is easy to show that the problem of determining the path that minimizes the interference is NP-Complete. In order to approximate the solution to this problem we first define the *criticality index* of a link.

Definition 2: The criticality index of a link l denoted by $w(l)$, is defined as the number of source destination pairs for which the link is 2-critical.

We approximate the problem of determining the path set that minimizes interference to the problem of determining the path set (two disjoint paths) that minimizes the sum of the criticality indices of the links in the path set. We use the algorithm of Suurballe and Tarjan [14] to solve this disjoint path problem as outlined in Section IV.

III. Problem Statement

Let $G = (\mathcal{N}, \mathcal{L}, B)$ describe the given network, where $\mathcal{N}$ is the set of routers (nodes) and $\mathcal{L}$ the set of links (edges) and B the bandwidth of the links. Let n denote the number of nodes and m the number of links in the network. Assume that there are a set of distinguished node (router) pairs $\mathcal{P}$. These can be thought of as the set of potential ingress-egress router pairs. Therefore, all connection set-up requests (demands) are assumed to occur between these pairs. We denote a generic element of this set by (s, d). Let p denote the cardinality of the set $\mathcal{P}$. For each $(s, d) \in \mathcal{P}$, let $\mathcal{J}_{sd}$ denote the set of all edge-disjoint path pairs between source s and destination d. Therefore, $\mathcal{J}_{sd}$ is the set of path pairs over which a demand between (s, d) can be routed. Demands are assumed to arrive one at a time. The current demand is assumed to be of D units from source node a to the destination node b, where $(a, b) \in \mathcal{P}$. The objective is to determine a pair of link disjoint paths between a and b to route this bandwidth request of D units. We are interested in routing a demand over an edge-disjoint path pair such that the sum of the criticality indices of the links in the path pair is minimized. As explained in the previous section, this policy is based on the intuition that we want to route demands in such a way that they do not significantly decrease the maximum 2-route flows for the different ingress-egress pairs. Therefore, the optimal path pair j^* for a demand to be routed between (s, d) is obtained as

$$j^* = \arg \min_{j \in \mathcal{J}_{sd}} \sum_{l \in j} w(l) \tag{1}$$

One of paths of the optimal path pair can be used for the active path, and the other for the backup path. Note that since residual capacities change as demands are routed or terminated, the set of the 2-critical links for any ingress-egress pair changes.

The next two sections, present two different algorithms for the min-interference routing problem. Both the algorithms require the computation of the set of 2-critical links. It will be shown that the 2-critical links can be determined from the maximum 2-route flows. The first algorithm computes the maximum 2-route flows exactly by solving a sequence of maxflow problems. This allows us to compute the set of 2-critical links exactly. The second algorithm computes the maximum 2-route flows approximately by solving a sequence of shortest path problems. Since the problem is solved approximately, the algorithm determines only a set of approximately 2-critical links (that contains the true set of 2-critical links). The second algorithm is very simple to implement and enables us to trade off accuracy for computational complexity.

IV. A Routing Algorithm Based on Maxflow Computation

Here we present the exact algorithm. As mentioned, the routing algorithm is based on computing the set of 2-critical links, which in turn is based on computing maximum 2-route flows. Maximum 2-route flows are computed by computing certain maxflows, as we describe below.

A. Maximum 2-route Flow Computation

We use an algorithm developed by Kishimoto [11] which computes the maximum K-route flow between two nodes in a network by solving at most $(K+1)$ maxflow problems. The algorithm is fairly simple, and for $K=2$ (the case in which we are interested), the algorithm works as described below.

For any network $G=(\mathcal{N},\mathcal{L},B)$, let $G^u=(\mathcal{N},\mathcal{L},B^u)$ denote a network with the same set of nodes and edges as G, but where all edge capacities greater than u are set to u. Therefore, for any link $l\in\mathcal{L}$, $b_l^u=\min\{u,b_l\}$, where b_l and b_l^u are the capacities of edge l in G and G^u, respectively. Network G^u is therefore a capacity-bounded version of network G, where the capacity bound is u.

As stated in [1], an interesting fact that relates the maximum 2-route flow with maxflows is that if the capacity u is chosen appropriately, then a maxflow computed in the capacity-bounded network G^u is a maximum 2-route flow in the original network G. Moreover, if the value of the maximum 2-route flow in G is v^*, then the maxflow in the network G^{u^*} with $u^*=(v^*/2)$ is a maximum 2-route flow in G (see also Lemma 3 in Appendix I). Intuitively, the following algorithm can be viewed as one that tries to find the appropriate capacity bound, u^*. Initially, the capacity bound is set to half of the maxflow value in G. If the maxflow value in the resulting capacity-bounded network is the same as the maxflow value in the original network, then the capacity bound is the correct one. Otherwise, the capacity bound is revised (based on the maxflow values already computed, as stated below) so that it corresponds to the correct capacity bound. The following algorithm gives the maximum 2-route flow between two nodes s and d in the network.

Algorithm **max_2-route_flow**(s,d)

- Step 1. Compute the maxflow between (s,d). Let the maxflow value be v_0.
- Step 2. Set $u\leftarrow v_0/2$. Now compute the maxflow between (s,d) in network G^u. Let the maxflow be f_1 and its value be v_1. Note that $v_1\le v_0=2u$.
- Step 3. If $v_1=2u$, stop. Then f_1 is a maximum 2-route flow in G.
- Step 4. If $v_1<2u$, set $u\leftarrow(v_1-v_0/2)$. Now compute the maxflow between (s,d) in network G^u. Let the maxflow be f_2 and its value be v_2. Then f_2 is a maximum 2-route flow in G, and v_2 satisfies $v_2=2u=2v_1-v_0$.

The proof of correctness of the above algorithm can be found in [11]. The maxflow between two nodes in a network can be computed in time $O(n^2\sqrt{m})$ by the Goldberg Tarjan highest label preflow push algorithm [8]. Since this algorithm solves at most three maxflow problems, the running time of the algorithm is $O(n^2\sqrt{m})$.

B. 2-Critical Link Computation

The solution of the maximum 2-route flow is now used to compute the set of 2-critical links for the ingress-egress pair (a,b). The next theorem gives the conditions that a link needs to satisfy for it to be 2-critical. In the following, let f^* be a maximum 2-route flow between (s,d) in a network G, and its value be v^*. Let $u^*=(v^*/2)$. Then it is easy to see that f^* is a feasible flow in G^{u^*}. Let $G^{u^*}_{f^*}$ be the flow residual graph for flow f^* in G^{u^*}. Let $\mathcal{C}_{sd}$ denote the set of 2-critical links for $(s,d)\in\mathcal{P}$. Also, let $b_{(i,j)}$ denote the capacity of edge (i,j) in network G. A 2-critical link for (s,d) is characterized by the following theorem.

Theorem 1: An edge $(i,j)\in\mathcal{C}_{sd}$ if and only if both of the following conditions are satisfied

- $b_{(i,j)}\le u^*$.
- There is no path between i and j in $G^{u^*}_{f^*}$.

The theorem is proved in Appendix I. Assuming that $G^{u^*}_{f^*}$ and u^* are already known, to determine whether a link is 2-critical or not (on the basis of the above result) requires us to run a depth first search algorithm, and requires $O(m)$ time. Note that u^* and f^* are already known from **max_2-route_flow**(s,d). Therefore, the determination of the set of 2-critical links for (s,d) (by constructing the residual flow graph $G^{u^*}_{f^*}$ and checking the above set of conditions for each edge), requires an additional $O(m^2)$ time. In practice, however, the overall running time of the procedure of determining the 2-critical links will be dominated by the maximum 2-route flow computation.

C. Disjoint Path Computation

Once the set of 2-critical links are known, the link criticality indices are computed accordingly. Now we describe how we can compute the disjoint path pair with the least total criticality index (i.e., the disjoint path pair that satisfies (1)). This problem can be formulated as a minimum cost network flow problem where each link has unit capacity, and the cost of the link is the link criticality index as just computed. Also, there is a supply of 2 units at node s and a demand of 2 units at node d. Any standard min-cost flow algorithm can be used to solve this problem.

A very fast and simple algorithm for this min-weight (shortest) disjoint path problem is presented in [14] and is given here for the sake of completeness. The algorithm works as described below. In the algorithm, it is assumed that the length of any link l is set to its criticality index, $w(l)$.

Algorithm **shortest_disjoint_path**(s,d)

- Step 1. Determine the shortest path tree from node s. Let d_i represent the shortest path length from note s to node i. We replace the length of link $l=(i,j)$ with $w(l)-d(j)+d(i)$.
- Step 2. Let P_1 represent a shortest path from s to d. Reverse all the links on P_1 and leave all the lengths as computed in Step 1. Solve the shortest path problem between nodes s and d on this new graph with the new lengths. Let P_2 represent this shortest path.
- Step 3. If any of the reversed P_1 links belong to P_2, eliminate these links from P_1 and P_2 to form link sets $P_1^{'}$ and $P_2^{'}$. The set $P_1^{'}\cup P_2^{'}$ is the set of link disjoint optimal paths.

The proof of correctness of this algorithm can be found in [14]. Note that these algorithms requires us to solve two shortest path problems, which can be done in $O(m+n\log n)$ time.

D. Routing Algorithm

Note that in order to compute the link criticality indices, we need to compute the set of 2-critical links for all ingress-egress pairs in $\mathcal{P}$. Therefore, from the above discussion, we see that the running time of the overall algorithm is $O(p(n^2\sqrt{m}+m^2))$ (recall that p is the total number of ingress-egress pairs).

Note that if we are routing a restorable demand which requires D units of bandwidth on both the active and backup paths, we need to first eliminate all links with a residual capacity less than D units in the network, and then run the disjoint path finding algorithm in the new network thus obtained.

The main steps of the Mimumum-Interference Restorable Routing (MIRR) algorithm are given in the adjoining figure.

Remarks

- Ingress-egress pair (s,d) can have a weight of α_{sd} associated with it. Then the criticality index of a link can be defined as the sum of the weights of the ingress-egress pairs for which the the link is 2-critical. For the sake of simplicity, we have considered the special case where $\alpha_{sd}=1$ for all (s,d). The algorithm is also applicable in the more general case of dissimilar pair weights.
- The weights can be made inversely proportional to the maximum 2-route flow values, i.e,$\alpha_{sd}=1/v^*_{sd}$ where v^*_{sd} is the maximum 2-route flow value for the ingress-egress pair (s,d). This weighting implies that the 2-critical links for the ingress-egress pairs with lower maximum 2-route flow values will be weighted heavier than the ones for which the maximum 2-route flow value is higher.

Minimum Interference Restorable Routing (MIRR) algorithm

Input:

A graph $G(\mathcal{N},\mathcal{L})$ and a set B of residual capacities on all the edges. An ingress node a and an egress node b between which a flow of D units have to routed.

Output:

Two disjoint paths between a and b, each having a capacity of D units.

Algorithm:

1. Compute the maximum 2-route flows $\forall\ (s,d)\in\mathcal{P}$ using the procedure **max_2-route_flow**.
2. Compute the 2-critical link sets $\mathcal{C}_{sd}\ \forall\ (s,d)\in\mathcal{P}$ from the maximum 2-route flows obtained in the previous step, by using the conditions stated in Theorem 1.
3. Compute the criticality indices $w(l)\ \ \forall l\in L$.
4. Eliminate all links which have residual bandwidth less than D and form a reduced network.
5. Assume that $w(l)$ is the length of link l, and compute the optimal disjoint path pair between a and b using the procedure **shortest_disjoint_path**.
6. Choose one of the two paths obtained from the previous step as the active path, and route the demand of D units on that path. The other path is the backup path.

V. A Routing Algorithm Based on Shortest Path Computation

In this section we present a routing algorithm that does not require maxflow computation, unlike the algorithm presented in the last section. Instead, in this primal-dual algorithm, the maximum 2-route flow problem is solved as a sequence of shortest disjoint path problems. Note that shortest disjoint path pair computations (as described in **shortest_disjoint_path** in the previous section) require solving two shortest path problems, and are therefore are considerably simpler than maxflow computations. However, this algorithm computes the maximum 2-route flows only approximately, and therefore, the set of 2-critical links determined by this algorithm is approximate too. It is possible, however, to compute the maximum 2-route flow at any desired level of accuracy with increased running time. Therefore we can trade off accuracy for computation time.

A. Maximum 2-route Flow Computation

First we provide an alternative formulation of the maximum 2-route flow problem, in terms of a linear program. Then we will show how this linear program can be solved efficiently using shortest disjoint path problems.

Let (s,d) be the source-destination pair for which we want to compute the maximum 2-route flow. For each disjoint path pair $j\in\mathcal{J}_{sd}$, associate a variable x_j such that it represents the flow on j (i.e, the flow on *each* of the two disjoint paths of j). Therefore, the maximum 2-route flow problem can be formulated as

$$\mathbf{P}: \quad \text{maximize} \sum_{j\in\mathcal{J}_{sd}} x_j$$

$$\text{subject to}: \quad \sum_{j:l\in j} x_j \ \le\ b_l \qquad \forall l\in\mathcal{L}$$

Note that the objective function of the problem $\mathbf{P}$ is simply the total flow on the set of active paths (or the set of backup paths) between s and d, whereas the constraint for any link l simply states that the overall flow (active plus backup) on the link is less than the link capacity. Note that the number of disjoint path pairs in a graph can be very large, and therefore there can be a very large number of variables in the above linear program. However, in the solution we describe below, we do not need to keep track of the flows on each disjoint path pair, and the time complexity does not depend on the number of disjoint path pairs. The approach is based on a work by Garg and Konemann [7], in which they propose ϵ-approximation algorithms for multicommodity flows and some other fractional packing problems. This kind of approach applies in our case too, and allows us to develop a simple ϵ-approximation algorithm for the maximum 2-route flow problem.

Now let us look at the dual of the problem $\mathbf{P}$. For each edge $l\in\mathcal{L}$, associate a dual variable λ_l, which we will call the "dual length" of the edge. Then the dual of the problem is as follows

$$\mathbf{D}: \quad \text{minimize} \sum_{l\in\mathcal{L}} \lambda_l b_l$$

$$\text{subject to}: \quad \sum_{l\in j} \lambda_l \ \ge\ 1 \qquad \forall j\in\mathcal{J}_{sd}$$

$$\lambda_l \ \ge\ 0 \qquad \forall l\in\mathcal{L}$$

Let λ denote the vector of the dual lengths. The "length" of a disjoint path pair j, represented as $length_j(\lambda)$, is defined as the sum of the dual lengths of all links on j, i.e., $length_j(\lambda) = \sum_{l \in j} \lambda_l$. Let $\alpha(\lambda)$ denote the length of the "shortest" disjoint path pair, i.e., $\alpha(\lambda) = \min_{j \in \mathcal{J}_{sd}} length_j(\lambda)$. Now note that the whole constraint set in the problem $\mathbf{D}$ can be simply written as $\alpha(\lambda) \geq 1$. Then the dual problem $\mathbf{D}$ can be written as $\min\{\sum_l \lambda_l b_l | \alpha(\lambda) \geq 1, \lambda \geq 0\}$.

The approach presented in [7] is a primal-dual approach, and proceeds in iterations. At any iteration, both the primal flow variables as well as the dual length variables are updated in a certain way. We briefly describe how this approach works in our case.

Let ϵ be a small number, chosen appropriately, depending on the desired level of accuracy. As we will see later, the accuracy of the algorithm depends on ϵ as $(1-\epsilon)^2$. Let $\lambda_l(i-1)$ denote the dual length of any edge l at the beginning of the i^{th} iteration. Initialize the length of every edge as $\lambda_l(0) = \delta$, for some small constant δ (δ needs to be chosen on the basis of ϵ, as we will see later). Now at the i^{th} iterative step, we add flow on the disjoint path pair of length $\alpha(\lambda(i-1))$, i.e., the on the shortest disjoint path pair. The amount of flow added is equal to the minimum of the capacities (the actual capacities, not the residual ones) of all edges on the shortest disjoint path pair. Let b denote this minimum capacity. Then b units of flow is added on each of the two paths of the shortest disjoint path pair. Now the length of any edge l on the shortest disjoint path pair is updated as $\lambda_l \leftarrow \lambda_l(1 + \epsilon b/b_l)$ (the lengths of the rest of the edges, where no flow is added, are kept unchanged). This is the length of the arc for the next shortest path iteration. Therefore, the length of an edge increases as more and more flow is routed on an edge. As mentioned, we always add flow on the shortest disjoint path pair; intuitively, therefore, we are trying to avoid adding flows on paths which are more loaded. Note that the length of the shortest disjoint path pair, $\alpha(\lambda(i))$ is strictly increasing with the iteration number i. The algorithm stops at the iteration in which this length becomes equal to or exceeds 1 for the first time (recall that any feasible dual solution requires this length to be no less than 1).

Note that in every iterative step, we are adding flows on paths without any consideration of the amount of flow that has been already routed on the edges, or the edge residual capacities. Therefore, as we would expect, this algorithm, on termination, could produce an infeasible flow assignment. In the end, therefore, the flows are "scaled" down so that they correspond to feasible flows. We will see how this scaling factor needs to be chosen (depending on ϵ and δ) so that the resulting flow is close to optimal, and also feasible. We describe the steps in the algorithm:

Algorithm **approx_max_2-route_flow**(s, d)

Step 1. (Initialization) Choose an appropriate ϵ and δ. Set $\lambda_l \rightarrow \delta \ \ \forall l \in \mathcal{L}$.

Step 2. (Shortest disjoint path pair computation) Compute $\hat{j}$, the shortest disjoint path pair between s and d based on the lengths λ_l. If $\alpha(\lambda)$, the length of $\hat{j}$ is not less than 1, go to Step 4.

Step 3. (Flow and length update) Compute b, the minimum of the capacities of all links on $\hat{j}$. Add a flow of b on each of the two paths in $\hat{j}$. Also for of all links l on $\hat{j}$, multiply the dual length λ_l by $(1 + \epsilon b/b_l)$. Go back to Step 2.

Step 4. (Scaling) Find an appropriate scaling factor ν, and scale down the entire flow by ν so as to make the flow feasible.

The values of δ and ν are chosen as per Theorem 2 (see below). Note that computation of the shortest disjoint path pair (required in Step 2) can be done using the algorithm **shortest_disjoint_path** outlined in the previous section. Also note that the scaling (Step 4) is simply done by dividing the flows on all links by the scaling factor ν.

Now we will see how we need to choose the scaling factor ν and δ so that we obtain a flow that is feasible and also close to optimal. The result is stated in the following theorem. In the theorem, we assume that L denotes the maximum number of edges on a path over which flow is routed.

Theorem 2: Assume that ϵ is given. Now choose δ as $\delta = (1+\epsilon)((1+\epsilon)2L)^{-1/\epsilon}$, and $\nu = \log_{1+\epsilon} \frac{1+\epsilon}{\delta} = \frac{1}{\epsilon}(1 + \log_{1+\epsilon} 2L)$. Then the 2-route flow found by algorithm **approx_max_2-route_flow** is at least $(1-\epsilon)^2$ times the maximum 2-route flow. Moreover, with this choice of δ, the running time of the algorithm is at most $m\lceil \frac{1}{\epsilon} \log_{1+\epsilon} 2L \rceil T_{sp}$ where T_{sp} is the time required to compute the shortest paths between two nodes in the graph.

Note that since the approximation factor of the algorithm is $(1-\epsilon)^2$, we can achieve any desired accuracy be choosing a sufficiently small ϵ. Of course, choosing a smaller ϵ results is a higher running time, as the above result shows. Note that T_{sp} is $O(m + n \log n)$. Therefore, the overall running time of the algorithm is $O(m(m + n \log n)\lceil \frac{1}{\epsilon} \log_{1+\epsilon} 2L \rceil)$.

Theorem 2 can be proved by proceeding exactly in the same way as in the analysis presented in [7] for the maximum multicommodity flow problem. The proof is a straightforward extension of the proof outlined in [7] and is omitted for brevity.

B. 2-Critical Link Computation

Once the maximum 2-route flow between (s, d) has been computed, we could use the conditions stated in Theorem 1 to determine the set of 2-critical links for (s, d), as before. However, since the maximum 2-route flow computed in this case is only approximate, we can expect that the set of 2-critical links obtained by this procedure will also be approximate. Let $\hat{f}$ be the flow that **approx_max_2-route_flow** computes, and let $\hat{v}$ be the corresponding flow value (note that $\hat{v}$ is the value of the total flow, including active and backup flows, and is therefore equal to twice the primal objective function value). Set the capacity bound $\hat{u}$ as $\hat{u} = \hat{v}/2$. Then following Theorem 1, we could consider a link (i, j) to be 2-critical iff both the following conditions hold: i) $b_{(i,j)} \leq \hat{u}$, ii) there is no path between nodes i and j in $G_{\hat{f}}^{\hat{u}}$. However, some of the 2-critical links obtained based on these criteria may not be 2-critical at all, and we may also fail to include some links that are actually 2-critical.

Now note that the algorithm **approx_max_2-route_flow** not only finds a feasible and close-to-optimal primal solution, but it

also produces a feasible dual solution (note that on termination, the length of the shortest disjoint path pair is greater than 1, which means that the constraints of the dual problem **D** are satisfied). Let $\tilde{\lambda}_l$ denote the dual length of any edge l when the algorithm **approx_max_2-route_flow** terminates. Then $\tilde{v}$ be two times the dual objective function value, i.e., $\tilde{v} = 2\sum_{l\in\mathcal{L}} \tilde{\lambda}_l b_l$. Then, by weak duality, $\hat{v} \leq v^* \leq \tilde{v}$, where v^* is the maximum 2-route flow value. Define $\Delta v = \tilde{v} - \hat{v}$ to be the duality gap. The algorithm described below uses the primal solution, and the value of the dual solution, to obtain the set of 2-critical links. Note that in the algorithm, $G^{\hat{u}}_{\hat{f}}(\mathcal{N}, \mathcal{L} - (i,j))$ represents the graph constructed from $G^{\hat{u}}_{\hat{f}}$ by removing the edge (i,j).

Algorithm **approx_2-critical_links**(s,d)

Step 1. Find the maximum 2-route flow for (s,d) (approximately) using the algorithm **approx_max_2-route_flow**. Let $\hat{f}$ be the flow, $\hat{v}$ be its value. Also let $\tilde{v}$ be twice the dual value obtained.

Step 2. Set $\hat{u} = (\hat{v}/2)$, and $\tilde{u} = (\tilde{v}/2)$. Compute the graph $G^{\hat{u}}_{\hat{f}}$. Let $\hat{b}_{(i,j)}$ denote the capacity of edge (i,j) in $G^{\hat{u}}_{\hat{f}}$.

Step 3. For each edge $(i,j) \in \mathcal{L}$, include (i,j) in $\mathcal{C}_{sd}$ iff all of the following conditions hold:
i) $b_{(i,j)} \leq \tilde{u}$.
ii) $\hat{b}_{(i,j)} \leq \Delta v$.
iii) There is no path between i and j in $G^{\hat{u}}_{\hat{f}}(\mathcal{N}, \mathcal{L} - (i,j))$ with capacity greater than $(\Delta v - \hat{b}_{(i,j)})$.

The conditions i),ii),iii) of Step 3 are necessary conditions for an edge (i,j) to be 2-critical for (s,d) in network G. In other words, the algorithm **approx_2-critical_links** will find a superset of the actual set of 2-critical links. By comparing with the conditions in Theorem 1, it is easy to see that for the case $\Delta v = 0$ (i.e., the primal and the dual problems have been solved optimally), the algorithm **approx_2-critical_links** finds the exact set of 2-critical links.

Note that once $\hat{f}$ is known, the computation of the graph $G^{\hat{u}}_{\hat{f}}$ requires $O(m)$ time. The depth first search in **approx_2-critical_links** can also be solved in $O(m)$ time. Therefore, computation of the sets of 2-critical links for all ingress-egress pairs requires $O(m^2)$ time.

Once the sets of 2-critical links for all ingress-egress pairs have been computed, the link criticality indices are computed accordingly. Then the shortest disjoint path pair can be computed by algorithm **shortest_disjoint_path** outlined in the Section IV-C. As mentioned before, this algorithm runs in $O(m + n\log n)$ time.

Note that the overall routing algorithm in this case is similar to the one presented in Section IV-D, except for the methods of computing maximum 2-route flows and the 2-critical link sets. A high level view of this routing algorithm can be obtained from the algorithm MIRR outlined in Section IV-D, by replacing steps 1 & 2 appropriately, using the algorithms **approx_max_2-route_flow** and **approx_2-critical_links**. In the rest of this paper, we will refer to this routing algorithm as the approx_MIRR.

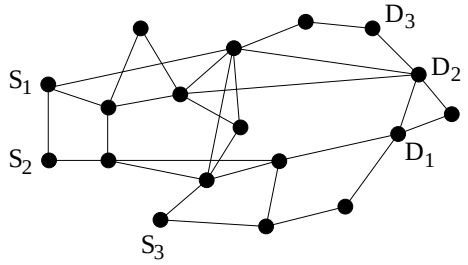

Fig. 2. Network 1 $(S_i, D_i),\ i = 1, 2, 3$ are the ingress-egress pairs)

VI. Simulation Results

In this section, we compare the algorithms proposed in this paper, MIRR and approx_MIRR, with a min-hop based routing algorithm for the same problem, as presented in [12]. In the algorithm described in [12], each demand is routed on a disjoint path pair for which the total number of links (hops) on the path pair is the minimum. This algorithm is, therefore, a natural generalization of the min-hop algorithm originally proposed for the case of routing no-backup connections. Note that the min-hop disjoint path pair can be computed by using the algorithm **shortest_disjoint_path** (after setting all the edge weights (lengths) to 1). As we demonstrate below, algorithms MIRR and approx_MIRR, where links are weighed according to their criticality indices, perform significantly better than this min-hop based routing algorithm.

The performance objective that we consider is the proportion of demands rejected by a routing algorithm. In all of the experimental results presented in this paper, we assume that demands arrive between each ingress-egress pair according to a Poisson process with an average rate λ, and the holding times are exponentially distributed with mean $\frac{1}{\mu}$. The bandwidth required by these demands are integral, and uniformly distributed between 1 and 3 units.

Figure 2 shows the network we consider (experimentation on 3 other networks also yielded similar results). The network, taken from [4], has 18 nodes and 30 links, and represents a typical ISP network. All link capacities are assumed to be 20 units. Figures 3-5 show the proportion of rejected demands under different load conditions, namely, $\rho = \frac{\lambda}{\mu}$ = 5, 7, 9. The proportion of demands rejected by the min-hop based algorithm under low, moderate and high load conditions are roughly 10%, 20% and 30%, respectively. The rejection ratios are shown for 20 different trials, where each trial corresponds to a different demand pattern (generated by a different random seed). The rejection ratio is measured over a window of 20000 demands.

Let us take a closer look at Figure 3. Firstly note that the rejection ratio in MIRR is significantly better than min-hop in all the trials. Note that MIRR and approx_MIRR with ϵ = 0.1 perform very closely. Also, min-hop and approx_MIRR with ϵ = 0.5 perform very closely. Moreover, note that the rejection ratio for approx_MIRR with ϵ = 0.3 is greater than that of approx_MIRR with ϵ = 0.1, but less than that of approx_MIRR with ϵ = 0.5. An examination of Figures 4 & 5 (rejection ratios under higher load) also reveals a similar trend.

Figures 3-5 demonstrate that the performance of MIRR is significantly better than the min-hop based algorithm. The plots

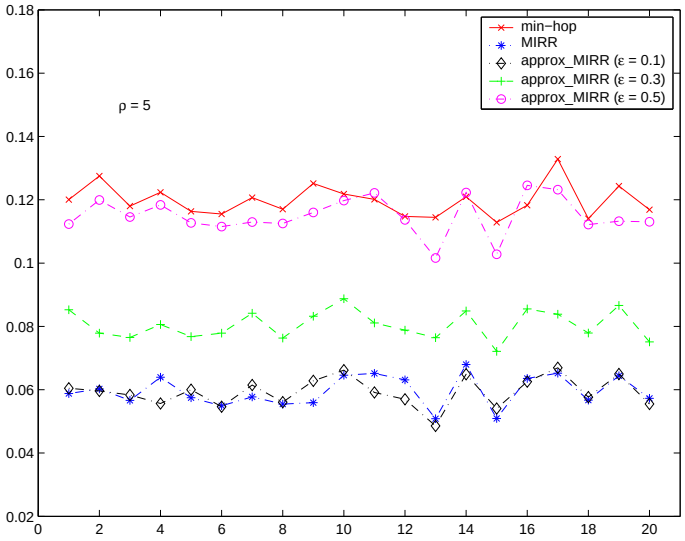

Fig. 3. Rejection ratio under low load ($\rho = 5$).

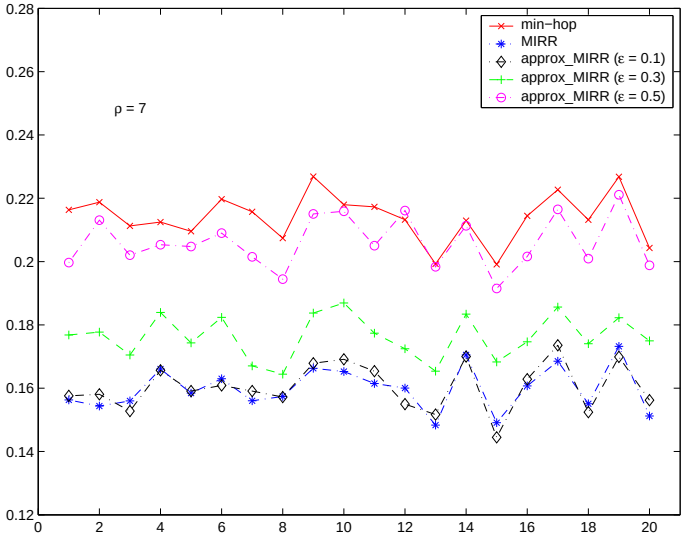

Fig. 4. Rejection ratio under moderate load ($\rho = 7$).

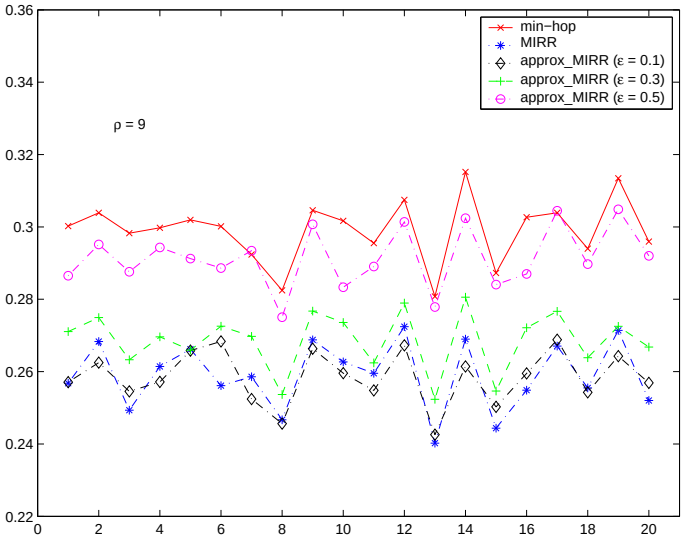

Fig. 5. Rejection ratio under high load ($\rho = 9$).

ρ	MIRR	approx_ MIRR ($\epsilon = 0.1$)	approx_ MIRR ($\epsilon = 0.3$)	approx_ MIRR ($\epsilon = 0.5$)
5	0.50	0.50	0.33	0.04
7	0.25	0.25	0.18	0.04
9	0.13	0.13	0.10	0.03

TABLE I

AVERAGE RELATIVE IMPROVEMENT IN REJECTION RATIOS (COMPARED TO MIN-HOP)

also demonstrate that for small values of the approximation parameter ϵ, approx_MIRR performs similar to MIRR, whereas for large values of ϵ, its performance degrades and becomes similar to min-hop. This is intuitively expected, since ϵ is a measure of the accuracy of the computation of the maximum 2-route flows (and therefore, it is also a measure of the accuracy of the computation of the set of 2-critical links).

Table I shows the average relative improvement in the rejection ratios (with respect to the min-hop based algorithm) under MIRR and approx_MIRR for our network, and for various values of $\rho = \frac{\lambda}{\mu}$. Relative improvement of MIRR compared to min-hop is computed as (rr(min-hop) - rr(MIRR))/rr(min-hop), where rr(min-hop) and rr(MIRR) are the rejection ratios under min-hop and MIRR, respectively. Relative improvement of approx_MIRR over min-hop is also defined similarly. The averaging is done over 20 independent trials, as shown in Figures 3-5.

The results show the quantitative performance improvement of MIRR over min-hop. They also demonstrate that the performance of approx_MIRR degrades smoothly from MIRR to min-hop as the value of ϵ increases. Note that a larger ϵ also implies a lower running time. Therefore, the algorithm approx_MIRR provides a tradeoff between performance and computation time. This allows us to run the algorithm at a suitable operating point, determined on the basis of the desired performance level and the computational time budget.

VII. CONCLUDING REMARKS

The need for restorability introduces many new and interesting issues to the well-studied area of QoS routing. With restorability being a necessity in optical networks and an important aspect of MPLS networks, dynamic routing of wavelengths or bandwidth guaranteed paths with restorability is an important area of study. Restorable connections have an active and a disjoint backup path, with the backup path being possibly shared for efficiency. In several circumstances, backup path sharing may not be possible. An important one is the case where data is simultaneously sent on both paths to facilitate very fast recovery upon failure. When sharing is not possible, performance efficiencies can be obtained only by good path selection. In this paper, we developed two new algorithms, that use the minimum-interference criteria, for efficiently routing restorable connections. As was shown in the paper, these

algorithms are not simple extensions of miniumum-interference based routing algorithms for non-restorable connections. One of the new algorithms uses only shortest path computations and is easy to implement. Both the proposed algorithms perform very well and improve on the previously proposed least-total-cost disjoint path routing algorithm.

APPENDIX I: PROOF OF THEOREM 1

We first state a lemma that will be used in the proof.

Lemma 3: Let the value of maximum 2-route flow for (s, d) in G be v^*. Let $u^* = (v^*/2)$. Then a maximum 2-route flow for (s, d) in G is a maximum flow for (s, d) in G^{u^*}, and vice versa.

The above lemma follows straightforwardly from Lemma 2 of [1], and we will not prove it here. Now we state the proof of Theorem 1.

Proof of Theorem 1:

(Necessity) To see that the first condition is necessary, consider the flow f^* in G. Note that the flow on edge (i, j) (corresponding to f^*) can be no greater than u^*. Therefore, if $b_{(i,j)} > u^*$, an infinitesimal reduction of the capacity of (i, j) does not reduce the maximum 2-route flow. So, necessity of first condition, $b_{(i,j)} \leq u^*$, is obvious.

For the second condition, consider the flow f^* in G^{u^*}. Note that from the previous lemma, f^* is a maxflow in G^{u^*}. If there exists a path between i and j in $G^{u^*}_{f^*}$, then we could reduce the flow on edge (i, j) (corresponding to f^*) by a small amount and route it on that path. The new flow thus obtained has the same value as f^*, and is therefore a maxflow in G^{u^*}. From Lemma 3, it is also a maximum 2-route flow in G. Therefore, if there exists a path between i and j in $G^{u^*}_{f^*}$, reduction of the capacity of edge (i, j) infinitesimally does not cause a reduction in the maximum 2-route flow in G. Therefore, the second condition is also necessary.

(Sufficiency) Assume that the two conditions stated in the theorem hold. We need to show that the edge (i, j) is 2-critical for (s, d) in G.

First we argue that edge (i, j) belongs to a minimum cut for (s, d) in G^{u^*}. Note that in the flow residual graph $G^{u^*}_{f^*}$, edge (i, j) must be filled to capacity, otherwise a path would exist between i and j, trivially (the edge (i, j) itself constitutes the path). Since all edge capacities are assumed positive, this implies that there is a positive flow on edge (i, j). This implies that there must be a path from i to s, and a path from t to j, in $G^{u^*}_{f^*}$. Now note that if there exists a path from s to j in $G^{u^*}_{f^*}$, then this would imply the existence of a path between i and j in $G^{u^*}_{f^*}$, thus contradicting our assumption. Therefore, there cannot be a path from s to j in $G^{u^*}_{f^*}$. By a similar argument, it follows that there cannot exist a path between i and t in $G^{u^*}_{f^*}$.

Now let $\mathcal{S}$ represent the set of nodes reachable from source s in $G^{u^*}_{f^*}$. From the above discussion, $j \notin \mathcal{S}$. Also, since f^* is a maxflow in G^{u^*} (by Lemma 3), it follows that $t \notin \mathcal{S}$ (else there would have been an augmenting path in $G^{u^*}_{f^*}$). Let $\mathcal{I}$ represent the set of nodes reachable from i in the residual graph. Let $\mathcal{S}' = \mathcal{S} \cup \mathcal{I}$. Note that both j and t are not in $\mathcal{S}'$. Therefore edge (i, j) belongs in the minimum cut $(\mathcal{S}', \mathcal{N} \backslash \mathcal{S}')$.

Now assume, for the sake of contradiction, that (i, j) is not 2-critical for (s, d) in G. Let $\hat{G}$ represent the graph formed by reducing the capacity of (i, j) in G by a small positive number ϵ (all other edge capacities in G are left unchanged). Since edge (i, j) is not 2-critical (by assumption), the value of the maximum 2-route flow in $\hat{G}$ is v^* (recall that the maximum 2-route flow does not decrease if the capacity of a non-2-critical link is decreased by a small amount). Therefore, from Lemma 3, the maxflow value in $\hat{G}^{u^*}$ must *be equal to* v^*. Now let us compare the the graphs G^{u^*} and $\hat{G}^{u^*}$. Since $b_{(i,j)} \leq u^*$, $\hat{G}^{u^*}$ can be constructed from G^{u^*} by decreasing the capacity of (i, j) in G^{u^*} by ϵ (leaving the capacities of all other edges in G^{u^*} unchanged). Since (i, j) belongs to a mincut on G^{u^*} (as we have shown above), it follows from the maxflow-mincut theorem that the maxflow value in $\hat{G}^{u^*}$ the strictly less than the maxflow value in G^{u^*} (by the maxflow-mincut theorem, the maxflow value decreases whenever the capacity of the mincut is decreased). Note that by Lemma 3, the maxflow value in G^{u^*} is equal to v^*. Therefore, it follows that the maxflow value in $\hat{G}^{u^*}$ must *be strictly less than* v^*. Therefore, we arrive at a contradiction, proving that (i, j) is 2-critical for (s, d) in G. ∎

REFERENCES

[1] C. C. Aggarwal, J. B. Orlin, "On Multi-Route Maximum Flows in Networks Networks", To appear in *Networks*. Sloan School Working paper 3954-97, May 1997.

[2] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, "Network Flows: Theory, Algorithms, and Applications", Prentice Hall, 1993.

[3] V. Anand, C. Qiao, "Dynamic Establishment of Protection Paths in WDM Networks, Part I", *Proc. of the 9th International Conference on Computer Communications (ICCCN '00)*, October 2000.

[4] G. Apostopoulos, R. Guerin, S. Kamat, A. Orda, and S. K. Tripathi, "Quality of Service Based Routing: A Performance Perspective", *Proc. of Sigcomm '98*, September 1998.

[5] B. Davie, Y. Rekhter, "MPLS Technology and Applications", Morgan Kaufman Publishers, 2000.

[6] B. T. Doshi, *et al.*, "Optical Network Design and Routing ", *Bell Labs Technical Journal*, Vol. 4, No. 1, pp.58-84, 1999.

[7] N. Garg, J. Konemann, "Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems", *Proc. of the 39th Annual Symposium on Foundations of Computer Science (FOCS '98)*, November 1998.

[8] A. V. Goldberg, R. E. Tarjan, "Solving Minimum Cost Flow Problem by Successive Approximation", *Proc. of the 19th ACM Symposium on the Theory of Computing*, pp.7-18, 1987.

[9] 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.

[10] D. Katz, D. Yeung, "Traffic Engineering Extensions to OSPF", *work in progress, Internet Draft*, 1999.

[11] W. Kishimoto, "A Method for Obtaining the Maximum Multi-route Flows in a Network", To appear in *Networks*.

[12] M. Kodialam, T. V. Lakshman, "Dynamic Routing of Bandwidth Guaranteed Tunnels with Restoration", *Proc. of Infocom '00*, March 2000.

[13] V. Sharma, et. al., "Framework for MPLS based Recovery", *Internet draft < draft-ietf-mpls-recovery-frmwrk-04.txt >*, July 2001.

[14] J. W. Suurballe, R. E. Tarjan, "A Quick Method for Finding Shortest Pairs of Disjoint Paths", *Networks*, Vol. 14, 1984, pp. 325-336.