Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications

Koushik Kar, Murali Kodialam, *Member, IEEE,* and T. V. Lakshman, *Senior Member, IEEE*

***Abstract*—This paper presents new algorithms for dynamic routing of bandwidth guaranteed tunnels, where tunnel routing requests arrive one by one and there is no *a priori* knowledge regarding future requests. This problem is motivated by service provider needs for fast deployment of bandwidth guaranteed services. Offline routing algorithms cannot be used since they require *a priori* knowledge of all tunnel requests that are to be routed. Instead, on-line algorithms that handle requests arriving one by one and that satisfy as many potential future demands as possible are needed. The newly developed algorithms are on-line algorithms and are based on the idea that a newly routed tunnel must follow a route that does not "interfere too much" with a route that may be critical to satisfy a future demand. We show that this problem is NP-hard. We then develop path selection heuristics which are based on the idea of deferred loading of certain "critical" links. These critical links are identified by the algorithm as links that, if heavily loaded, would make it impossible to satisfy future demands between certain ingress–egress pairs. Like min-hop routing, the presented algorithm uses link-state information and some auxiliary capacity information for path selection. Unlike previous algorithms, the proposed algorithm exploits any available knowledge of the network ingress–egress points of potential future demands, even though the demands themselves are unknown. If all nodes are ingress–egress nodes, the algorithm can still be used, particularly to reduce the rejection rate of requests between a specified subset of important ingress–egress pairs. The algorithm performs well in comparison to previously proposed algorithms on several metrics like the number of rejected demands and successful rerouting of demands upon link failure.**

***Index Terms*—Maximum flow, MPLS, optimization, quality of service routing, traffic engineering.**

I. Introduction

We consider the problem of setting up bandwidth guaranteed tunnels in a network, where tunnel setup requests arrive one by one and future demands are unknown. The only dynamic information available to the tunnel routing algorithms are the link residual capacities which can be obtained from routing protocol extensions such as in [10], [13], [15]. We also assume that quasi-static information such as the ingress–egress nodes in the network are known. This knowledge of the ingress–egress nodes should be exploited to reduce the number of request rejections due to insufficient network capacity. Even when all nodes are ingress–egress nodes, it is likely that some subset of ingress–egress nodes are more important, and so tunnel requests between them will be required to have a lower probability of being rejected. The algorithm should be able to protect such important ingress–egress pairs.

We develop new algorithms for routing bandwidth guaranteed tunnels in this scenario. The problem is motivated by the needs of service providers to quickly setup bandwidth guaranteed paths in their backbone or transport networks. An important context in which these problems arise is that of dynamic label switched path (LSP) setup in multiprotocol label switched (MPLS) networks. For conciseness and ease of terminology, we focus on this application in the rest of the paper, even though the developed algorithms can be used in other networking applications requiring dynamic bandwidth provisioning.

A. MPLS

In MPLS [14], packets are encapsulated, at ingress points, with labels that are then used to forward the packets along label switched paths (LSPs). Service providers can use bandwidth guaranteed LSPs as components of an IP virtual private network (VPN) service, the bandwidth guarantees being used to satisfy customer service-level agreements (SLAs). 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" (FECs) [14], [4]. The classification into FECs is done using packet filters that examine header fields such as sources address, destination address, type-of-service bits, etc. The filter rules determining the FECs can be established in a variety of ways such as by downloading from a policy or route server, or by interacting with routing protocols. The purpose of classifying packets into FECs is to enable the service provider to traffic engineer the network and route each FEC in a specified manner. This is done by mapping arriving packets belonging to an FEC to one of the LSPs associated with the FEC.

B. Explicit Routing of LSPs for Traffic Engineering

Before mapping packets onto an LSP, the LSP is setup using a signaling protocol such as RSVP or CR-LDP (constraint routing label distribution protocol). A key aspect of LSPs relevant to this paper is that LSPs can be explicitly routed along

Manuscript received October 15, 1999; revised April 15, 2000.

K. Kar is with the ECE Department, University of Maryland, College Park, Maryland, MD 20742 USA (e-mail: koushik@eng.umd.edu).

M. Kodialam and T. V. Lakshman are with Bell Labs Lucent Technologies, Holmdel, NJ 07733 USA (e-mail: muralik@bell-labs.com; lakshman@research.bell-labs.com).

Publisher Item Identifier S 0733-8716(00)09228-3.

specific paths. This means that when an LSP is being setup, it is possible to specify all intermediate points between the ingress and egress.[1]

This explicit (or strict) routing feature of MPLS allows the potential addressing of many shortcomings associated with current IGP routing schemes, which are hampered by the requirement of forwarding packets based only on destination addresses (such as shortest path routing with mostly static and traffic-characteristic independent link metrics). A prime problem is that some links on the shortest path between certain ingress–egress pairs may get congested while links on possible alternate paths remain free. Even in the best-effort model, this means that available network resources are not being used well and there is potential for providing better quality of service with the same network infrastructure. In MPLS networks, link congestion caused by shortest path IGP-like routing of LSPs can cause LSP setup requests to be rejected even though these requests may have been admissible using a different routing scheme. Therefore, routing schemes that can make better use of network infrastructure are needed. Efficient network usage is the key purpose of network traffic engineering and it has been suggested that one of the most significant initial applications of MPLS will be in traffic engineering [3]. RSVP extensions to support explicit routing by incorporating an EXPLICIT_ROUTE object into RSVP Path messages has been proposed in [2]. The intent is to allow the MPLS network to be able to control the path from ingress node to egress node, and therefore optimize the utilization of network resources and enhance performance.

C. Bandwidth Guaranteed LSPs

Another key aspect of MPLS is that the signaling mechanisms for LSP setup will permit specification of quality-of-service attributes for the LSP. This is already inherent to RSVP (refer to [2] for a discussion of RSVP extensions relevant to MPLS). In this paper, we mostly consider only the setting up and routing of LSPs with bandwidth guarantees (which we merely refer to as LSPs in the rest of the paper). This does not mean that SLAs cannot incorporate other metrics such as delay and losses. We concentrate on bandwidth routing because we think that the most common traffic engineering usage of LSPs will be to setup bandwidth guaranteed paths. If QoS constraints such as delays and losses are to be incorporated in SLAs, the most practical way of handling 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 computationally difficult and requires information that are difficult to acquire (such as nodal load versus delay characteristics). Note that if the SLA provides only bandwidth guarantees, then there is no need for the initial conversion of SLAs into effective bandwidths. Even though we discuss the routing problem with only bandwidth constraints, it is possible to incorporate various policy, hop-count, and delay constraints within the bandwidth-routing framework. We briefly mention some possible approaches, but that is not the focus of this paper.

[1]There is also an option for partial specification of routes. However, in this paper we restrict attention to the case where the routes are fully specified.

II. Requirements for MPLS Routing Algorithms

We first try to identify the main requirements that a path selection algorithm for MPLS must satisfy in order to be useful in practice.

1) *Necessity to use on-line algorithms*. For traffic engineering purposes, it is usually assumed that all point-to-point demands are known. While this is a valid assumption for network design, for MPLS applications this implies that all LSPs that traverse the network are known at the time of initial routing. This is unlikely to be the case in practice. Furthermore, in this offline (all LSPs are known) model, the objective usually is to make the most efficient use of the network, i.e., to minimize the resource usage for the LSPs that are being routed. Note that with this objective, it may happen that there is no available capacity between certain ingress–egress routers after the routing has been done (even though a different routing may have resulted in some available capacity between those routers). In the offline model, this lack of residual capacity between certain ingress–egress pairs is not relevant since all LSPs that need to be routed are known and no future routing requests are expected. If any new LSPs are to be routed, this may require rerouting of existing LSPs. It is unlikely that existing LSPs will be rerouted except upon link failures (and perhaps for relatively rare network reoptimization). In practice, since the possibility of having to route future LSP demands cannot be excluded, the routing algorithm must be an on-line algorithm capable of routing requests in an "optimal" manner when the requests are not all presented at once and rerouting of existing LSPs is not allowed.
2) *Use knowledge of ingress–egress points of LSPs*. Even though future demands may be completely unknown, the routers where LSPs can potentially originate and terminate are known since these are the network's edge routers. The algorithm must be able to use any available knowledge regarding ingress–egress pairs and must not always assume that every router can potentially be an ingress and egress point (though this may be the case sometimes). To our knowledge, the algorithm we present is the first algorithm to take ingress–egress information explicitly into account.
3) *Good rerouting performance upon link failure*. This is clearly an important performance metric. When a link fails, it must be possible to find alternate routes for as many LSPs as possible. If, before failure, certain ingress–egress pairs have no residual capacity available between them, then rerouting LSPs between these pairs after link failure is not possible. The algorithms that we present try to maximize some surrogate measure of the residual capacity between the ingress–egress pairs and this makes them perform well upon link failure.

4) *Routing without traffic splitting*. Although splitting is used for load balancing purposes (by routing demands over multiple LSPs at the ingress point), it is not permissible for the routing algorithm to always split traffic in an arbitrary manner since the traffic being routed may be inherently unsplittable. Hence, the algorithm must be able to route a desired amount of bandwidth between a given ingress–egress pair without being able to split traffic onto multiple paths in an arbitrary way at every potential router in the path even though such splitting could permit better network usage.
5) *Computational requirements*. We show later that the optimal routing in our formulation is NP-hard. Any heuristic or approximation algorithm must be implementable on routers and route servers and must execute within a reasonable time-budget for networks with a few thousand ingress–egress pairs.[2]
6) *Feasibility of distributed implementation*. Even though the paper presents the route computation as being done in a centralized route server (which we believe is a viable option for intradomain MPLS routing), it is desirable that the algorithm be amenable to distributed implementation where each LSPs explicit route is computed at the local ingress router without communication with a domain or area wide route server. Hence, it is desirable for the algorithm to restrict its use of dynamic information to information derivable from current routing protocols or their extensions. The algorithm we propose uses topology information and residual capacities on links. This is the same information that even min-hop routing with bandwidth guarantees will require. Within an OSPF area, the topology information can be derived from the link state database and residual capacities can be obtained if extensions such as those suggested in [10], [13], [15] are implemented. We also use the possible set of ingress–egress pairs. This information is quasi-static and we take that to be provisioned information (note that the algorithm is applicable even if this information is unavailable because then every router can be assumed to be a potential ingress and egress).
7) *Information useful for aggregation*. While routing over multiple areas or multiple domains, the algorithm should, if possible, generate information that can be useful for aggregation of QoS metrics. Although this is not a necessity, it is a desirable feature and our algorithm takes this into account.
8) *Reoptimization*. The on-line routing objective must permit optimization of existing LSPs' routes by using the same objective. Although frequent rerouting (as would happen with offline algorithms) is not permissible, it may be acceptable to occasionally reroute existing LSPs to optimize routing so as to carry more traffic. This optimization is possible because the on-line route selection happened with less information than that available later, when the set of LSPs that have already been setup is known. Note that this optimization cannot use an offline algorithm since it still has to account for future arrivals. If occasional optimization is desired, the on-line algorithm's path selection objective must be such that a consistent optimization is possible (this is explained later in the paper).
9) *Policy constraints*. The algorithm must be able to incorporate common policy constraints such as policy restrictions on the type of links or routers that are permissible for routing a given LSP.
10) *Other requirements*. The algorithm must be able to accommodate requirements such as preemption and setup priorities. A detailed specification of requirements is in [3].

[2]The number of edge routers may be large but edge routers are usually connected to the same core router. This router can be taken as the ingress–egress point for the core MPLS network, and we do not expect more than a few thousand ingress–egress pairs in a core backbone network.

III. Problem Definition and System Model

We consider a network of n routers. A subset of these routers is assumed to be ingress–egress routers between which LSPs can be setup. However, it is not necessary that there be a potential LSP between every ingress and every egress. Instead, from a certain ingress, LSPs may be allowable only to certain egresses. This may be because of policy or service constraints (such as certain VPN traffic may only originate and exit at certain ingress–egress pairs). We assume that any such information is known, changes not very frequently, and is made available to the route server (we describe for simplicity only a centralized route computation in the paper) by a provisioning or administrative mechanism.

Each request for an LSP setup arrives at a route server which determines the explicit route for the LSP. The request either arrives directly to the route server (if the LSPs are being setup manually) or may first arrive at ingress routers which then query the route server to generate the explicit route (details of protocols that may be used for this interaction are left out for conciseness as also are details of mechanisms to initialize definitions of FECs). The explicit route is then communicated back to the ingress router which then uses a signaling mechanism such as RSVP or LSP to setup the path to the egress and to reserve bandwidth on each link on the path.

For calculating the explicit route, the route server needs to know the current topology and available capacities. We assume the topology is either known administratively or that a link state routing protocol is operational and that its link-state database is accessible. The algorithm keeps track of available capacities and we assume that the initial link capacities are known (this is for descriptive purposes only; if routing protocol extensions allow the determination of residual bandwidths, then that information can be used). Failure of LSPs due to link failures is detected from signaling protocol (CR-LDP or RSVP) information by the edge routers. They can request a rerouting of the LSPs after the link-state database has been updated by routing protocols or by other means. (An alternative, not studied in the paper, is to setup a disjoint path backup LSP so that failures can be accommodated by changing the FEC to LSP mapping at the ingress routers.)

We consider the request for an LSP i to be defined by a triplet (o_i, t_i, b_i). The first field o_i specifies the ingress router, the second field t_i specifies the egress router, and the third field b_i specifies the amount of bandwidth required for LSP i. We assume that the QoS requirements have been translated into an effective bandwidth requirement. We assume that requests for LSPs come in one at a time and there is no knowledge of the characteristics of future demands. The objective is to determine a path (if one exists) along which each demand for an LSP is routed so as to make "optimal" use of network infrastructure.

IV. Current State of the Art

We define the residual bandwidth along a link to be the difference between the bandwidth of the link and the sum of the LSP demands that are routed on that link. First note that a new LSP can be routed along a given link only if the residual bandwidth on that link exceeds the bandwidth requested by the new LSP. These links will be referred to as feasible links (with respect to the given LSP demand). The network consisting of all the routers and just the feasible links will be referred to as the feasible network. Therefore, when performing the routing, we can restrict our attention to paths in the feasible network. Note that if the ingress and the egress routers are disconnected in the feasible network, then there is no path that has the desired bandwidth and the LSP request is rejected. The objective of any routing scheme is to reject as few demands as possible.

The most commonly used algorithm for routing LSPs is the min-hop algorithm [2]. In this algorithm, the path from the ingress to the egress with the least number of feasible links is chosen. This algorithm, although simple, uses the same information that the proposed new algorithms use. Its performance in terms of efficient network usage can be easily improved upon with a little more computation. With the rapid rise in processor speeds and with traffic trunks not expected to be setup and torn down at high rates, it is justifiable to tradeoff increased computation for more efficient network usage. The min-hop algorithm does not take information on ingress–egress pairs into account nor does it adapt routing to increase chances of successful rerouting upon link failure.

A min-hop like algorithm which attempts to load balance the network traffic is proposed in [11]. This widest-shortest path algorithm (WSP) finds a feasible min-hop path between ingress and egress such that the chosen min-hop path has the maximum residual path bottleneck link capacity. This algorithm too does not take ingress–egress information into account. With every node being assumed to be a potential ingress and egress point, a widest path algorithm without a min-hop restriction does not work well since long paths which increase network usage get chosen. Hence, a widest-shortest path hop heuristic performs better. However, if the ingress–egress pairs are known, then as we shall see later, an algorithm which picks paths longer than min-hop works well provided it avoids unnecessarily loading certain "critical" links.

More sophisticated algorithms in addition use the residual bandwidth on the link to influence the weight of the link and the shortest path is chosen with respect to these dynamically changing weights [16]. Since the weights are chosen to increase with link load, the idea is not to use up link capacity completely if alternate lower loaded paths are available. This has the tendency to make capacity available for future demands. Nevertheless, this algorithm is also oblivious to information regarding ingress–egress pairs, and therefore can pick long paths to defer loading on links which may not be important to satisfy future demands.

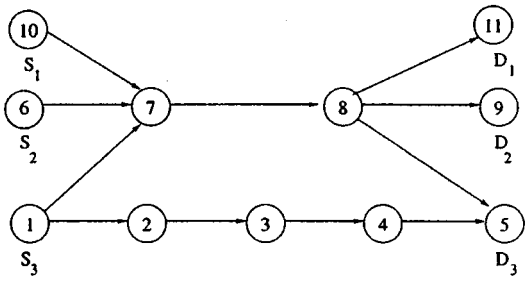

Fig. 1. Illustrative example.

In [12], a mathematical programming formulation is presented for on-line routing taking ingress–egress pairs into consideration. The case of both split and unsplit routing of bandwidth requests was presented. In this paper, instead of solving a linear or integer program, the approach is to route using a shortest path computation on an appropriately weighted graph. This is described in the next two sections.

V. Key Ideas for Minimal Interference Routing Algorithm

In this section, we give an informal description of the key ideas used in our routing algorithm. The next section has a more formal mathematical description and also a proof of NP-hardness of this formulation of the MPLS routing problem. The NP-hardness justifies use of a heuristic algorithm in the absence of a known approximation algorithm.

A. Illustrative Example

As seen in the previous section, existing schemes take into account the topology of the network and the residual capacities on the links, but do not take into account the location of the ingress/egress routers. These routers serve as sources and destinations of future traffic. If routing is done oblivious to the location of these sources and destinations of traffic, then we may "interfere" with the routing of some future demands. We illustrate this with a simple example. Consider the network shown in Fig. 1. There are three potential source destination pairs, (S_1, D_1), (S_2, D_2), (S_3, D_3). Assume that all links have a residual bandwidth of 1 unit. We now have a request for an LSP between S_3 and D_3 with a bandwith request of 1 unit. If min-hop routing is used, the route will be 1-7-8-5. Note that this route blocks the paths between S_2 and D_2 as well as S_1 and D_1. In this example, it is better to pick route 1-2-3-4-5 even though the path is longer.

B. Interference

The key idea is to pick paths that do not interfere too much with potential future LSP setup requests (demands) between

other source destination pairs. We first have to make this concept of interference more concrete. Consider the maximum flow (maxflow) value [1] v_1 between a given ingress–egress pair (S_1, D_1). This maxflow value is an upper bound on the total amount of bandwidth that can be routed between that ingress–egress pair (S_1, D_1). Note that maxflow value v_1 decreases whenever a bandwidth demand of D units is routed between S_1 and D_1. Note that the value of v_1 can also decrease when an LSP is routed between some other ingress–egress pair. We define the amount of interference on a particular ingress–egress pair, say (S_1, D_1), due to routing an LSP between some other ingress–egress pair as the decrease in the value of v_1.

C. Minimum Interference Paths

With interference defined as above, we can think of a minimum interference path for an LSP between, say (S_1, D_1), as that explicit route which maximizes the minimum maxflow between all other ingress–egress pairs. Intuitively, this can be thought of as a choice of path between (S_1, D_1) that maximizes the minimum open capacity between every other ingress–egress pair. Although this formulation has intuitive appeal, it has the drawback that it is the minimum maxflow that impacts the routing irrespective of values of the other maxflows. Hence, another objective might be to pick a path that maximizes a weighted sum of the maxflows between every other ingress–egress pair. We formalize these notions in the next section. These formulations not only make capacity available for the uncertain but possible arrival of future demands, but also make capacity available for rerouting of LSPs in case of link failures (this is illustrated by experimental results in Section VIII).

D. Critical Links

To progress from the notion of minimum interference paths to a viable routing algorithm that uses familiar max-flow and shortest path algorithms, we need the notion of "critical links." These are links with the property that whenever an LSP is routed over those links, the maxflow values of one or more ingress–egress pairs decreases. The next section gives an algorithm to determine critical links within a reasonable amount of computation (i.e., remains within a small target time budget for networks with hundreds of routers and thousands of ingress–egress pairs).

E. Path Selection by Shortest Path Computation

Once the critical links are identified, we would like to avoid routing LSPs on critical links to the extent possible. Also, we would like to utilize the well-used Dijkstra or Bellman–Ford algorithms to compute the actual explicit route. We do this by generating a weighted graph where the critical links have weights that are an increasing function of their "criticality" (see the next section for the actual link weight functions). The increasing weight function is picked to defer loading of critical links whenever possible. The actual explicit route is calculated using a shortest path computation as in other routing schemes.

VI. Mathematical Formulation

Before we describe the problem formulation, we define some of the notation used. Let $G(N, L, B)$ describe the given network, where N is the set of routers (nodes) and L the set of links (arcs) and B is the bandwidth of the links. Let n denote the number of nodes and m the number of links in the network. We assume that all bandwidths and demands for bandwidths are integral. 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. We denote a generic element of this set by (s, d). Let p denote the cardinality of the set $\mathcal{P}$. All LSP setup requests (demands) are assumed to occur between these pairs. Let M represent the node-arc incidence matrix. Each row in this matrix corresponds to a node in the graph, and each column of the matrix corresponds to an arc. Each column has exactly two nonzero entries. The column corresponding to arc (v, w) has a $+1$ in the row v and a -1 in row w, and a zero corresponding to all other rows. Assume that the arcs are numbered sequentially in any arbitrary order. Let x^{sd} be an m-vector corresponding to a pair $(s, d) \in \mathcal{P}$. Let θ_{sd} represent a scalar that is the maximum flow that can be sent between nodes s and d in the residual network. Let R be an m-vector of residual capacities. Entry j in vector corresponds to the residual capacity of arc j. The value of R is initialized to B. Let e^{sd} represent an n vector with $a+1$ in position d and -1 in position s. We assume that demands arrive one at a time. The current demand is assumed to be between routers a and b, where $(a, b) \in \mathcal{P}$. The demand is assumed to be for D (integral) units of bandwidth. Note that at this point other demands may already have been routed, and the residual capacities of the arcs have been updated to reflect these routed demands.

Assume that the maximum flow (maxflow) problem is now solved between an ingress–egress pair (s, d) using the current residual capacity of an arc as the arc capacity. This maximum flow value represents an upper bound on the total amount of bandwidth that can be routed from s to d, and this bound is tight in the case of unit demands. We define the interference between a given path between a and b, and the ingress–egress pair (s, d), as the reduction in maxflow value between that ingress–egress pair due to the routing of bandwidth on that path. Since we do not assume any knowledge of future demands, one possibility is that the demand of D units between a and b be routed to maximize the smallest maxflow value for the ingress–egress pairs in the set $\mathcal{P}$ excluding the pair (a, b), i.e., $\mathcal{P}\backslash(a, b)$. This problem MAX-MIN-MAX can be formulated as an integer programming problem, as stated below. We do not want to solve the integer program to calculate explicit routes. Later, we will present algorithms which avoid solving the integer program and yet work very well in practice.

$$\max\ z$$

$$M x^{sd} = \theta_{sd} e^{sd} \qquad \forall (s, d) \in \mathcal{P}\backslash(a, b) \tag{1}$$

$$M x^{ab} = D e^{ab} \tag{2}$$

$$x^{sd} + x^{ab} \le R \qquad \forall (s, d) \in \mathcal{P}\backslash(a, b) \tag{3}$$

$$z \le \alpha_{sd} \theta_{sd} \qquad \forall (s, d) \in \mathcal{P}\backslash(a, b) \tag{4}$$

$$x^{sd} \ge 0 \qquad \forall (s, d) \in \mathcal{P} \tag{5}$$

$$x^{ab} \in \{0, D\}^m. \tag{6}$$

Equation (1) is the maximum flow problem for each of the ingress–egress pairs in $\mathcal{P}$ except the current pair (a, b). Note that θ_{sd} represents the maximum flow value for the pair (s, d). Equation (2) states that D units of flow have to be sent between nodes a and b. Equation (3) ties up the variables for these problems. It ensures that the maximum flow problems only utilize the arc capacities that are left over after routing the current demands. The nonnegativity restrictions are specified in (5), and (6) ensures that the demand is routed along a single path.

The main drawback with the max–min formulation is that the only value that determines the objective function is the value of the smallest maxflow. Other flows do not affect the optimal solution value. For example, if there are 4 ingress–egress pairs, then the maxflow vector solutions (10, 5, 15, 20) and (15, 5, 20, 30) are treated as the same. In practice, the second solution will be preferred to the first. There are two ways to get around this problem. One way is to solve an alternate formulation to maximize the weighted sum of the maxflows. In this formulation, WSUM-MAX, the objective function of MAX-MIN-MAX, is replaced with

$$\max \sum_{(s,d)\in\mathcal{P}\setminus(a,b)} \alpha_{sd}\theta_{sd}$$

where α_{sd} is the weight for the ingress–egress pair (s, d), and constraints (4) are dropped from the problem. The weights for each ingress–egress pair are chosen to reflect the "importance" of the ingress–egress pair to the service provider (as, for instance, the relative revenue potential of traffic carried between each ingress–egress pair). Next we state a couple of definitions that are needed in order to develop the second formulation.

Definition 1: Given two vectors a and b, vector a is defined to be lexicographically not less than b if the first nonzero component in $a - b$ is nonnegative. We will denote that a is lexicographically not less than b by $a \succeq b$.

Definition 2: Given an n-vector a, an nondecreasing ordering of a, denoted by $I(a)$, is a renumbering of the components of a such that $a_1 \leq a_2 \leq \cdots \leq a_n$.

The second way to get around the problem with MAX-MIN-MAX is to define the problem LEX-MAX. In this problem, the objective is to find a route such that the smallest maxflow value is as high as possible. Among all the solutions with the same smallest maxflow value, the secondary objective is to find the solution such that the second lowest maxflow is as high as possible, and so on. More formally, let $\mathcal{F}$ be the set of feasible $(p-1)$-vector of maxflow values between the $p-1$ ingress–egress pairs (all pairs except the current one) after the current demand is routed. The objective of LEX-MAX is to find the $\tilde{\theta} \in \mathcal{F}$ such that $I(\tilde{\theta}) \succeq I(\theta)$ for all $\theta \in \mathcal{F}$. It is possible to define an integer programming problem (which is much more complicated than MAX-MIN-MAX) to solve LEX-MAX. Since we do not solve this problem exactly, we will not state this integer programming problem here. Note that solving LEX-MAX is at least as difficult as solving MAX-MIN-MAX. We now show that all the problems defined above are NP-hard. First we show that WSUM-MAX is NP-hard. In order to prove this, we first show the following result.

Lemma 1: In both these formulations given above, there exists an optimal integral $\hat{x}^{sd}\ \forall (s, d) \in \mathcal{P}$ if all the capacities are integral.

Proof: Consider an optimal solution, $\tilde{x}$ to the max–min problem. Note that $\tilde{x}^{ab}$ is integral. Set $x^{ab} = \tilde{x}^{ab}$ in WSUM-MAX. The problem then decomposes into $p-1$ independent maxflow problems. The capacity of the links in each of these maxflow problems is the integral vector $R - \tilde{x}^{ab}$. The constraint matrix for the maxflow problem is totally unimodular. Therefore, there exists an optimal solution for the maxflow problem that is integral if all the capacities are integral. □

We can now use this result to prove the following.

Theorem 2: The problem WSUM-MAX is NP-hard.

Proof: (Outline) Consider the case where there are just two disjoint ingress–egress pairs (s_1, d_1) and (s_2, d_2). Consider the problem of routing one unit of flow (without splitting) from s_1 to d_1, and the question is whether this can be done so that the maxflow between s_2 and d_2 is greater than some value k. Since both the flows are integral, one can use the same techniques used by Even, Itai, and Shamir [6] to show that the directed two commodity integral flow problem is NP-hard. The transformation is from 3SAT. □

Theorem 3: Problems MAX-MIN-MAX and LEX-MAX are NP-hard.

Proof: Since even the two ingress–egress pair problem is NP-hard, this implies that MAX-MIN-MAX is NP-hard. Since solving LEX-MAX is at least as difficult as solving MAX-MIN-MAX, this problem is also NP-hard. □

VII. Solution Approach

In this section we outline a solution approach for solving WSUM-MAX and LEX-MAX. We first outline the approach for WSUM-MAX, and later in this section we show how to modify this approach to solve LEX-MAX approximately.

A. Solving WSUM-MAX

In light of the fact that WSUM-MAX is NP-hard, one option is to solve the linear programming relaxation of this problem. If the linear program results in not splitting the demand of D units between a and b, then the solution is optimal to WSUM-MAX. If it splits the flow, good heuristics have to be found to compute a single path. The main drawback of this approach is the fact that it is not possible to incorporate any path constraints. For example, if there is a restriction on the number of hops that the demand can take, then it is not possible to incorporate it into the linear programming formulation. Further, if the demand is split along many different paths, then we need a good rounding approach to generate a good feasible solution to the problem where the demand is not split. Hence we do not use this approach. Also, because of the splitting, the linear programming formulation is not a good benchmark for evaluating the efficacy of our routing heuristic. An integer programming formulation avoids the splitting problem but is feasible only for small problems.

The approach that we take is to determine appropriate weights for the links in the network and route the demand along the weighted shortest path. Additional path constraints can be incorporated when the shortest path problem is being solved. The problem now is to determine appropriate link weights so that the current flow does not interfere too much with potential future

demands. The link weights are estimated by the following procedure. First we relax constraint (2), i.e., we remove the requirement that D units of flow have to be routed between nodes a and b. Equivalently, this can be viewed as setting the value of D to zero. This results in the decoupling of WSUM-MAX into $p-1$ independent maxflow problems, one for each ingress–egress pair in $\mathcal{P}\backslash(a, b)$. These problems are now solved and let $\hat{\theta}_{sd}$ represent the maxflow value for ingress–egress pair (s, d). The optimal solution to WSUM-MAX when $D = 0$ is given by $\alpha_{sd}\hat{\theta}_{sd}$. Now if some demand between a and b is routed on a link, the residual capacity of the link decreases. This may result in a decrease in the current optimal solution value of WSUM-MAX. The weight of a link is now estimated to be the rate of change in the optimal solution of WSUM-MAX with respect to changing the residual capacity of the link. Let $(\partial\hat{\theta}_{sd}/\partial R(l))$ represent the change in maxflow value between ingress–egress pair (s, d) if the residual capacity of link l is changed incrementally. Therefore , the partial derivative represents the reduction in the (s, d) maxflow value when an incremental amount of the current demand is routed on link l. The weight $w(l)$ of a link l is set to

$$w(l) = \sum_{(s,d)\in\mathcal{P}\backslash(a,b)} \alpha_{sd}\frac{\partial\hat{\theta}_{sd}}{\partial R(l)}. \tag{7}$$

The weight of a link represents the change in the objective function value of WSUM-MAX if an incremental amount of the current demand is routed on that link. Note that the weight of the link $w(l)$ is a heuristic since it ignores the dependencies between the different links, and also the fact that D units of flow have to be routed from a to b. By linear programming duality, associated with each maximum flow is a minimum cut. The maximum flow value of a particular ingress–egress pair decreases whenever the capacity of any of the arcs in any mincut for that pair is decreased infinitesimally. In fact, the maximum flow value will not decrease if the capacity of any other link (not in a mincut) is decreased infinitesimally.

Definition 3: An arc is defined as critical for a given ingress–egress pair, if that arc belongs in any mincut for that ingress–egress pair. The mincut is computed with the current residual capacities on the links.

Let C_{sd} represent the set of critical links for the ingress–egress pair (s, d). From the maxflow-mincut, theorem

$$\frac{\partial\hat{\theta}_{sd}}{\partial R(l)} = \begin{cases} 1 & \text{if } l \in C_{sd} \\ 0 & \text{otherwise} \end{cases}$$

therefore,

$$w(l) = \sum_{(s,d):\, l\in C_{sd}} \alpha_{sd}.$$

Therefore, the problem of computing the weights of the arcs is now reduced to determining the set of critical arcs for all ingress–egress pairs. The maximum flow 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] or in time $O(nm+n^2\log U)$ using an excess scaling algorithm. There may be several alternate mincuts for a given ingress–egress pair. The critical links for the ingress–egress pairs are arcs belonging to the union of all these mincuts. We show that the set of critical links for a given ingress–egress pair can be determined by one execution of the maxflow algorithm between that ingress–egress pair. We use a flow residual graph [1] in determining critical links.

Theorem 4: Let $G = (V, E, c)$ be the given directed graph. Let s represent the source node and d the destination node. Assume that a maximum flow between s and d has been computed. Let S be the set of nodes reachable from s in the flow residual graph. Let T represent the nodes that can reach the sink d in the flow residual graph. An arc $(i, j) \in C_{sd}$ if

- arc (i, j) is filled to capacity,
- $j \notin S$ and $i \notin T$,
- there is no path between i and j in the flow residual graph.

Proof: (Outline) Let I represent the set of nodes reachable from i in the residual graph. let $S' = S \cup I$. Note that from the assumptions made both j and t are not in S'. Therefore, arc (i, j) belongs in the minimum cut $(S', V\backslash S')$. □

The procedure for determining all the arcs in C_{sd} takes $O(m^2)$ time in addition to the maxflow computation. In practice, the running time for determining C_{sd} will be dominated by the maxflow computation.

- The value of α_{sd} can be chosen to reflect the importance of the ingress–egress pair (s, d).
- If the value of $\alpha_{sd} = 1$ for all (s, d) then $w(l)$ represents the number of ingress–egress pairs for which link l is critical.
- The weights can be made inversely proportional to the maxflow values, i.e., $\alpha_{sd} = 1/\hat{\theta}_{sd}$ where $\hat{\theta}_{sd}$ is the maximum flow value for the ingress–egress pair (s, d). This weighting implies that the critical arcs for the ingress–egress pairs with lower maximum flow values will be weighted heavier than the ones for which the maxflow value is higher.
- If the network also carries best effort traffic and if the delays are proportional to the flow on the links, then the residual capacity of the link can be used to influence the weight of the link. For example, the weight of link l can be set to $w'(l)$, where $w'(l) = w(l)/R(l)$ and $w(l)$ is defined above and $R(l)$ is the residual capacity of link l.

Once the weights of the links are determined, the idea is to route the traffic along the shortest weighted path from a to b. Before this is done, all links having a residual capacity of less than D units are eliminated. When computing the shortest weighted path, the weight of link l is set to $w(l)$. In order to ensure that among arcs with zero weights, i.e., arcs that are not critical to any ingress–egress pair, we choose the one where the number of hops is minimal, we set the weight of these arcs to some small positive number. A high level view of the minimum interference routing algorithm (MIRA) is described in the figure.

```
Minimum Interference Routing Algorithm
(MIRA)
INPUT:
A graph G(N, L) and a set B of residual
  capacities on all the arcs. An ingress
```

node a and an egress node b between which a flow of D units have to routed.

OUTPUT:

A route between a and b having a capacity of D units.

ALGORITHM:

1. Compute the maxflow values $\forall (s,d) \in \mathcal{P}!\backslash(a,b)$.
2. Compute the critical link sets C_{sd} $\forall (s,d) \in \mathcal{P}\backslash(a,b)$.
3. Compute the weights $w(l) = \sum_{(s,d):\, l \in C_{sd}} \alpha_{sd}\ \forall l \in L$.
4. Eliminate all links which have residual bandwidth less than D and form a reduced network.
5. Using Dijkstra's algorithm compute the shortest path in the reduced network with $w(l)$ as the weight of link l.
6. Route the demand of D units from a to b along this shortest path and update the residual capacities.

Remarks:

1) For routing each demand, $p-1$ maxflow problems have to be solved. The maximum flow values are computed using an implementation of the Goldberg Tarjan highest level pushing algorithm. We have observed that this algorithm is extremely fast [1] and it is possible to solve thousands of maxflow values on networks with a few hundred nodes in the order of a few seconds.
2) If the links are undirected, then the number of maxflow computations will be $\min(p-1, n)$ using the Gomory–Hu algorithm [9]. Some modifications are needed in order to maintain all the arcs in the mincut, but this can be done efficiently.
3) If the links are directed, it is unlikely that the number of maximum flow computations can be reduced to less than $p-1$. Frank and Frisch [7] give examples where the number of different maximum flow values is $\Omega(n^2)$. However, if an approximation is made, where the critical links for a particular ingress–egress pair, say (s, d), are defined as the critical links for (s, d) if $\hat{\theta}_{sd} < \hat{\theta}_{ds}$, and the critical links for (d, s) otherwise. In other words, the critical links for a given (s, d) pair are defined as the links in the minimum cut in the direction in which the maxflow value is smaller. In this case, one can show that a suitable symmetric cut function can be defined and using the method of Cheng and Hu [5] the critical links can be determined with $\min(n, p-1)$ maxflow computations.
4) This algorithm can be easily adapted if the routers are bottlenecks, in addition to the links being bottlenecks. This is done by splitting the nodes in the network into two nodes and introducing an arc between these two nodes. The node capacity is now represented as an equivalent arc capacity on this newly introduced arc.
5) It is easy to incorporate hop-count constraints, since the final routing is done via a shortest path algorithm. We can use Bellman–Ford instead of using Dijkstra's algorithm in order to incorporate hop count constraints. Other constraints such as delay constraints make the problem a constrained shortest path problem which is NP-hard. However, one can use a pseudopolynomial time algorithm or some heuristic approach to solve this problem.

B. Solving LEX-MAX

As outlined in the previous section, LEX-MAX is NP-hard. One can solve LEX-MAX as a sequence of MAX-MIN-MAX problems with additional constraints. This approach will be computationally intensive. Instead, we approximate the LEX-MAX problem as a special case of the WSUM-MAX problem (with a special set of weights) and use the minimum interference routing algorithm to solve LEX-MAX. Recall that given $\mathcal{F}$, the set of feasible $(p-1)$-vector of maxflow values between the $p-1$ other ingress–egress pairs after the current demand is routed, the objective of LEX-MAX is to find the $\tilde{\theta} \in \mathcal{F}$ such that $I(\tilde{\theta}) \succeq I(\theta)$ for all $\theta \in \mathcal{F}$. Note that the $I(\tilde{\theta})$ represents a nondecreasing arrangement of components of $\tilde{\theta}$. The approximation that we make is to assume that the ordering of the maxflow values, after the current demand is routed, will be the same as the current ordering of the maxflow values. This is usually accurate considering that demand to be routed is small. However, this is clearly not always true. Intuitively, making this approximation means that we want to protect the ingress–egress pair with the smallest current maxflow value first, followed by the second smallest, and so on. This is done by solving WSUM-MAX where the weight of the ingress–egress pair with the smallest maxflow value is the highest followed by the second smallest maxflow value, and so on. Assume that the ingress–egress pairs are numbered such that

$$\hat{\theta}_{s_1 d_1} \le \hat{\theta}_{s_2 d_2} \le \cdots \le \hat{\theta}_{s_{p-1} d_{p-1}}$$

where $\hat{\theta}_{sd}$ denotes the maxflow value for the ingress–egress pair (s, d). Now we determine the constants $\beta_i = \alpha_{s_i d_i}$ so that WSUM-MAX solves the approximate version of LEX-MAX. We want to choose weights for WSUM-MAX such that a difference of one unit in the ith smallest maxflow should outweigh the effect of the maximum difference in all of the $(i+1)$th, $(i+2)$th, $\cdots (p-1)$th smallest maxflows. Then we have

$$\beta_i > (\beta_{i+1} + \beta_{i+2} + \cdots + \beta_{p-1})\mu \qquad i = (p-2), \cdots, 1 \tag{8}$$

where μ is the largest change in maxflow value for any ingress–egress pair. Since μ represents the change in maxflow value after routing a demand of D units,

$$\mu \le m \times D. \tag{9}$$

The inequality comes from the fact that on routing a demand of value D, the value of any cut can be reduced by at most D times the size of the cut, which is bounded by mD. From (8) and (9) we get

$$\beta_i = \begin{cases} 1 & i = (p-1) \\ mD(1+mD)^{p-i-2} & i = (p-2), \cdots, 1. \end{cases} \tag{10}$$

We can now use MIRA to solve the approximate version of LEX-MAX.

C. Δ-Critical Links

Recall that a link is critical for a particular ingress–egress pair if routing one unit of flow (of the current demand) on that link reduces the maxflow value for the ingress–egress pair. This is the same as saying that the maxflow value for the ingress–egress pair decreases when the capacity of the critical link is decreased by one unit. This also implies that if the capacity of a noncritical link for an ingress–egress pair is decreased by one unit, then the maxflow value for that ingress–egress pair does not decrease. However, if the capacity of a noncritical link for a given ingress–egress pair is decreased by more than one unit, the maxflow value for that ingress–egress pair may decrease. This brings us to a generalization of the notion of a critical link, a Δ-critical link.

Definition 4: A Δ-critical link for an ingress–egress pair is a link such that if the capacity of the link is decreased by Δ, the maxflow value for the ingress–egress pair decreases.

Note that the critical links as defined in Definition 3 are 1-critical links, because if the capacity of a link is decreased by one unit, then the maxflow decreases by one unit. In the definition of Δ critical links, we just insist that the maxflow value decreases (not necessarily by Δ units). The notion of a link being Δ critical captures the notion that there may be links that are close to being critical that we may want to identify. This is especially important if the computation of critical links is done periodically, i.e., once every k demands. In this case, it is important to "protect" links that are currently not critical but are close to being critical because they may become critical before the computation is performed again. Next comes the question of finding the set of Δ-critical links for a given value of Δ. One naive way to find whether a link is Δ-critical for an ingress–egress pair is to reduce the capacity of the link by an amount Δ, and compute the maxflow all over again, and check if the maxflow value decreases. However, this approach is prohibitively expensive, since it requires m times the time for a single maxflow computation to find links that are critical. A faster approach of finding Δ-critical links is an open algorithmic problem. Instead of determining the set of Δ-critical links exactly, we determine the approximate set of Δ-critical links. The heuristic that we use is described in APPROX Δ-CRITICAL LINKS.

Approx Δ-Critical Links
INPUT: A graph $G(N, L)$ and a set B of residual link capacities
A threshold Δ and a ingress–egress pair $(s, d) \in \mathcal{P}$
OUTPUT: $C_{sd}(\Delta)$, the set of Δ-critical links for ingress–egress pair (s, d)
PROCEDURE:
1. $C_{sd}(\Delta) \leftarrow \phi$
2. Compute maxflow f for ingress–egress pair (s, d)
   Let $G_f(N, L)$ be the flow residual graph after the maxflow computation, and $R_f(l)$ be the residual capacity of any link $l \in L$ in $G_f(N, L)$.
3. For each link $l = (i, j) \in L$, include l in $C_{sd}(\Delta)$ if and only if both of the following two conditions hold:
   - $R_f(l) < \Delta$
   - There is no path between i and j in $G_f(N, L)$ with capacity greater than or equal to $\Delta - R_f(l)$

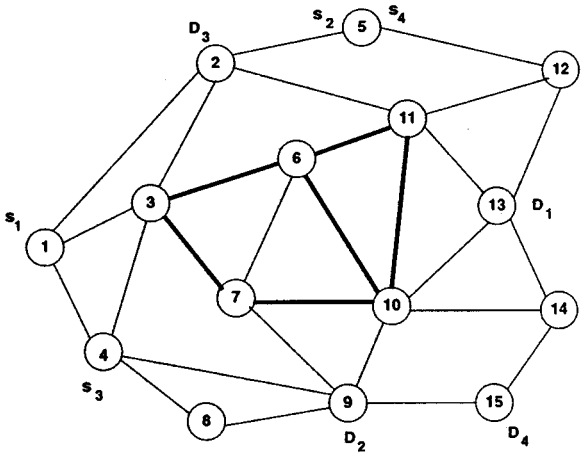

Fig. 2. Illustrative example (network 1).

Note that if the flow residual capacity of a link is not less than Δ, or if there is a path of capacity $\Delta - R(l)$ from node i to node j in the flow-residual graph, then the link cannot be Δ-critical. However, this is only a necessary condition. Using the procedure APPROX Δ-CRITICAL LINKS, we will determine some links to be Δ-critical even if they are not. The time-complexity of APPROX Δ-CRITICAL LINKS is $O(m^2)$ (the same as that required to determine the exact set of 1-critical links using the procedure outlined in Theorem 4), in addition to the time complexity for maxflow computation.

VIII. Performance Studies

In this section, we will compare the performance of our routing algorithm (MIRA) to min-hop (MHA) and widest shortest path (WSP) [11] routing algorithms, on some sample networks. Comparisons will be made using both versions of MIRA: the one involving the lexicographic criteria (called L-MIRA), and the other involving the weighted sum criteria (called S-MIRA). In all the experiments involving S-MIRA, we will assume the weights of all the ingress–egress pairs to be the same. The performance of each algorithm is measured by the proportion of LSPs rejected by the algorithm. We will also study the effect of the parameter Δ and the frequency of maxflow computation on the performance of MIRA.

A. Illustrative Example

Most of the performance comparisons shown in this paper are done using the network shown in Fig. 2. The ingress–egress pairs are shown in the figure. For this illustrative example, the

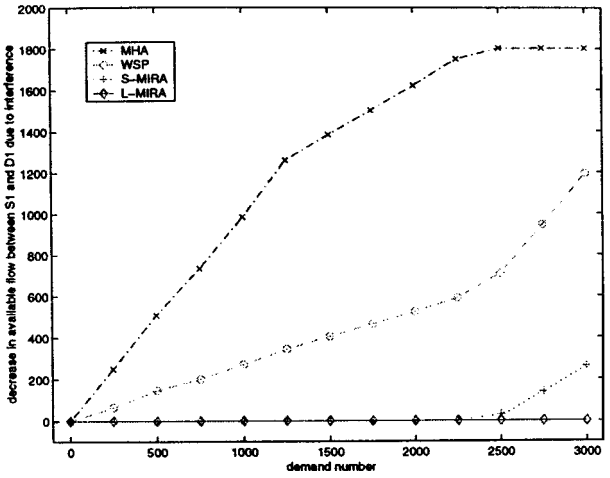

Fig. 3. Decrease in available flow between S1 and D1 due to interference.

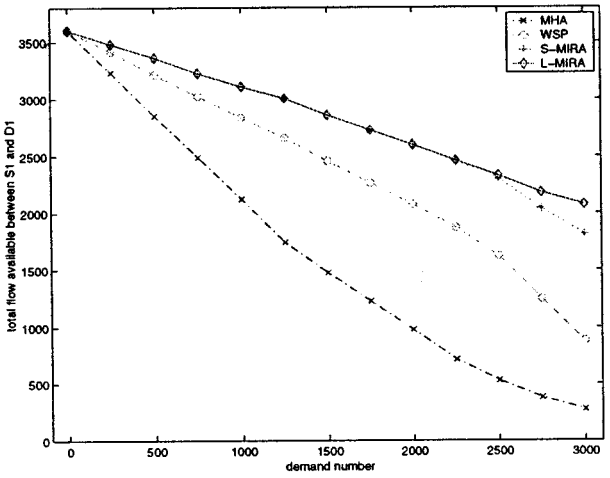

Fig. 4. Total available flow between S1 and D1 after every LSP is routed.

capacity of the light links is 12 units and that of the dark links is 48 units (taken to model the capacity ratio of OC-12 and OC-48 links) and each link is bidirectional (i.e., acts like two unidirectional links of that capacity). In all the simulation experiments described in this paper, LSPs arrive randomly, and at the same average rate for all ingress–egress pairs.

B. Interference

We first show experimentally that the interference, as defined in Section VI, is indeed reduced by MIRA in comparison to min-hop and WSP. We then show that this reduced interference translates to better LSP acceptance. In the experiments of this subsection, we will use the network of Fig. 2, but we will scale all the capacities by 100. This larger capacity network is used for the performance studies because this permits us to experiment with thousands of LSP setups. The LSP bandwidth requests are taken to be uniformly distributed between 1 and 3 units. All the experiments of this subsection are done for the "static" case, i.e., arriving LSPs stay in the network forever. In this case, we will look at the flows between various ingress–egress pairs after a certain number of LSPs have arrived.

Fig. 3 plots the decrease in max-flow between ingress–egress pair (S1, D1) as more and more LSPs are routed between other ingress–egress pairs. We see that with S-MIRA, there is no interference at all till more than 2000 LSPs have been setup, whereas with L-MIRA, there is no interference till 3000 LSPs are routed. However, every LSP setup using the other two algorithms causes interference with potential future LSPs between S1 and D1.

Fig. 4 shows the total available flow between S1 and D1 after every LSP is routed. Note that for L-MIRA, the decrease in the available flow between S1 and D1 is entirely due to the LSPs routed between S1 and D1 since, as seen from Fig. 3, there is no interference at all when the number of LSPs is less than 3000 (the same is true for S-MIRA when number of LSPs is less than 2000). The available capacities are much lower for min-hop and WSP. This means that min-hop and WSP should experience more blocking between S1 and D1. (S1 and D1 were picked for illustrative purposes only. The same applies for other ingress–egress pairs.)

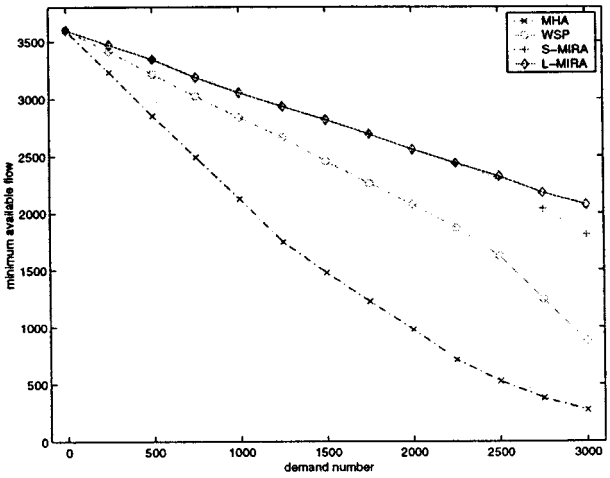

Fig. 5. Minimum available flow between all ingress–egress pairs after every LSP is routed.

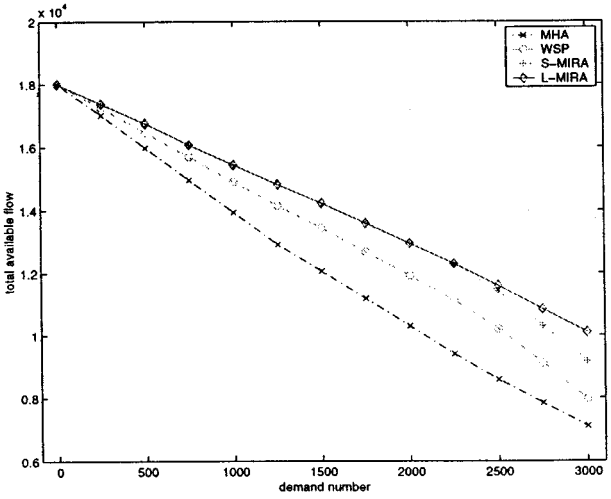

Fig. 6. Total available flow between all ingress–egress pairs after every LSP is routed.

Figs. 5 and 6 show the minimum and the sum of the available flows, respectively, between every ingress–egress pair after each LSP is routed. Again, we see that the available flows for MIRA

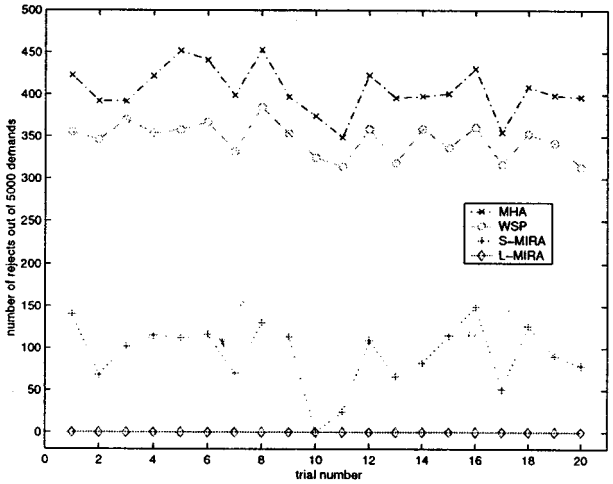

Fig. 7. Static case: number of LSP setup requests rejected for 20 experiments.

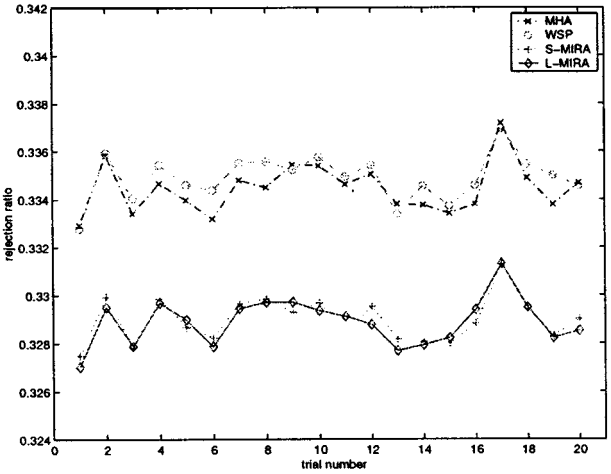

Fig. 9. Dynamic case: rejection ratio for 20 experiments—network 2.

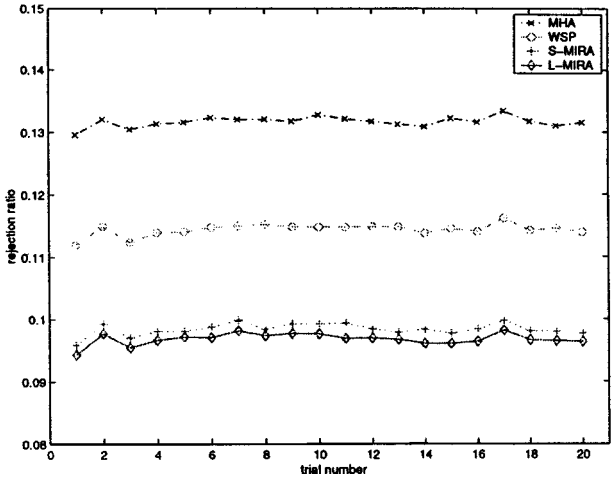

Fig. 8. Dynamic case: rejection ratio for 20 experiments—network 1.

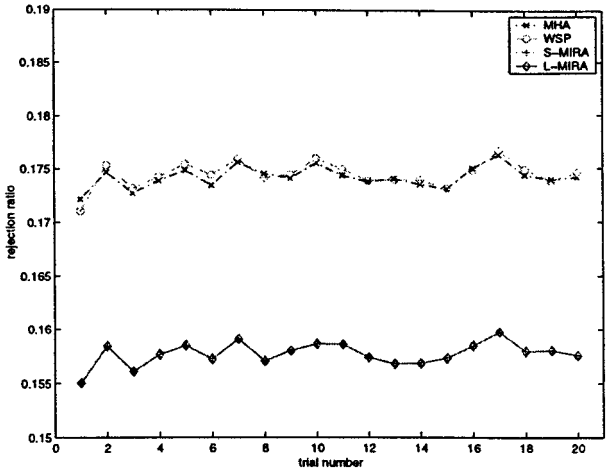

Fig. 10. Dynamic case: rejection ratio for 20 experiments—network 3.

are consistently higher than those for min-hop and WSP. Moreover, we see that the available flows for L-MIRA are higher than S-MIRA. Next we see how these increased available flows translate to better performance.

C. LSP Acceptance

In the first LSP acceptance experiment, we assume that all LSPs are long lived ("static" case). We load the network with 5000 LSPs and observe the number of LSPs rejected by the different algorithms. We conducted 20 trials. The results are shown in Fig. 7. Observe the noticeable degradation in performance of min-hop and WSP due to interference. Also note that L-MIRA performs much better than S-MIRA; till 5000 LSP arrivals, the number of rejects in L-MIRA is zero for all the experiments. Note that in S-MIRA, in an attempt to maximize the total available maxflow over all ingress–egress pairs, the available capacity between some ingress–egress pair may become very small, causing a large number of rejects. L-MIRA, however, avoids this problem by giving the maximum priority to the ingress–egress pair with the least maxflow, and hence performs better.

In the second experiment, we tried to determine the dynamic behavior of the three algorithms. Fig. 8 shows the proportion of LSPs rejected for 20 experiments, under the following scenario. LSPs 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 $1/\mu$. For our experiments, $(\lambda/\mu) = 150$. In this case, too, bandwidth demands for LSPs are uniformly distributed between 1 and 3 units. For our experiment in this case, we take the same network as in Fig. 2 (network 1), but the capacities are scaled only by 10 (and not by 100 as for the static case—this is done so that the system "warmup" time is not too large). The rejection ratio is calculated over a window of approximately 1 000 000 LSP setup requests. Again, we see that both L-MIRA and S-MIRA perform much better than min-hop and WSP. Between L-MIRA and S-MIRA, L-MIRA performs better, as we would expect.

Figs. 9 and 10 show the proportion of LSPs rejected for 20 experiments, for the dynamic case, under the same conditions as above, but for two other networks, network 2 (18 nodes, 30 links), and network 3 (20 nodes, 35 links). In all the cases, we see that S-MIRA and L-MIRA perform much better than MHA and WSP. Note that in Fig. 10, L-MIRA and S-MIRA perform identically.

Fig. 11 shows results for the same experimental setup as that for Fig. 8, except that the load is changed so that the rejection ratio for

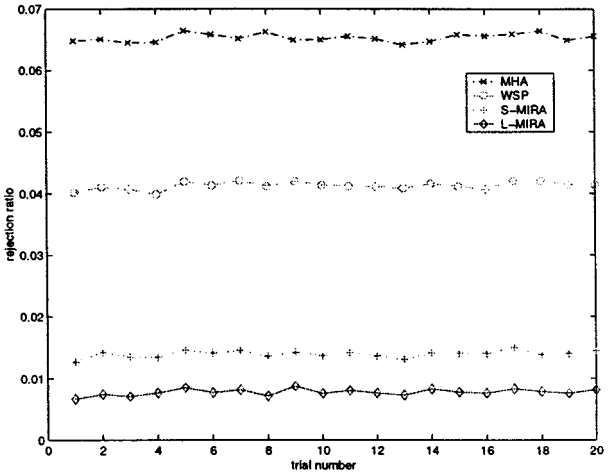

Fig. 11. Dynamic case: rejection ratio for 20 experiments, operating point with lower rejection ratio—network 1.

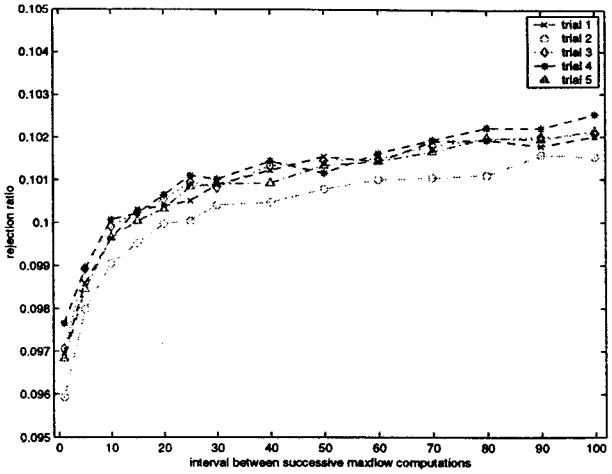

Fig. 12. Rejection ratio versus interval (in terms of LSP request arrivals) between successive critical link computations.

TABLE I
PERCENTAGE IMPROVEMENT IN REJECTION RATIOS (COMPARED TO MHA) FOR VARIOUS ALGORTIHMS FOR DIFFERENT NUMBERS OF INGRESS–EGRESS PAIRS

No. of pairs	WSP	S-MIRA	L-MIRA
4	13.14%	25.36%	26.50%
8	-0.33%	8.84%	9.42%
16	0.00%	3.81%	3.90%
64	7.39%	15.89%	13.25%

the minimum interference based algorithms is around 1%. Comparing to Fig. 8, we see that the performance improvement of both L-MIRA and S-MIRA, in comparison to min-hop and WSP, is even higher at this operating point of lower rejection ratios.

Also, for the same experimental setup as that for Fig. 8, we varied the number of ingress–egress pairs and compared the percentage improvement in rejection ratios with respect to the rejection ratio for min-hop. The results are shown in Table I. We see that even with a large number of ingress–egress pairs (64 out of a possible 210 ingress–egress pairs), there is significant performance improvement over min-hop (and also over WSP) with S-MIRA and L-MIRA.

D. Effect of Critical-Link Computation Frequency on Performance

In the experiments described so far, the computation of the maxflow, and hence the computation of the critical links, is done on every LSP arrival. However, it is worthwhile examining whether it is necessary to recompute the critical links after each LSP is routed. Note that without critical link computation, each LSP routing involves only a shortest-path computation. Ideally, one would like to avoid frequent computation of critical links. We carried out a number of experiments to determine the effect of the critical link computation frequency on the system performance. Five representative trials are shown in Fig. 12, when the routing algorithm is L-MIRA. The network is the same as the one shown in Fig. 2 (but with the capacities scaled by 10), and the simulation conditions are the same as those for the dynamic case described in the last subsection. In the figure, the y axis shows the rejection ratio, and the x axis shows the interval between two successive critical link computations, in terms of LSP request arrivals. Note that the computed critical link set, and hence the path weights, do not change between successive critical link computations. The plots show that, in general, as we increase the interval between successive critical link computations, the rejection ratio shows an increasing trend, as we would expect. However, the rejection ratio, even when the critical link computation frequency has been reduced by a factor of 50 or more, does not differ vastly from the case when the critical link computations are done on every LSP request arrival. Note that the gradient of the plots are steeper when the interval between successive critical link computations is small, and become flattened out as the interval increases, as we would intuitively expect.

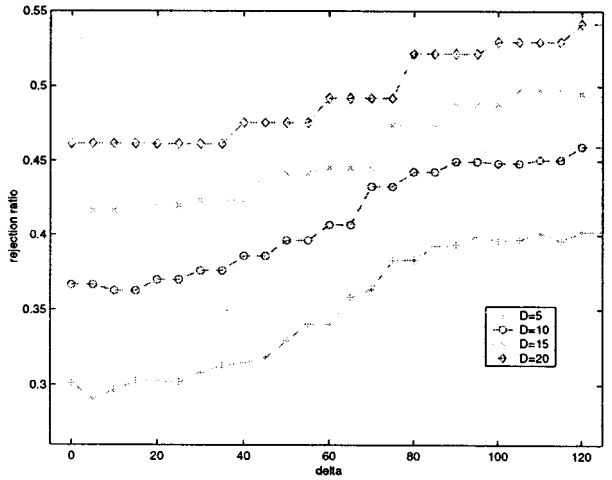

Fig. 13. Rejection ratio versus Δ, for various demand sizes D.

E. Effect of Δ on Performance

Now we will study the effect of the parameter Δ (which is used for determining the critical link set) on the performance of the algorithm. Fig. 13 shows some representative examples to illustrate this. The network is the same as the one in Fig. 2 (with the capacities scaled by 10), and the simulation conditions are the same as those for the dynamic case for the last two subsections. However, in this case, the size of each demand (D) is kept fixed. In the figure, the results are shown for four different

demand sizes (D). Moreover, for demand size D, the average load $(\lambda/\mu) \times D$ is kept fixed at 300 units. The figure shows, as we would intuitively expect, that the rejection ratio, in general, increases as we increase Δ. A careful inspection shows that for three out of the four cases, namely $D = 5, 10, 20$, as we start increasing Δ from 1, the number of rejects decreases very slightly, after which it increases monotonically (for $D = 15$, the number of rejects always increase with increasing Δ). For $D = 5, 10, 20$, the minimum is attained when $\Delta = D$. For $\Delta \geq 120$, the number of rejects remain the same even as we increase Δ. We have also verified that for $\Delta \geq 120$, the number of rejects is the same as that for min-hop. Note that our algorithm reduces to min-hop when Δ is greater than the maximum link capacity (then all the links would be considered critical links and would be given the same weight). Note that in our network, all but five links have a capacity of 120 units, and the remaining five have 480 units. Thus, we would expect that our algorithm would perform similar to min-hop when $\Delta \geq 480$. In this case, however, the performance is the same for $\Delta \geq 120$. Fig. 13 also demonstrates that when the size of the demand is increased while keeping the average load constant, the number of rejects increase. This can be intuitively explained in the following way. Consider two demand sizes, D and $2D$. First, an arrival of a demand of size $2D$ is equivalent to a burst of two arrivals of size D. Second, since a flow cannot be split, we may have to reject a demand of size $2D$ because of the nonavailability of a path with $2D$ units of bandwidth, even though we may be able to route one (or more) demands of size D.

For most of the experiments we have carried out, we have observed that the minimum rejection ratio is achieved either at $\Delta = D$, or $\Delta = 1$, when the demand size is kept fixed at D. However, in a practical scenario, we would expect that the demand size would not be fixed, and would vary over a range. What value of Δ achieves the minimum rejection ratio in that case remains to be seen.

Note that when the demand sizes are small compared to the link capacities (which is typically the case), setting $\Delta = 1$ may not be a bad choice, if the critical links are computed on every request arrival. This is also evident from Fig. 13. However, if the maxflow computation, and hence the determination of the critical links, is done less frequently, then setting $\Delta = 1$ may be a bad choice, even if the demand sizes are small. In this case, a lot of flow might be pushed into the network between two successive maxflow computations, and hence links that were not the most critical during a particular maxflow computation might become the most critical links before the next maxflow computation occurs. Thus, we would like to set Δ to some larger value, such that it identifies the near-critical links, i.e., the links that can potentially become bottlenecks before the next maxflow computation. Fig. 14 illustrates this on a network 2 (18 nodes, 30 links) with 3 ingress–egress pairs. The traffic is Poisson with $(\lambda/\mu) = 150$, and D is uniformly distributed between 1 and 3 units. The plots are shown for the cases when the interval between successive maxflow computations is 30, 40 (in terms of LSP request arrivals). The plots demonstrate that when the interval between successive maxflow computations is large, rejection ratio initially decreases with increase in Δ, after which it starts increasing again. When Δ is too small, then we are missing out too many near-critical links; while when Δ is too large, then too many links are being included in the set of critical links, reducing the algorithm to min-hop. The value of Δ which achieves the minimum rejection ratio would, in general, depend on the demand size distribution and the interval (in terms of LSP request arrivals) between successive maxflow computations.

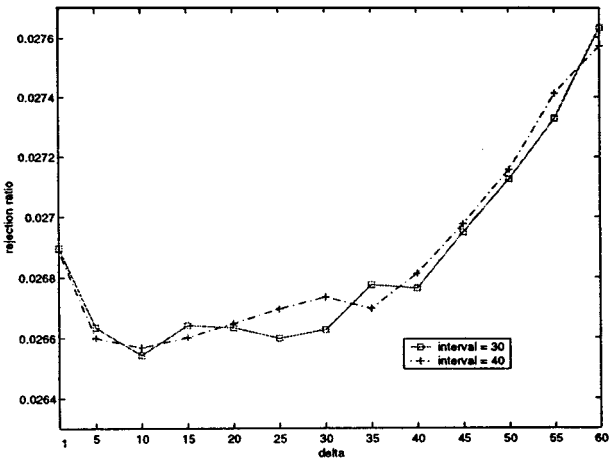

Fig. 14. Rejection ratio versus Δ, for different critical link compuatation intervals.

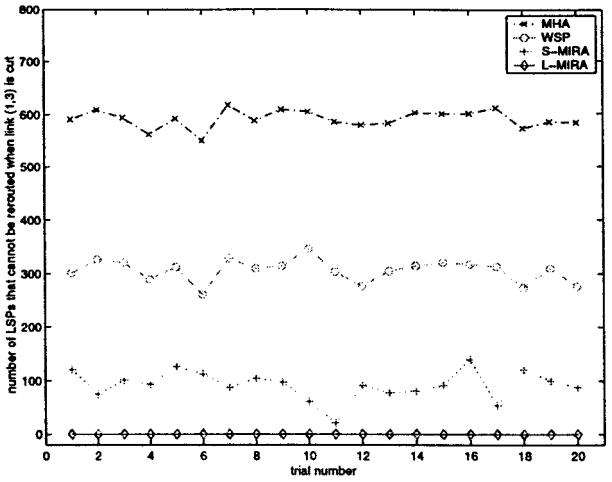

Fig. 15. Rerouting performance for 20 trials.

F. Link Failure Rerouting Performance

Next we compare the performance of the algorithms in terms of how good the algorithms are in rerouting already setup LSPs when a link failure happens. Note that to successfully reroute an LSP, there must be capacity available between the corresponding ingress–egress pair after the link failure has happened. Since MIRA tries to leave capacity open between ingress–egress pairs, it should have better rerouting performance. To test this, we first route some randomly generated (static) LSPs in the network shown in Fig. 2 (capacities scaled by 100), under the four different algorithms, S-MIRA, L-MIRA, WSP, and min-hop. We then cut a randomly chosen link and then use the same algorithm for rerouting the LSPs that had been routed on the link that is cut. The results, for 20 different trials, are shown in Fig. 15. For the results shown, the cut link is (1, 3) and the number of

LSPs routed initially (before the link is cut) is 3800. It can be seen from the figure that, as expected, S-MIRA and L-MIRA have much better rerouting performance than min-hop and WSP. Moreover, in the results shown, L-MIRA performs better than S-MIRA.

IX. Concluding Remarks

The primary contribution of the paper is the development of routing algorithms based on the notion of minimum interference. It is based on the observation that usage of certain critical links must be avoided to the extent possible. The objective in min-hop routing is to minimize resource usage. It does not account for traffic asymmetries which result from certain ingress–egress pairs having more offered traffic. Also, it does not permit the protection of certain ingress–egress pairs from having their available capacities being reduced too much by traffic traversing between other ingress–egress pairs. Minimum interference routing takes these factors into account when determining the path for the current demand. The primary application of minimum interference routing is in explicit routing of LSPs in MPLS networks. Another application is for routing wavelength paths in dynamically provisionable optical networks provided wavelength conversion is permitted at each node. We showed by simulations on three networks that minimum interference routing has very good LSP acceptance and rerouting performance in comparison to min-hop and WSP, which do not take the ingress–egress information into account. The difference in performance spans a wide variety of operating conditions. We also showed that most LSP routes can be computed using only a shortest-path computation, and that frequent determination of critical links is not necessary to ensure good performance. An immediate extension to this work is the incorporation of priorities. Another topic for future study is aggregation for interdomain routing. The max-flow values between ingress–egress pairs calculated by the algorithm can be used as a measure of available bandwidth for aggregation.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[2] D. O. Awduche, L. Berger, D. Gan, T. Li, G. Swallow, and V. Srinivasan, "Extensions to RSVP for LSP tunnels," Internet Draft draft-ietf-mpls-rsvp-lsp-tunnel-04.txt, Sept. 1999.

[3] D. O. Awduche, J. Malcom, J. Agogbua, M. O'Dell, and J. McManus, "Requirements for traffic engineering over MPLS," RFC 2702, Sept. 1999.

[4] R. Callon, N. Feldman, A. Fredette, G. Swallow, and A. Viswanathan, "A framework for multiprotocol label switching," Internet Draft draft-ietf-mpls-framework-03.txt, June 1999.

[5] "Ancestor tree for arbitrary multi-terminal cut functions," in *Proc. Conf. Integer Programming Combinatorial Optimization*, 1990.

[6] S. Even, A. Itai, and A. Shamir, "On the complexity of timetable and multicommodity flow problems," *SIAM J. Computing*, vol. 5, pp. 691–703, 1976.

[7] H. Frank and I. T. Frisch, *Communication, Transmission, and Transportation Networks*. Reading, MA: Addison-Wesley, 1971.

[8] A. V. Goldberg and R. E. Tarjan, "Solving minimum cost flow problem by successive approximation," in *Proc. 19th ACM Symp. Theory Computing*, 1987, pp. 7–18.

[9] R. E. Gomory and T. C. Hu, "Multi-terminal network flows," *J. SIAM*, vol. 9, pp. 551–570, 1961.

[10] R. Guerin, D. Williams, A. Przygienda, S. Kamat, and A. Orda, "QoS routing mechanisms and OSPF extensions," Internet Draft draft-guerin-qos-routing-ospf-04.txt, Dec. 1998.

[11] R. Guerin, D. Williams, and A. Orda, "QoS routing mechanisms and OSPF extensions," in *Proc. Globecom*, 1997.

[12] M. Kodialam and T. V. Lakshman, "On-line routing of guaranteed bandwidth tunnels," in *Proc. 7th IFIP Workshop Performance Modeling Evaluation ATM/IP Networks*, June 1999.

[13] D. Katz and D. Yeung, "Traffic engineering extensions to OSPF," work in progress, Internet Draft, 1999.

[14] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," work in progress, Internet Draft draft-ietf-mpls-arch-02.txt, July 1998.

[15] H. Smit and T. Li, "IS-IS extensions for traffic engineering," work in progress, Internet Draft, 1999.

[16] S. Plotkin, "Competitive routing of virtual circuits in ATM networks," *IEEE J. Select. Areas Commun., Special Issue Advances Fundamentals Networking*, pp. 1128–1136, 1995.

Koushik Kar received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1997, and the M.S. degree in electrical and computer engineering from the University of Maryland, College Park, in 1999, where he is currently working towards the Ph.D. degree.

His research interests include scheduling in high-speed switches, routing and congestion control, and fairness and pricing issues in communication networks.

Murali Kodialam (M'99) received the Ph.D. degree in operations research from Massachusetts Institute of Technology in 1991.

He has been working at Bell Labs. since October 1991. He is currently in the Performance Analysis Department where he works on resource allocation and performance of communication systems including routing in MPLS systems, topoplogy construction, and routing in ad hoc wireless networks, and reliable routing in optical networks.

Dr. Kodialam is a member of INFORMS.

T. V. Lakshman (S'84–M'85–SM'98) received the Ph.D. degree in computer science from the University of Maryland, College Park; and the Master's degree from the Department of Physics, Indian Institute of Science, Bangalore, India.

He is currently a Director in Bell Labs. Research. Previously, he was at Bellcore, where he was most recently a Senior Research Scientist and Technical Project Manager in the Information Networking Research Laboratory. His recent research has been in issues related to traffic characterization and provision of quality of service, architectures and algorithms for gigabit IP routers, end-to-end flow control in high-speed networks, traffic shaping and policing, switch scheduling, routing in MPLS, and optical networks.

Dr. Lakshman is a corecipient of the 1995 ACM Sigmetrics/Performance Conference Outstanding Paper Award, and the IEEE Communications Society 1999 Fred. W. Ellersick Prize Paper Award. He is an Editor of the IEEE/ACM Transactions on Networking.