Profile-Based Routing: A New Framework for MPLS Traffic Engineering

Subhash Suri* Marcel Waldvogel† Priyank Ramesh Warkhede‡

Abstract—**We present a new algorithm and framework for dynamic routing of bandwidth guaranteed flows. The problem is motivated by the need to dynamically set up bandwidth guaranteed paths in carrier and ISP networks. Traditional routing algorithms such as minimum hop routing or widest path routing do not take advantage of any knowledge about the traffic distribution or ingress-egress pairs, and therefore can often lead to severe network underutilization. Our work is inspired by the recently proposed "minimum interference routing" algorithm (MIRA) of Kodialam and Lakshman, but it improves on their approach in several ways. Our main idea is to use a "traffic profile" of the network, obtained by measurements or service level agreements (SLAs), as a rough predictor of the future traffic distribution. We use this profile to solve a** *multicommodity network flow* **problem, whose output is used both to guide our** *online* **path selection algorithm as well as impose admission control. The offline multicommodity solution seems very effective at distributing the routes and avoiding bottlenecks around hot spots. In particular, our algorithm can anticipate a flow's blocking effect on groups of ingress-egress pairs, while MIRA only considers one ingress-egress pair at a time. Our simulation results show that the new algorithm outperforms shortest path, widest path, and minimum interference routing algorithms on several metrics, including the fraction of requests routed and the fraction of requested bandwidth routed. Finally, the framework is quite general and can be extended in numerous ways to accommodate a variety of traffic management priorities in the network.**

I. Introduction

We present a new algorithm and framework for dynamic routing of bandwidth guaranteed flows. Our algorithm is *online*, meaning that it routes requests one at a time, without specific knowledge of future demands. We use quasi-static information about the network and traffic to select paths so as to minimize the number of requests that are rejected or the network bandwidth that is wasted. Clearly, if no assumptions are made about the flow requests, a pathologically chosen set of requests can foil *any* online algorithm. We make minimal assumptions that are justifiable in practice and lead to significant improvement in network utilization. In particular, we assume that the ingress and egress nodes in the network are known, and that a *traffic profile* between pairs of ingress-egress nodes is also known. The traffic profile between ingress-egress node pairs can be either measured or inferred from service level agreements (SLAs). Our algorithm uses this quasi-static information in a *preprocessing* step (one multi-commodity flow computation), to determine certain bandwidth allocations on the links of the network. The online phase of the routing algorithm then routes tunnel requests using a "shortest path" (SPF) like algorithm *but with the additional information given by the preprocessing phase.* The multi-commodity preprocessing phase allows the online algorithm to exercise *admission control* by rejecting some requests because of their blocking effects in the network.

The motivation for our problem arises from the needs of service providers who must dynamically reserve bandwidth guaranteed routes in carrier and ISP networks. Following Kodialam and Lakshman [8], we will describe our algorithms in the context of setting up paths in Multi-Protocol Label Switched (MPLS) networks, although our algorithms are applicable in other contexts as well. MPLS networks [10] allow explicit routing of packets by putting labels on them, which can then be used to forward packets along specific Label Switched Paths (LSPs). Service providers can perform this encapsulation at the ingress routers, and then use LSPs to implement Virtual Private Networks (VPNs) [6] or satisfy other quality of service (QoS) agreements with clients. At the ingress routers, packet classification [9], [11], [12] can be used to map packets into "forwarding equivalence classes" by examining packet headers. This aggregation (mapping into equivalence classes) also has the potential advantage of smoothing out the bandwidth requirement across many bursty streams. In addition, the service providers can use a measurement-based mechanism to build a traffic profile for an ingress-egress node pair. Such a profile can be as simple as an average bandwidth requirement over a certain

*Department of Computer Science, Washington University, St. Louis, MO 63130, USA. Research partially supported by NSF Grant ANI 9813723.

†Department of Computer Science, Washington University, St. Louis, MO 63130, USA.

‡Department of Computer Science, Washington University, St. Louis, MO 63130, USA. Research supported by NSF Grant ANI 9813723.

time period.

A Label Switched Path requires set up, meaning that all the intermediate routers between the ingress and egress nodes are specified. The path is set up using a signaling protocol such as RSVP [3] or LDP (Label Distribution Protocol [1]). The ability to specify explicit paths for any flow gives the service providers an important tool to engineer how their traffic is routed, and thereby improve the network utilization, by minimizing the number of requests that are rejected when the network becomes overloaded. Current intra-domain routing schemes, which forward packets based on destination address only, do not take into account what other flows are currently, or likely to be, requested. Thus, their routing behavior is highly myopic—they will reject a flow when the default shortest path route become congested, even if an alternative path is available. The algorithms like widest path routing also suffer from similar problems. We therefore need better schemes for routing flow requests that take better advantage of the network infrastructure, network topology, and traffic distribution. We show that this problem is NP-Complete even in highly simplified form, but propose a novel multi-commodity based framework, which eliminates many of the shortcomings of shortest path routing, widest path routing, and even minimum interference routing.

While we present our algorithm in the context of bandwidth guarantees, it can also perform routing based on other QoS metrics such as delay, loss etc. As pointed out by Kodialam and Lakshman [8], if additional constraints, such as delay or loss, are to be incorporated into SLAs, one can do so effectively by converting those requirements into a bandwidth requirement.

Our framework is quite general, and it can be extended and generalized in multiple ways to handle additional metrics and requirements. In particular, the multi-commodity flow formulation permits a *cost* function, which we minimize to achieve optimal routing. In order to minimize the number rejected requests, we use the simple *linear cost function.* A variety of *non-linear* cost functions can be used to handle features like *minimum guaranteed bandwidth* or *fairness* across multiple flows.

II. Routing Requirements

In this section, we briefly discuss the requirements that a flow routing algorithm must satisfy. Kodialam and Lakshman [8] give a detailed list of ten important criteria that a dynamic path selection algorithm must meet. We discuss only the most important requirements here.

- [*Routing without splitting flows*] It is assumed that the flow should be routed on a single path, without splitting. Many flow requests may involve traffic that is inherently unsplittable (circuit emulation or voice), and therefore it is important to route them on single paths. Thus, for each flow request, the algorithm must find a path with desired amount of bandwidth between the ingress and egress nodes, or determine that the flow is unroutable.
- [*Online routing*] We assume that the individual flow setup requests arrive online, one at a time, and the algorithm must process each request without having to know the future requests. In network provisioning and design phase, it is customary to assume that exact point-to-point demands are known. But that assumption is highly impractical for the MPLS tunnel set up problem. While we make use of the quasi-static information such as traffic profile in our algorithm, those profiles are used only as a rough guide for the aggregate demands. Furthermore, our routing algorithm is completely online—it does not need to know anything about individual requests, their bandwidth requirements, or their time of arrival. Of course, if the actual demands in aggregate deviate significantly from the assumed profile, the performance improvement achieved by our algorithm may degrade, but that is to be expected for any online algorithm.
- [*Computational requirement*] We want the path selection algorithm to be quite fast and scalable. Individual flow setup requests are typically processed at the ingress routers or switches, which operate at very high load and have limited computing power. Thus, the computational requirement per flow setup request must be kept as low as possible. In this regard, our algorithm is just as efficient and simple as the shortest path algorithm, and substantially faster than Kodialam-Lakshman algorithm. The expensive part of our algorithm is the preprocessing phase, which is run very infrequently and offline, only when the quasi-static information changes. The online algorithm runs a single breadth-first search algorithm, which is several *orders of magnitude faster* than the max flow computations needed by MIRA [8].
- [*Policy constraints*] A good path selection algorithm should be able to incorporate additional policy constraints. For example, a service level agreement may require avoiding links with certain loss rate. Similarly, SLAs may require a minimum flow acceptance guarantee; for example, over a period of one hour, flows with total bandwidth at least 100 Mbps must be accepted. In Section IX, we describe mechanisms to implement policy constraints into the framework.
- [*Traffic profile*] Our algorithm uses information about "expected" flows between some ingress-egress nodes. We explain the exact form of this information later, but briefly speaking our belief is that yesterday's traffic between an

ingress-egress pair can serve as a good predictor for today's traffic. This should be especially true in light of fact that service providers aggregate a large number of flows, using forwarding equivalence classes, for the ingress-egress pairs. Service providers can have multiple classes per ingress-egress pair, and keep separate profiles for various classes. These profiles can be either measurement based, or they can be inferred from service level agreements.

- Finally, like shortest path routing, our algorithm also uses only the link-state information and, like the widest path routing algorithm, it uses some auxiliary capacity information. In order to keep the presentation simple, We describe our algorithm for the centralized route serve model, though it can also be implemented in the distributed mode.

III. Review of Existing Algorithms

The most commonly used algorithm for routing LSPs is the *shortest path routing*. In the shortest path routing, the path with the least number of links between ingress and egress nodes is chosen. The routing algorithm keeps track of the current *residual capacity* for each link, and only those links that have sufficient residual capacity for the new flow are considered. The shortest path algorithm is very simple, but it can also create bottlenecks for future flows, and lead to severe network under-utilization. (See examples in Section V.) Our new proposed algorithm is just as efficient and fast as the shortest path algorithm (during the path selection phase), but by using additional information about the network and traffic in a preprocessing phase, we can significantly reduce the number of requests that might be rejected due to inappropriate route selection. Instead of a full-fledged shortest path algorithm that has to deal with weights, which have undergone heavy manual tuning by the network operators to achieve just the desired traffic distribution, our algorithm could even use the simpler "minimum hop" algorithm (which is just a breadth-first search) to select a path in the online phase, thanks to the powerful preprocessing).

Guerin et al [7] propose a variant of the shortest path algorithm, called widest shortest path (WSP), where they choose a feasible shortest path path that has the largest residual capacity—in other words, the smallest link residual capacity along the path is maximized. While WSP certainly improves on the shortest path routing, it also has a myopic behavior—since the algorithm does not make use of the ingress-egress pairs or the traffic characteristics, it can create bottlenecks. More significantly, neither the shortest path nor the widest shortest path routing algorithm impose any form of *admission control*. Thus, these algorithms will always accept a flow if there is a feasible path in the network, even if accepting that flow has the *potential to block off a large number of future flows.* The example in Figure 2 dramatically illustrates the effect of admission control—without admission control, one can force any online algorithm to achieve close to zero network utilization!

The work most closely related to ours, and indeed the basis for our work, is the "minimum interference routing algorithm" (MIRA) of Kodialam and Lakshman [8]. MIRA is quite a bit more sophisticated algorithm than either shortest path or WSP, and it takes critical advantage of ingress-egress pairs. The basic observation in [8] is that routing a flow along a path can reduce the *maximum permissible flow* between some other ingress-egress pairs. They call this phenomenon "interference." Their thesis is that if paths that reduce a large amount of possible max-flow between other ingress-egress pairs are avoided, creation of bottlenecks can also be avoided. Their algorithm performs multiple max-flow computations to determine the path of least interference.

The idea of minimizing interference is a good one, but we believe it has several limitations. First and foremost is the observation that MIRA focuses exclusively on the interference effect on *single* ingress-egress pairs. It is not able to estimate the bottleneck created on links that are critical for *clusters* of nodes. (See examples in Section V.) Second, MIRA considers simply the *reduction* in the maximum flow between a pair, without regard to the expected bandwidth between that pair. Thus, MIRA might reject a flow request even though the network retained sufficient residual bandwidth to route the flow between the affected pair. Finally, MIRA is computationally very expensive. While shortest path, widest shortest path, and our new proposed algorithm all perform a single shortest path computation to route a request, MIRA performs *hundreds of maximum flow computations*, each of which is several orders of magnitude more expensive than the shortest path calculation.

IV. Problem Statement

We model the network as a graph $G = (V, E)$, where V is the set of routers and E is the set of links. The current *residual* capacity of a link $e \in E$ is denoted $cap(e)$—this is the additional bandwidth that can be routed on link e. A subset of routers are assumed to be ingress-egress routers, between which label switched paths (LSPs) can be set up. We assume that the ingress-egress pairs are known, and that this information is quasi-static, meaning it changes very infrequently. An example is shown in Figure 1, which is borrowed from Kodialam-Lakshman [8]. We call this network the KL-graph, and it is one of the several networks on which we report our simulation results.

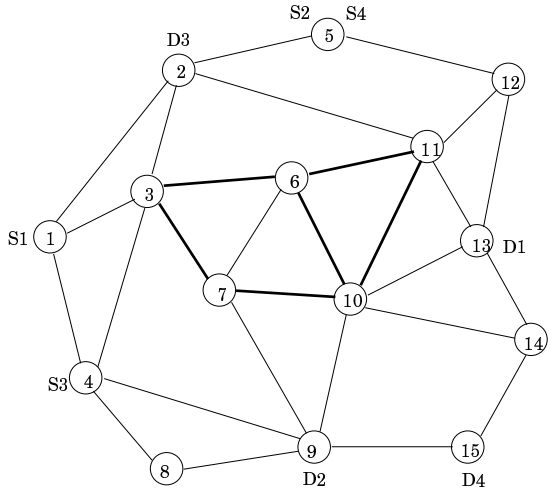

Fig. 1. An example network, showing ingress-egress nodes. This network borrowed from [8] is referred to as KL-graph in our paper.

A request for an LSP setup is defined by a quadruple (id, s_i, d_i, b_i), where id is the request ID, s_i is the ingress (source) router, d_i is the egress (destination) router, and b_i is the bandwidth requested for the LSP. (The reason for having a separate id for each request is that there can be multiple request for the same (s_i, d_i) pair.) As mentioned earlier, all QoS requirements for the flow are assumed to have been folded into the bandwidth b_i. Given a request (id, s_i, d_i, b_i), the algorithm can either accept it, in which case it must find a path in the network from s_i to d_i along which each link has residual capacity at least b_i, or the algorithm may reject the request. The admission control feature allows our algorithm to reject a request even if there is a feasible path—this may happen if the algorithm determines that accepting this request may create a significant bottleneck for future requests (based on its knowledge of the ingress-egress pairs and their traffic profile). We assume that all LSP set up requests arrive online, one at a time, and the algorithm does not know anything about individual future requests, their bandwidth requirements, or their time of arrival.

The traffic profile information used by our algorithm records the expected flow between pairs of ingress-egress routers, and represents an aggregated demand profile between ingress-egress pairs. Such information can be either measurement-based or it can be calculated from SLAs that have been entered by a service provider with its clients. Each traffic profile is also defined by a quadruple: $(classID, s_i, d_i, B_i)$, where $classID$ is the traffic class, s_i, d_i are the ingress and egress nodes, and B_i is the aggregate traffic to be expected for this class between s_i and d_i. Between the same s_i, d_i pair, there can be multiple traffic classes (corresponding to different service types offered by the provider). Each LSP request can be mapped to a unique traffic profile class. (Conversely, a traffic profile class acts as an aggregate proxy for all the LSP requests mapped to it.)

The traffic profile is a rough indication of the amount of traffic that can be expected between a pair; the LSP set up request sequence however arrives online. A convenient way to think about this is that total sum of all LSP requests between s_i and d_i for the class i is a *random variable* with mean B_i. But the time of arrival of individual requests and their bandwidth requirements are entirely unpredictable. Thus, as for as our routing algorithm is concerned, the request sequence is completely online.

For simplicity, we assume there is a route server that knows the current network topology and available link capacities. Instead of dedicating a single machine to perform route computations, this job could also be shared among all ingress nodes without changes to the framework.

V. Examples Illustrating Limitations of Existing Routing Algorithms

In this section, we informally describe the shortcomings of existing routing algorithms using some simple illustrative examples. Our basic theme is that algorithms that do not adapt to the traffic distribution in the network (taking advantage of ingress-egress pairs and some rough estimate of the traffic flow between pairs) will always lead to suboptimal network utilization, which can be quite severe in some cases. In particular, the routing by algorithms like shortest path and WSP that do not impose any form of admission control can occasionally lead to significant bottlenecks. Simply having more information about the network or traffic does not guarantee better routing. Our proposed framework assumes minimal information about the network and traffic, which we believe can be easily obtained. Our algorithm, though as simple and computationally efficient as shortest path, leads to fewer rejected requests and better network utilization.

We use three simple examples to illustrate the shortcomings of existing routing algorithms. In order to drive home the point, these examples are necessarily artificial looking, but their general form is not at all unusual. In fact, real networks are quite likely to contains subgraphs resembling the *concentrator* or the *distributor* example. The parking lot topology is common as well, but also depends on the selection of ingress-egress pairs. Since pair selection is often outside the influence of the ISP, the occurrence of this pathological case is likely to appear in the real world.

- [**Parking Lot**] Figure 2 shows a simple network with $3n + 3$ nodes. The ingress-egress pairs for the LSP set up requests are $(S_0, D_0), (S_1, D_1), \ldots, (S_n, D_n)$, and the

bandwidth requested for each LSP is 1. All link capacities in the network are either 1 or $1 + \varepsilon$, as shown.

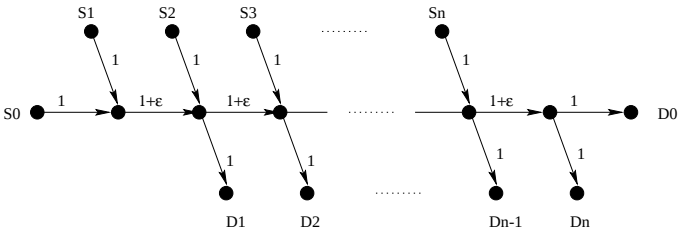

Fig. 2. The **parking lot** topology PL.

Suppose the online sequence of LSP requests arrive in the order $(S_0, D_0), (S_1, D_1), \ldots, (S_n, D_n)$. Accepting the request (S_0, D_0) completely chokes off the networks—no other LSP request can be satisfied. However, since neither the shortest path nor WSP rejects flow requests if there is a feasible path, they will accept (S_0, D_0), resulting in the total network utilization of 1. An optimal algorithm will reject (S_0, D_0), and will accept $(S_1, D_1), \ldots, (S_n, D_n)$, for a total network utilization of n.

The choice of capacity $1+\varepsilon$ for the links along the spine of the parking lot also foils MIRA—since these links are not in the min cut for any (S_i, D_i) pair, and are not considered critical. Thus, MIRA also accepts the first flow request, and ends up rejecting all other requests.

Although the links are drawn as directed, path selection and blocking behavior would remain the same for bidirectional links. In the following two examples, some of the links need to be unidirectional. Even though unidirectional links are rare (i.e., satellite downlinks and downstream-only cable modem installations), unidirectional remaining capacity is quite common. Due to asymmetric links or loads, the remaining capacity in the opposite direction could become too small to be useful.

• [**Concentrator**] Figure 3 shows a network, which we call a *concentrator graph*—one node C acts as a feeder for n ingress nodes $S_1, \ldots, S_n$. The concentrator node C is connected to a high capacity link, *fat pipe*, of capacity $n + 1$, whose other endpoint is a egress node D. One high bandwidth ingress S_0 is also connected to the concentrator, through a capacity n link. S_0 is also connected to D via an alternative 3-hop path, of capacity n.

In this example, an online sequence of $n + 1$ requests arrive (S_0, D), $(S_1, D), \ldots, (S_n, D)$. The first request has bandwidth requirement n, while all others have bandwidth requirement 1. Using either the shortest path or the WSP, one would route the first request through the concentrator node (using 2 hops). This leaves residual capacity 1 along the link CD, and so of the remaining n requests at most one can be satisfied.

This example also illustrates the shortcoming of MIRA—

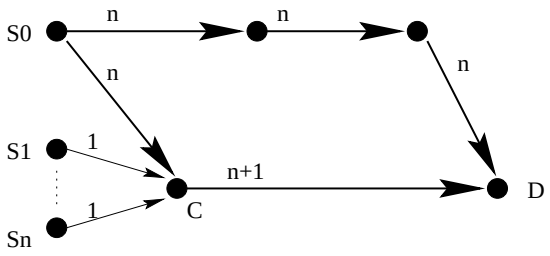

Fig. 3. The concentrator topology CN.

the fat link CD is not in the minimum cut for any *individual* ingress-egress pair. Thus saturating it doesn't seem harmful to MIRA. So, the MIRA algorithm will also choose incorrect paths in this scenario. The optimal algorithm will route the (S_0, D) request along the top alternative path, and use the fat link to route the n 1-unit requests from S_i to D.

• [**Distributor**] While the preceding example shows why it may be a good idea to not use the fat pipe sometimes, our next example shows that the converse is also true.

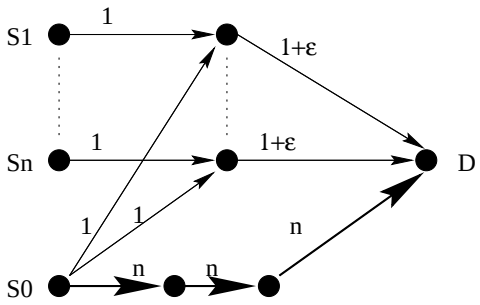

Fig. 4. The distributor topology DS.

In this example, we get n requests between S_0 and D, each of bandwidth 1. In addition, we also get n requests between each S_i and D, also of bandwidth 1. Again, the shortest path algorithms (shortest path and WSP) will use the two-hop paths, for each of the first n requests, choking off the $1 + \varepsilon$ links. Thus, each of the remaining n requests between S_i and D are rejected. The routes selected by MIRA are also the same, since the links of capacity $1 + \varepsilon$ are not in the minimum cut for any S_i, D pair. By contrast, the optimal algorithm will route all the requests from S_0 along the bottom fat path (3 hops), leaving the top 2-hop paths for S_i to D requests.

The preceding examples are meant to illustrate how a bad path selection for one flow can create significant bottlenecks for future flows. An online routing algorithm that does not have any additional information about the flows can perform quite poorly in the worst case. As the parking lot example shows, in some cases these algorithms cannot guarantee that even 1% of the network bandwidth is utilized, whereas the optimal algorithm achieves 100%. We

build on the work by Kodialam and Lakshman [8] and propose a new algorithm as well as general framework, where we exploit information about the ingress-egress nodes as well as a measured (or estimated) traffic profile to perform both path selection and admission control. Our algorithm is both simpler than MIRA and it also performs better in many cases where MIRA falls into the same traps as the shortest path or widest shortest path routing algorithms.

VI. Multi-Commodity Flows

We begin with the observation that even if the exact sequence of tunnel requests were known in advance, the problem is intractable. In particular, given an offline sequence of LSP set up requests, it is NP-Complete to determine what is the maximum number of requests that can be simultaneously routed. Thus, the difficulty is not necessarily in the online nature of the problem—rather it lies in having to choose which of many paths to select for routing a flow. We turn this difficulty around by formulating the offline problem as a *multi-commodity flow problem*, on a modified network. We use the traffic profile data for the ingress-egress pairs as the offline aggregate data. The solution to the multicommodity flow problem is then used to *pre-allocate* link capacities to various flows, which are then used by the online algorithm to perform path selection. When the allocated capacity for a flow becomes zero (or was assigned zero from the beginning), that flow request is rejected—even though there might be sufficient capacity in the network to route that flow. Let us begin with some preliminaries about multicommodity flows. Interested reader can find a comprehensive treatment of network flows in the book [2].

Given a directed graph $G = (V, E)$, with positive capacity $cap(u, v)$ for each edge (u, v), a *flow* on G is a real-valued function f on node-pairs having the following properties:

- [**Skew Symmetry**] $f(v, w) = -f(w, v)$. If $f(v, w) > 0$, then we there is a flow from v to w.
- [**Capacity Constraint**] $f(v, w) \leq cap(v, w)$. If (v, w) is not an edge of G, then we assume that $cap(v, w) = 0$.
- [**Flow Conservation**] For every vertex v, other than the source or the sink (i.e. ingress or egress), the flow is conserved: $\sum_w f(v, w) = 0$.

It is straightforward to prove that the problem of determining whether a given (offline) set of LSP requests can be routed is NP-Complete.

Theorem VI.1. *Given a network $G = (V, E)$, where each link has a positive capacity, and a set of k LSP requests (id, s_i, d_i, b_i), for $i = 1, 2, \ldots, k$, deciding whether it is possible to simultaneously route all k requests in G is* NP-Complete.

Indeed, the LSP routing problem is a generalization of the simple two-commodity integral flow problem, which is known to be NP-Complete [5]. The 2-commodity integral flow problem asks whether it is possible to find two flow functions that deliver some required set of flows from two source nodes to two sink nodes. Specifically, suppose we are given a directed graph $G = (V, E)$, node pairs (s_1, d_1), (s_2, d_2), positive integral capacity $cap(e)$ for each edge $e \in E$, and bandwidth requirements b_1 and b_2. Then, it is NP-Complete to decide if there are flow functions f_1, f_2 such that (1) for each link $e \in E$, $f_1(e) + f_2(e) \leq cap(e)$, (2) for each node other than s_1, s_2, d_1, d_2, flows f_1 and f_2 are conserved, and (3) the net flow to d_i under f_i is at least b_i.

We are now ready to describe the details of our algorithm.

VII. Profile-Based Routing

Examining the problem more closely, we find that the intractability of LSP set up problem stems from two requirements: unsplittability of the flows, and separate demand functions for each flow. In other words, if flows are allowed to be split, and if the objective is to maximize *total* flow rather than to satisfy each individual flow, then the problem can be solved efficiently through linear programming. Unfortunately, in the LSP problem, we do not want flows to be split, and we do want to enforce some kind of fairness so as to admit as many flow as possible. Fortunately, we are able to finesse the problem on both counts by using a multi-commodity flow framework on the *traffic profiles*, rather than individual flows. First, the individual flow requests sizes are typically much smaller than the link capacities—for instance, the link capacities might be range from OC-12 to OC-192, while a typical request might be just a few megabits per second. Second, we use the multicommodity flow in the preprocessing phase, where "commodities" correspond to highly aggregated traffic profiles, and not individual LSP requests. So, when a commodity is split, it does not mean that a flow is split; rather it just means that a "group" of flows is routed on a different path than another group. An individual LSP request is never split—our algorithm either finds a single path to route it, or reject it.

Our algorithm has two phases: a preprocessing phase, where we solve a multicommodity flow problem to preallocate link capacities for various traffic classes; and an online routing phase, where each LSP request is routed online using a shortest path like algorithm. Let us first describe the preprocessing phase.

A. Multi-Commodity Flow Preprocessing

The input to the preprocessing phase is the network $G = (V, E)$, with capacity $cap(e)$ for each edge $e \in E$. We are given a set of traffic profiles $(classID, s_i, d_i, B_i)$, where $classID$ is the traffic class, s_i, d_i are the ingress and egress nodes, and B_i is the aggregate bandwidth requirement for this class between s_i and d_i. We treat each traffic class as a separate *commodity*. Suppose there are k commodities, numbered 1 through k. The goal is to find routes in the network to send as much of each commodity as possible from its source node to the destination node.

As noted earlier, satisfying all bandwidth requirements however may not be possible. We therefore put additional edges in the network, called *excess edges*, so that the problem always have a feasible solution, and use *edge costs* to distinguish between the network edges and the excess edges. In particular, we add an **infinite capacity** excess edge between each ingress-egress pair, as shown in Figure 5. Thus, $cost(e) = 1$ if $e \in E$, and $cost(e) = cap(e) = \infty$ if e is an excess edge, where ∞ is an appropriately large number. The large cost of the excess edges forces as much of the feasible flow as possible to go through original network edges. Let G' denote the graph obtained by adding these excess edges.

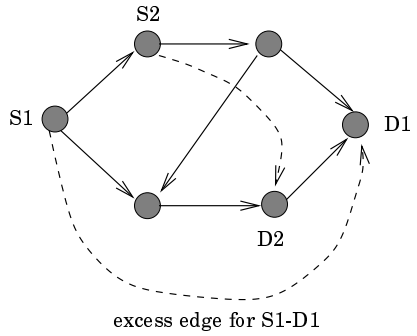

Fig. 5. The excess edges added to make the multicommodity flow always feasible. The cost of each excess edge is ∞, a large constant, while all other edges have cost one.

Now, let $x_i(e)$ denote a real-valued variable, denoting the amount of commodity i that is routed through edge e. Then, the multicommodity problem to be solved for graph G' is to

$$\text{minimize} \quad \sum \left(cost(e) \sum_{i=1}^{k} x_i(e) \right)$$

subject to the following constraints:

- capacity constraints are satisfied for all edges—if e is not an excess edge, then $\sum_{i=1}^{k} x_i(e) \leq cap(e)$,
- the flow for each commodity is conserved at all nodes, except the corresponding ingress and egress nodes,
- the amount of commodity i reaching its destination, d_i, is B_i.

The output of the multicommodity flow computation is the values for the variables $x_i(e)$. We use these values to set a *pre-allocation* of e's capacity for various flows. In other words, $x_i(e)$ part of e's capacity will be used by the online algorithm to route flows belonging to the traffic class i. In summary, the multi-commodity phase of the algorithm determines admission control thresholds for each traffic class, and computes pre-allocation of link capacities to maximize network utilization. The online routing phase of the algorithm is described next.

B. Online Path Selection for LSP Requests

The input to this phase of the algorithm is the input graph $G = (V, E)$, where for each edge $e \in E$, we keep track of the *residual capacity* $r_j(e)$ for each traffic class $j = 1, 2, \ldots, k$. (Note than these residual capacities are per traffic class, not per flow.) The initial value for $r_j(e)$ is set to $x_j(e)$, which is the output of the multi-commodity preprocessing phase. The algorithm then process an online sequence of LSP set up requests (id, s_i, d_i, b_i), where id is the request ID, s_i is the ingress (source) router, d_i is the egress (destination) router, and b_i is the bandwidth requested for the LSP. We assume that each LSP can be mapped (by the ingress router s_i) to a unique traffic class. Our online routing algorithm runs on the reduced graph, which uses the pre-allocated capacities corresponding to this class. In this reduced graph, we select a minimum hop path between s_i and d_i, if one exists.

PROFILE BASED ROUTING

Input: The input graph $G = (V, E)$. For each edge e, we maintain residual capacity $r_j(e)$ for each commodity (traffic class) $j = 1, 2, \ldots, k$. The LSP request is between an ingress-egress pair s, d, and the bandwidth requirement is b. Let j be the traffic class to which this LSP belongs.

Output: A path from s and d, such that for each edge e along this path there had been $r_j(e) \geq b$ (during the algorithm, $r_j(e)$ is updated to contain the updated residual bandwidth).

Algorithm:

1. Delete from G all edges e for which $r_j(e) < b$. (These edges have insufficient residual capacity for class j.)
2. In the reduced graph, find a path P with minimum number of hops, using a breadth first search, from s to d.
3. For each edge e in the path P, decrease the residual capacity $r_j(e)$ by b.
4. Route LSP (s, d, b) along the path P.

C. Complexity Analysis

If the network has N nodes and M edges, the breadth first search algorithm computes a shortest path in $O(N + M)$ time. This is a linear-time algorithm, and should be several orders of magnitude faster than the MIRA algorithm, which needs to perform several hundred (as many as the number of ingress-egress pairs) maxflow computations. *Each maxflow computation itself takes $O(N^3)$ time.* Thus, during path selection phase, our algorithm has the same run time complexity as the currently used shortest path algorithm. Our algorithm is faster than the widest shortest path routing algorithm, because that algorithm must execute a Dijkstra style shortest path computation.

The preprocessing phase of our algorithm solves a minimum cost multi-commodity flow problem, which can be slow. But that step can be executed offline, and does not require recomputation unless the network information changes, such as ingress-egress pairs or their traffic profile. Those changes are very infrequent. Thus, our algorithm needs occasional heavy preprocessing to build a pre-allocation table, which it uses to run the online path selection phase.

VIII. Performance Results

Without real network topologies and large amounts of traffic data, it is difficult to perform meaningful and conclusive experiments. We will follow the tradition set by other authors, and perform experiments on several handcrafted topologies, using both worst-case and synthetic flow data. We present qualitative as well as quantitative evidence for why we believe our Profile-Based Routing algorithm should (and does) perform better than others. One very attractive feature of our algorithm is that it is computationally as efficient as the shortest path or widest path routing, and substantially faster than MIRA.

We used four network topologies to measure the performance of our Profile-Based Routing (PBR) algorithm. The first three topologies are the ones we used for illustration in Section V. The fourth topology, called KL, is the one used by Kodialam and Lakshman [8] in their experiments. In the parking lot topology (PL), all link capacities are set to 4800 (to model OC-48). In the concentrator (CN) and distributor (DS) topologies, we used $n = 5$, and scaled up all link capacities by 800. Thus, all links with capacity 1 in Figure 3 and capacity $1 + \varepsilon$ in Figure 4 become links of capacity 800, while those with capacity n or $n + 1$ become links of capacity 4800. In the network KL, all light edges have capacity 12, while dark ones have capacity 48 (meant to model OC-12 and OC-48 links, respectively). In their paper, Kodialam and Lakshman also used a scaled-up version of their network, in which capacity of links 2–3, 2–5, and 14–15 is increased to 48, and then all capacities are multiplied by 100. Finally, we used a publically available implementation of the minimum cost multi-commodity flow algorithm, PPRN package, for our preprocessing phase (available at `http://www-eio.upc.es/~jcastro/pprn.html`).

A. Worst-Case Results

We did not have access to an implementation of MIRA or WSP for our studies, so we compared the performance of our algorithm with the shortest path routing. In the absence of those implementations, we were also unable to compare worst-case performance of those algorithms. We can, however, infer their behavior on the three constructed network topologies, namely, the parking lot (PL), the concentrator (CN) and the distributor (DS). Table I documents these results.

TABLE I
WORST-CASE PERFORMANCE IMPROVEMENT.

Graph Name	Total Req.	Requests Routed by SPF	MIRA	PBR	Factor of Improv.
PL	$1+n$	1	1	n	n
CN	$2n$	n	n	$2n$	2
DS	$2n$	n	n	$2n$	2

In the parking lot topology (PL), if the first request is between the nodes S_0 and D_0, then all three algorithms (shortest path, WSP, and MIRA) accept it, which blocks all future requests from being routed. Our new algorithm rejects the first request, and is then able to satisfy all remaining n requests between S_i and D_i. As the number of ingress-egress pairs n increases, the percentage of network utilized by shortest path, WSP, or MIRA goes to zero.

Kodialam and Lakshman also propose a cost threshold (total weight $< W$) for admission control. However, that modification of MIRA also doesn't work for this topology since none of the edges used by the first path from S_0 to D_0 are in the minimum cut of any S_i, D_i pair, and consequently the weights of these edges remain zero.

In the concentrator topology (CN), a single request of size n by source S_0 will be routed by both shortest path and MIRA along the path that goes through the concentrator node C, which then blocks all future requests between S_i and D. In this case, the edge CD is not found to be critical by MIRA because it does not belong to the minimum cut of any single ingress-egress pair; it is only in the minimum cut for a cluster of ingress-egress pairs. Thus, in this case,

PBR routes all $2n$ units of traffic, while the other three algorithm route only n units.

The same performance is also observed in the distributor topology (DS).

B. Simulation Results

We next carried out a series of experiments to measure the performance of PBR relative to the shortest path algorithm, using randomly generated request sequence. In each experiment we generated a random sequence of individual flow requests, and measured the performance of our profile based routing as well as the minimum hop routing. The performance was measured both in terms of the number of flows routed, as well as the total bandwidth request that was satisfied.

Following Kodialam and Lakshman [8], we varied the bandwidths requested by individual flows between 1 and 4. This was intended to capture the fact that individual flow requests are much smaller than link capacities. However, we also ran tests to evaluate the effect of larger individual flow bandwidths.

The individual flow requests are generated in proportion to their traffic profile data. That is, if a flow belongs to traffic class i, and the total bandwidth of class i is B_i, then a flow from this class was generated with probability $B_i / \sum_j B_j$. In some cases, the traffic profile data was generated manually. In others, we used the multi-commodity flow algorithm to find feasible aggregated flows, which were then used as the profile data. Because of the random process, the expected amount of traffic requested for a traffic class was a random variable, with mean set to the profile value of that class.

C. SPF Routing vs. PBR

Clearly, when the network is lightly loaded, the shortest path routing, or any other routing, is expected to do well. The beneficial effects of admission control and good path selection become evident only when the network is at least partially congested. Towards this end, we first generated enough requests that almost all paths between all ingress-egress pairs were saturated. The number of requests satisfied at this point is used as an indication of how important path selection and admission control are in improving the network utilization. In each experiment, the results are averaged over several runs so as to smooth out any effects of random request generation.

Table II shows the results of this experiment, by measuring the total amount of bandwidth that is routed by the two algorithms.

Table III shows the results of the same experiment, but this time measures the total number of flow requests that were accepted by the two algorithms. Thus, even for random requests, the PBR seems to consistently outperform the shortest path algorithm.

TABLE II
TOTAL BANDWIDTH ROUTED BY THE SHORTEST PATH ALGORITHM VS. PBR.

Graph	SPF	PBR	Improvement
CN	$7,323$	$8,799$	20.16%
DS	$6,474$	$7,200$	11.21%
KL	$7,913$	$8,400$	6.15%
KL2	$7,863$	$8,400$	6.83%
PL	$14,686$	$23,999$	63.41%

TABLE III
TOTAL NUMBER OF REQUESTS ROUTED BY THE SHORTEST PATH ALGORITHM VS. PBR.

Graph	SPF	PBR	Improvement
CN	2,921	3,509	20.13%
DS	2,616	2,896	10.70%
KL	3,193	3,392	6.23%
KL2	3,137	3,355	6.95%
PL	6,017	9,570	59.05%

Our next experiment tried to evaluate the effect of increasing the size of the maximum bandwidth requested by an individual flow. In this experiment, the individual flow requests were generated with a random bandwidth requirement in the range from 1 to 48 (that is, the largest request size being 1% of the link capacity). The number of flows was proportionately reduced to keep the total bandwidth requested the same. The increased bandwidth size didn't make much difference. Tables IV and V show these results.

TABLE IV
TOTAL BANDWIDTH ROUTED: SHORTEST PATH VS. PBR WHEN THE LARGEST BANDWIDTH REQUESTED BY AN INDIVIDUAL FLOW IS 48.

Graph	SPF	PBR	Improvement
CN	7,463	8,757	17.34%
DS	6,570	7,153	8.87%
KL	7,837	8,396	7.14%
KL2	7,907	8,399	6.22%
PL	15,319	23,991	56.61%

TABLE V
TOTAL NUMBER OF REQUESTS ROUTED BY SHORTEST PATH ALGORITHM VS. PBR WHEN THE LARGEST BANDWIDTH REQUESTED BY AN INDIVIDUAL FLOW IS 48.

Graph	SPF	PBR	Improvement
CN	313	367	17.13%
DS	276	301	9.04%
KL	322	347	7.64%
KL2	334	354	6.08%
PL	655	990	50.99%

The number of flow set-up requests satisfied by both algorithms are compared in Table V. Fragmentation of edge capacities does not seem to have any significant impact. The improvement shows a similar form as for smaller requests, as seen in above tables. The issue of fragmentation of edge capacities is discussed in detail later.

D. Curse of Bandwidth Fragmentation

Assuming that the network traffic shows short or long-term persistent pattern, the profile data should be a good predictor of the actual observed traffic. Let us suppose for a minute that the flow requests closely matched the traffic profile used by the algorithm. How close is the performance of PBR compared to an optimal (but offline) algorithm? The multi-commodity flow solution gives an *upper bound* on how much flow is routable, which may not be achieved by our online algorithm due to *bandwidth fragmentation.* The pre-allocations $x_i(e)$ are portions of each link that are reserved for a traffic class, but because the individual flows have arbitrary bandwidth requirements, we may end up reserved capacities on different links that are insufficient to route an individual request without splitting it. How severe is the effect of this bandwidth fragmentation? A measurement of this may indicate *how close to optimal* does PBR come?

In this experiment, we generated individual requests with random bandwidth requirements in the range from 1 to $\max$. Four values of $\max$, namely, 4, 12, 48 and 192, were used, and for each value, the total number of requests generated was scaled down so as to keep total bandwidth requested about the same. Table VI shows these results. As expected, the amount of bandwidth potentially wasted due to fragmentation increases with larger requests, but the degradation is quite small for even very large individual flow requests (4% of the link capacity). Even then, there remains a substantial net gain for Profile-Based Routing in all the scenarios.

TABLE VI
FRACTION OF LINK CAPACITIES THAT CAN BE POTENTIALLY WASTED DUE TO FRAGMENTATION, AS A FUNCTION OF THE MAXIMUM SIZE OF INDIVIDUAL REQUESTS. THE PERCENTAGE WASTE IS MEASURED AS THE REDUCED AMOUNT OF TOTAL BANDWIDTH ROUTED AS COMPARED TO THE OPTIMAL.

	Max. Flow Size			
Graph	4	12	48	192
CN	0	0.10	0.49	1.32
DS	0	0.26	0.26	3.57
KL	0	0	0.012	0.43
KL2	0	0	0.01	0.51
PL	0	0.01	0.03	0.38

E. To accept or not to accept?

PBR imposes an admission control and judiciously rejects flow requests that might create significant bottlenecks in the network. The obvious hope (and expectation) is that the flows whose bottleneck we are trying to avoid will eventually be requested. What happens if those flows are not requested? In that case, PBR might have a lower network utilization that a more myopic routing algorithm, such as the shortest path algorithm. In order to evaluate this aspect of PBR, we decided to take performance snapshots at intermittent times during the online run of the algorithm. In other words, we measured what fraction of the incoming requests were accepted, at intervals of $10\%, 30\%, 50\%, 70\%, 90\%$ of the total traffic. If PBR aggressively imposes admission control in the beginning and saves network resources for flows that do not arrive for a long time, its performance should be lower in the early duration of the run. As Table VII below shows that it does not take long for the admission control of PBR to pay off. Except for one minuscule drop in the KL topology, the number of requests accepted by PBR always seems to be more than the number accepted by the shortest path algorithm.

IX. CONCLUDING REMARKS AND EXTENSIONS

We presented a new Profile-Based Routing algorithm for dynamic routing of bandwidth guaranteed paths. The online routing phase of the algorithm is as simple and computationally efficient as the commonly used min hop routing or the widest shortest path routing, and it is substantially faster than the recently proposed minimum interference routing algorithm [8]. Our algorithm improves network utilization, and accepts more flows than these other

TABLE VII
PERFORMANCE IMPROVEMENT OF PBR OVER THE SHORTEST PATH ALGORITHM AT VARIOUS POINTS DURING THE RUN OF THE ALGORITHM.

	Fraction of requests				
Graph	0.1	0.3	0.5	0.7	0.9
CN	0	0	0	15.03	25.79
DS	0	2.41	6.82	14.22	20.14
PL	0	0	0	23.02	62.32
KL	0	0	-0.38	8.54	6.34
KL2	0	0	0	6.11	7.19

algorithms, because of its improved path selection and admission control. The algorithm takes advantage of quasi-static information about the network and traffic in an offline preprocessing phase, whose output is used to both guide our *online* path selection algorithm as well as impose admission control. In particular, our algorithm is able to spot potential bottleneck links that may be in the min cut of "clusters" of ingress-egress pairs (cf. concentrator and distributor topologies), as opposed to single pairs identified by MIRA.

The multi-commodity preprocessing framework proposed in our paper is quite powerful and admits numerous extensions and generalization, which can be used to implement additional policies and requirements. Just to illustrate the ideas, we mention two such extensions.

- [*Minimum Service Level*] Suppose a service provider wants to ensure that a bursty traffic class receives at least a guaranteed minimum level of service. In other words, while the expected bandwidth of a traffic class i might be B_i, the service provider wants to ensure that at least M_i level of bandwidth is guaranteed during a certain time period. We can implement this requirement by using a different *cost function* in the objective function of our multi-commodity formulation. The objective function is augmented by an additive term C which kicks in if more than $B_i - M_i$ units are routed along the *excess edge* corresponding to this traffic class.
- [*Imposing Fairness*] A flow routing algorithm can achieve large network utilization, but may unfairly punish some clients by rejecting a disproportionate share of their flows. Service providers can implement a minimum level of fairness by ensuring that a given set of traffic classes each receives a proportionate share of total bandwidth. For a traffic class j, we add $\alpha^{B_j - M_j}$ to our objective function, which is exponential in the bandwidth not routed, for a tunable parameter α. This guarantees that one class receiving an unfairly low bandwidth allocation leads to a steep cost, and will be avoided.

REFERENCES

[1] L. Andersson, et al. LDP Specification. Internet draft (Work in Progress).

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

[3] R. Braden, D. Clark, and S. Shenker. RSVP: A New Resource ReSerVation Protocol. IEEE Network, 7(9), 1993.

[4] J. Castro and N. Nabona. Nonlinear multicommodity network flows through primal partitioning and comparison with alternative methods. In Proc. of the 16th IFIP Conference, System Modeling and Optimization, 1994.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[6] B. Gleeson, et. al. A Framework for IP Based Virtual Private Networks. Internet RFC 2764, 2000.

[7] R. Guerin, A. Orda and D. Williams. QoS routing mechanisms and OSPF extensions. In *Proc. 2nd Global Internet Miniconference*, 1997.

[8] M. Kodialam and T. V. Lakshman. Minimum Interference Routing with Applications to MPLS Traffic Engineering. In *Proc. of IEEE Infocom*, 2000.

[9] T. V. Lakshman and D. Stidialis. High Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching. *Proc. of ACM Sigcomm*, 1998.

[10] E. Rosen, A. Vishwanathan and R. Callon. Multiprotocol Label Switching Architecture. Internet draft (Work in Progress).

[11] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast Scalable Level Four Switching. *Proc. of Sigcomm*, 1998.

[12] V. Srinivasan, S. Suri and G. Varghese. Packet Classification using Tuple Space Search. *Proc. of Sigcomm*, 1999.