ELSEVIER

Computer Communications 26 (2003) 351–365

computer communications

www.elsevier.com/locate/comcom

Profile-based routing and traffic engineering

Subhash Suri[a], Marcel Waldvogel[b,*], Daniel Bauer[b], Priyank Ramesh Warkhede[c]

[a]*Department of Computer Science, UC Santa Barbara, 2111 Engineering I, Santa Barbara, CA 93106, USA*
[b]*IBM Zurich Research Laboratory, 8803 Rüschlikon, Switzerland*
[c]*Cisco Systems, San Jose, CA 95134, USA*

Received 12 May 2002; accepted 14 May 2002

Abstract

We present a new algorithm and framework for dynamic routing of bandwidth-guaranteed flows. The problem is motivated by the need to set up bandwidth-guaranteed paths in carrier and ISP networks dynamically. Traditional routing algorithms such as minimum-hop 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 as a rough predictor of the future traffic distribution. We use this profile to solve a *multi-commodity network flow* problem, whose output is used both to guide our *online* path-selection algorithm as well as to impose admission control. The offline multi-commodity 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, whereas, 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.

Keywords: Routing; Flow routing; Profile; Label switching; Multi-protocol label-switched; Traffic engineering

1. 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 a 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. This traffic profile can be measured, inferred from service-level agreements (SLAs), created by rule-of-thumb or any other mechanism suitable to the network operator. 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 Internet Service Provider (ISP) networks. Following Kodialam and Lakshman [2], 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

* Corresponding author. Fax: +1-805-893-8553.
E-mail addresses: suri@cs.ucsb.edu (S. Suri), mwl@zurich.ibm.com (M. Waldvogel), dnb@zurich.ibm.com (D. Bauer), priyank@cisco.com (P.R. Warkhede).

0140-3664/03/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0140-3664(02)00154-8

well, most notably virtual-circuit systems. MPLS networks [3] 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) [4] or satisfy other Quality-of-Service (QoS) agreements with clients. At the ingress routers, packet classification [5–7] 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 time period.

An LSP 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 [8] or Label Distribution Protocol (LDP [9]). 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 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 or drop packets and flows when the default shortest-path route becomes congested, even if an alternative path is available. Algorithms such as 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 that eliminates many of the shortcomings of shortest-path routing, widest-path routing, and even minimum interference routing.

Although 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 [2], 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 [10].

Our framework is quite general and 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. To minimize the number of rejected requests, we use the simple *linear cost function*. A variety of *non-linear* cost functions can be used to handle features such as *minimum guaranteed bandwidth* or *fairness* across multiple flows.

2. Routing requirements

In this section, we briefly discuss the requirements that a flow routing algorithm must satisfy. Kodialam and Lakshman [2] 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 the 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 set-up requests arrive online, one at a time, and the algorithm must process each request without having to know future requests. In the 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 setup problem. While we make use of quasi-static information such as traffic profiles in our algorithm, those profiles are used only as a rough guide for the aggregate demands to be expected. 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 the Kodialam–Lakshman algorithm. The expensive part of our algorithm is the preprocessing phase, which however 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 the Minimum Interference Routing Algorithm (MIRA) [2].

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,

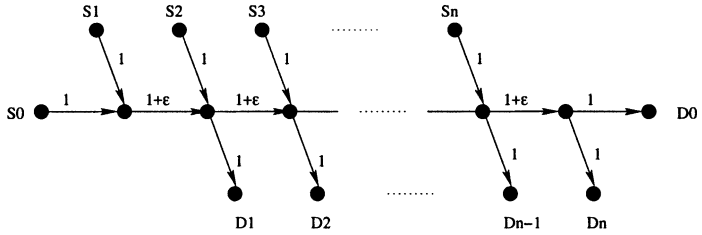

Fig. 1. The parking-lot topology PL.

over a period of one hour, flows with a total bandwidth of at least 100 Mbps must be accepted. In Section 9, we describe mechanisms to implement such policy constraints into our framework.

Traffic profile. Our algorithm uses information about 'expected' flows between some ingress–egress node pairs. We explain the exact from 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 the 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 either be measurement based or inferred from SLAs.

Routing information. 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 server model, although it can also be implemented in a distributed fashion.

3. Review of existing algorithms

The most commonly used algorithm for routing LSPs is the *shortest-path routing*, in which the least number of links between ingress and egress nodes is chosen. Each of these links has been assigned a cost or weight, to determine the link's preference. 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 underutilization (see examples in Section 5). Our new 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 the number of requests that might be rejected because of inappropriate route selection can be significantly reduced. To reduce the creation of bottlenecks, network operators often manually tweak the weights associated with individual links heavily until the worst bottlenecks have been resolved. Instead of full-fledged shortest path algorithm that has to deal with weights, 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.

Guérin et al. [11] propose a variant of the shortest-path algorithm, called Widest–Shortest Path (WSP), in which they choose a feasible shortest 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 shortest-path routing, it remains prone to myopic behavior: it does not take into account the influences that selecting a path between a given ingress–egress pair has on a potentially large number of other pairs. WSP has no notion of active ingress–egress pairs or traffic characteristics; all it cares for is the amount of bandwidth currently available on the individual links and the requirements of the flow that is currently being routed. As it does neither know nor care which other ingress–egress pairs will be impacted and how much, WSP still can create bottlenecks as outlined below. More significantly, neither the shortest-path nor WSP 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. Even when the path choice would perfectly minimize the impact on other potential flows, to give other flows a chance at all, it may sometimes be better to reject one flow in order to admit a large number of other flows. The example in Fig. 1 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 MIRA of Kodialam and Lakshman [2]. MIRA is a rather more sophisticated algorithm than either shortest path or WSP, and it takes critical advantage of ingress–egress pairs. The basic observation in Ref. [2] is that routing a flow along a path can reduce the *maximum permissible flow* between some other ingress–egress pairs. Kodialam and Lakshman call this phenomenon 'interference'. Their thesis is that if paths that reduce a large amount of possible maximum 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.

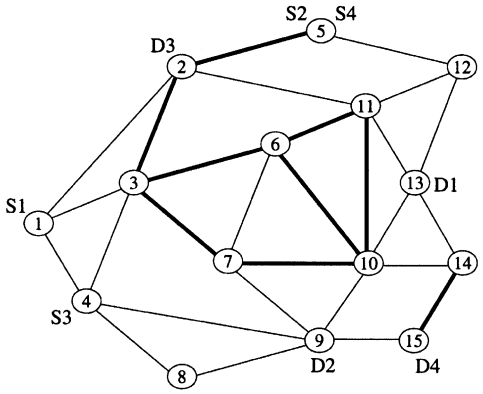

Fig. 2. An example network, showing ingress–egress nodes.

Another approach to minimum interference routing is presented in Ref. [12]. It defines interference based on the notion of 'critical paths', which are computed using WSP. It achieves the same performance as MIRA at a considerably reduced computational complexity.

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 incapable of estimating the bottleneck created on links that are critical for *clusters* of nodes (see examples in Section 5). Second, MIRA considers only the fact that using a given link would reduce the maximum bandwidth between a pair, without regard to the expected bandwidth between that pair. Thus, MIRA might choose a wastefully long path, although the more direct path retains sufficient residual bandwidth to accommodate the demands of other flows. Third, MIRA has no notion of admission control, which is necessary to provide fairness and adhere to SLAs. Finally, MIRA is computationally very expensive. While shortest-path, WSP, and our new algorithm all perform a single shortest-path computation to route a request, MIRA performs a maximum flow computation per ingress–egress pair, each of which can be several orders of magnitude more expensive than the shortest-path calculation.

Matta et al. [13] also use a profile for making routing decisions. For each source-destination pair, their profile contains a Probability Density Function (PDF) describing the likelihood of individual *request sizes*. This PDF is then used to reduce the link wastage among a set of alternate routes. Load profiling is most effective when the request sizes are large compared to the link bandwidth, and is orthogonal to profile-based routing.

4. 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 LSPs can be set up. We assume that the ingress–egress pairs are known, and that this information is quasi-static, i.e. that it changes very infrequently. An example is shown in Fig. 2, which is borrowed from Kar–Kodialam–Lakshman [14].[1] We call this network the KL1 graph, and it is one of the several networks we used for our simulations.

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 a residual capacity of at least b_i or reject it. 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 creates a significant bottleneck for future requests (based on its knowledge of the ingress–egress pairs and their traffic profile). We assume that all LSP setup 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 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 rough indication of the amount of traffic that can be expected between a pair; however the LSP setup request sequence 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 far as our routing algorithm is concerned, the request sequence is completely online.

[1] This figure actually reflects the graph used for the MIRA measurements in Ref. [14], although, according to one of the authors, the figure that was printed in their paper does not correctly reflect all high-bandwidth links.

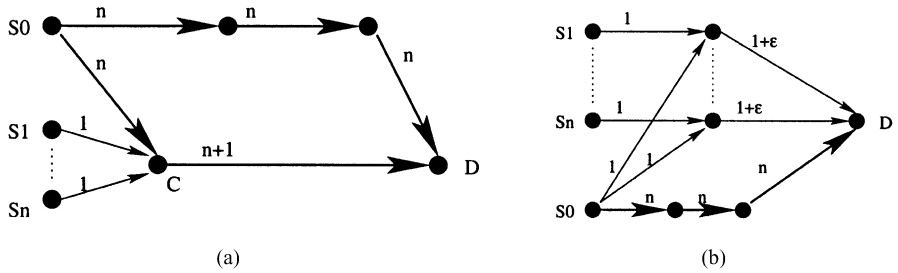

Fig. 3. Example topologies. (a) The concentrator topology CN. (b) The distributor topology DS.

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.

5. 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 such as 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 only a minimum of information about the network and traffic, which we believe can be easily obtained. Our algorithm, although as simple and computationally efficient as the shortest-path algorithm, 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 appear by necessity artificial, but their general form is not at all unusual. In fact, real networks are quite likely to contain subgraphs that resemble the *concentrator* or the *distributor* example. The parking-lot topology is common as well, but also depends on the selection of ingress–egress pairs. As 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. Fig. 1 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.

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 network—no other LSP request can be satisfied. However, as neither the shortest path nor WSP will reject flow requests if there is a feasible path (independent of the resulting impact) they will accept (S_0, D_0), resulting in a total network utilization of 1. An optimal algorithm will reject (S_0, D_0), and 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—as these links are not in the min cut for any (S_i, D_i) pair, and are not considered critical. Even if MIRA were modified to support admission control, it would still accept the first flow request and end up rejecting all other requests.

Although the links are drawn as directed, path selection and blocking behavior would remain the same for bi-directional 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. Because of asymmetric links or loads, the remaining capacity in the opposite direction could become too small to be useful.

Concentrator. Fig. 3(a) shows a network, which we call a *concentrator graph*—one node C acts 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 egress node D. One high-bandwidth ingress S_0 is also connected to the concentrator, though a capacity-n link. S_0 is also connected to D via an alternative three-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, 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 two hops). This leaves residual capacity 1 along the link CD, and thus at most one of the remaining n requests can be satisfied.

This example also illustrates the shortcoming of MIRA—the fat link CD is not in the minimum cut for any *individual* ingress–egress pair. Thus saturating it does not seem harmful to MIRA, and accordingly MIRA 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 sometimes may be a good idea not to use the fat pipe, our next example shows that the converse is also true.

In the example shown in Fig. 3(b), 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 is rejected. The routes selected by MIRA are also the same, as 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 (three hops), leaving the top two-hop paths for S_i to D requests.

The preceding examples are intended to illustrate how a bad path selection for one flow can create significant bottlenecks for future flows. An online routing algorithm that has no 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 [2], and propose a new algorithm as well as a general framework, in which we exploit information about the ingress–egress nodes as well as 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.[2]

6. 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 setup requests, it is NP-Complete to determine the maximum number of requests that can be simultaneously routed. Thus, the difficulty lies not necessarily in the online nature of the problem, but rather 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 multi-commodity flow problem is then used to *preallocate* 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 initially assigned zero), that flow request is rejected—even though there might be sufficient capacity in the network to route it. Let us begin with some preliminaries about multi-commodity flows. Interested readers can find a comprehensive treatment of network flows in Ref. [15].

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 there is a flow from v to w.

Capacity Constraint. $f(v,w)\leq \mathrm{cap}(v,w)$. If (v,w) is not an edge of G, then we assume that $\mathrm{cap}(v,w)=0$.

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 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 [16]. The two-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 $\mathrm{cap}(e)$ for each edge $e\in E$, and bandwidth requirements b_1 and b_2. Then, it is NP-Complete to decide whether there are flow functions f_1, f_2 such that (1) for each link $e\in E, f_1(e)+f_2(e)\leq \mathrm{cap}(e)$, (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.

7. Profile-based routing

Examining the problem more closely, we find that the intractability of the LSP setup 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

[2] Although MIRA later received a sibling, L-MIRA [14], which adds a fuzz factor when determining the bottleneck links, this is easily defeated either artificially by appropriately choosing ε or naturally through the inherent dynamic nature of the allocation process. In the latter case, adding flows to the 'better' link will soon shrink its available bandwidth, getting it beyond the fuzz threshold.

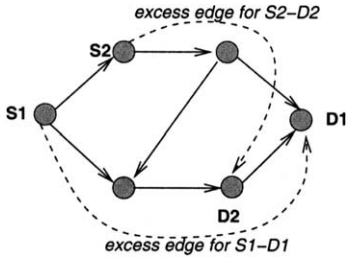

Fig. 4. The excess edges added to make the multi-commodity flow always feasible.

programming. Unfortunately, in the LSP problem, we do not want flows to be split, and we do want to enforce some kind of fairness to admit as many flows 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 request sizes are typically much smaller than the link capacities—for instance; the link capacities might range from OC-12 to OC-192, while a typical request might be just a few megabits per second. Second, we use the multi-commodity flow in the preprocessing phase, where 'commodities' correspond to highly aggregated traffic profiles, rather than to individual LSP requests. So, when a commodity is split, it does not mean that a flow is split; it merely 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 rejects it.

Our algorithm has two phases: a preprocessing phase, in which we solve a multi-commodity flow problem to pre-allocate link capacities for various traffic classes, and an online routing phase, in which LSP request is routed online using a shortest-path-like algorithm. Let us first describe the preprocessing phase.

7.1. Multi-commodity flow preprocessing

The input to the preprocessing phase is the network $G = (V, E)$, with capacity $\text{cap}(e)$ 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.

However, as noted earlier, satisfying all bandwidth requirements may not be possible. We therefore put additional edges in the network, called *excess edges*, so that the problem always has 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, shown as dashed lines in Fig. 4. Thus, $\text{cost}(e) = 1$ if $e \in E$, and $\text{cost}(e) = \text{cap}(e) = \infty$ 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.

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

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

subject to the following constraints:

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

The output of the multi-commodity flow computation is the values for the variables $x_i(e)$. We use these values to set a *pre-allocation* of the capacity of the edges e for various flows. In other words, the $x_i(e)$ part of e's capacity will be used by the online algorithm to route flows belonging to traffic class (and thus ingress–egress pair) 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.

7.2. 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 that 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 processes an online sequence of LSP setup 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 s 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.

To allow for significant deviations from the profile, the network operators may choose to hide some percentage of the link bandwidths from the multi-commodity flow

computation and make it available only at the time of flow routing on a first-come first-serve basis. The *classless residue* available this way on link e is referred to as $R(e)$.[3]

Algorithm 1. *Input*. The input graph $G = (V, E)$. For each edge e, we maintain residual capacity $r_j(e)$ for each commodity (traffic class, i.e. ingress–egress pair) $j = 1, 2, \ldots, k$. In addition, each edge maintains a classless residue, $R(e)$, which can be used by all commodities, once their capacities are exhausted. For standard profile-based routing, all $R(e)$ can be considered zero. 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 to d, such that for each edge e along this path there had been $r_j(e) \geq b$ (during the algorithm $r_j(e)$—and possibly $R(e)$—are updated to contain the new residual bandwidths).
Operation:

(1) Delete from G all edges e for which $r_j(e) + R(e) < b$. (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 path P, decrease the residual capacity $r_j(e)$ by b. If the resulting $r_j(e)$ becomes negative, bandwidth is taken from $R(e)$ as required.
(4) Route LSP (s, d, b) along path P.

7.3. 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 (as many as the number of ingress–egress pairs) maxflow computations. Each of these maxflow computation itself takes $O(N^2\sqrt{M})$ time. Additionally, MIRA needs to enumerate the links belonging to minimum cuts, which is of complexity $O(M^2)$. Thus, the total complexity of MIRA is $O(N^2\sqrt{M} + M^2)$. During the path-selection phase, our algorithm has the same run-time complexity as the currently used shortest-path algorithm. Our algorithm is faster than the WSP routing algorithm, because that algorithm must execute a Dijkstra-style shortest-path computation [17].

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 then uses to run the online path-selection phase.

7.4. Network dynamics

So far, we considered the network topology and profile to be static. In this section, we will discuss the results of link failures/additions or profile changes. As the problem becomes trivial when rerouting of existing flows is possible, we will only discuss the cases when rerouting is not an option.

When a link is removed, the following steps can be taken to get the network back into operation. It is assumed that the flows that have been severed by the link failure should be restored on the new topology.

(1) Remove the link from the graph.
(2) Remove all flows that crossed the link and remember them.
(3) Create a graph of a *residual network*, in which the capacity of each link is described by the total residual capacity $(R(e) + \sum r_j(e))$ of that link in the real network.
(4) Create a *residual profile*, which is the base profile reduced by the capacities of the currently routed flows.
(5) Run the multi-commodity flow computation with the residual network and profile (same complexity as initial computation, Section 7.3).
(6) Try to reroute the flows that crossed the removed link.

When a new link is added, the multi-commodity flow computation is also run using residual networks and profiles similarly created. A reduction or increase of the capacity of a link is handled as a straightforward modification of the link removal or addition process, respectively.

7.5. Dynamic profiles

When a profile is changed at run time and rerouting of existing flows is not an option, the changes may be activated as described in Section 7.4, using residual networks and profiles.

When no a priori profile information is available, but it can be assumed that current demand is an indication of future demand, *self-profiling* can be used: after routing the first few requests without a profile, using a standard routing algorithm, a profile is extrapolated from the requests seen so far. This profile is then used to route future requests, again using residual networks and profiles. Self-profiling is also useful if the actual network traffic is expected to deviate significantly from the recorded profile.

[3] To avoid unnecessary fragmentation when a classless residue is desired for additional flexibility, $R(e)$ should either be taken away before running the multi-commodity flow algorithm or taken away as a 'tax' from *all* paths.

Table 1
Worst-case performance improvement

Graph name	Total requests	Requests routed by					Factor of improvement
		SP	WSP	SWP	MIRA	PBR	
PL	$1+n$	1	1	1	1	n	n
CN	$2n$	n	n	n	n	$2n$	2
DS	$2n$	n	n	$2n-1$	n	$2n$	2 (except SWP)

8. 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 (PBR) algorithm should (and does) perform better than others. One highly attractive feature of our algorithm is that computationally it is online as efficient as the shortest-path or WSP routing, and substantially faster than MIRA.

In this section, we compare the performance of PBR with the well-known path computation algorithms Shortest-Path (SP), Shortest–Widest-Path (SWP), and WSP. In addition, we also include performance figures of a MIRA based on flow maximization. This algorithm uses an approximation for maximizing the weighted sum of flows among ingress–egress pairs, which is described in more detail in Ref. [18]. We denote this algorithm WSUM-MAX Approximation (or WSUM-MAX for short), which, similar to MIRA, tries to approximate the WSUM-MAX problem [14]. This approximation should meet or exceed a performance comparable to MIRA, especially as it avoids some of the pitfalls described earlier for MIRA.[4]

We used four network topologies to measure the performance of our PBR algorithm. The first three topologies are the ones we used in Section 5. The fourth topology, referred to as KL1, is the one used by Kodialam and Lakshman [2] in their experiments. It will be used with different ingress–egress pairs and link weights, as explained in the simulation section.

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 Fig. 3(a) and capacity $1 + \in$ in Fig. 3(b) become links of capacity 800, while those with capacity n or $n + 1$ become links of capacity 4800. In the KL1 network, all light edges have capacity 1200, while dark ones have capacity 4800 (meant to model OC-12 and OC-48 links, respectively), matching the values used in Ref. [14]. Finally, we used a publicly available implementation of the minimum cost multi-commodity flow algorithm, the PPRN package, for our preprocessing phase (available at http://www-eio.upc.es/~jcastro/pprn.html).

The experiments have been conducted in a route server environment as described in Ref. [19]. In each experiment, we generated a random sequence of individual requests, and measured the performance of the various path computation algorithms. All of the experiments are conducted using 'static' requests, which means that the bandwidth allocated for a request is never freed again. The results shown are average values of 20 test runs for each experiment, except experiment 5, which shows average values of 100 test runs.

8.1. Worst-case analysis

In this first comparison, we revisit the three topologies (PL, CN, and DS) from Section 5 and infer the worst-case behavior of shortest-path, WSP, MIRA, and PBR on these topologies without simulation. Table 1 documents these results.

In the PL topology, if the first request is between nodes S_0 an D_0, then all greedy algorithms (shortest path, WSP, SWP, and MIRA) accept it, which blocks all future requests from being routed. As the number of ingress–egress pairs n increases, the percentage of network utilized by the greedy algorithms goes to zero. In order to avoid such a near-zero utilization of the network, our new algorithm (PBR) rejects the first request, and is then able to satisfy all remaining n requests between S_i and D_i.

Even if MIRA were extended by a cost threshold (total weight $< W$) for admission control, it would not work for this topology because 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 CN topology, 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 DS topology, except that shortest–widest path routing (SWP)

[4] We had to revert to using this algorithm as an approximation of MIRA, as the MIRA specification is not sufficiently complete to create an independent implementation suitable for direct performance comparison.

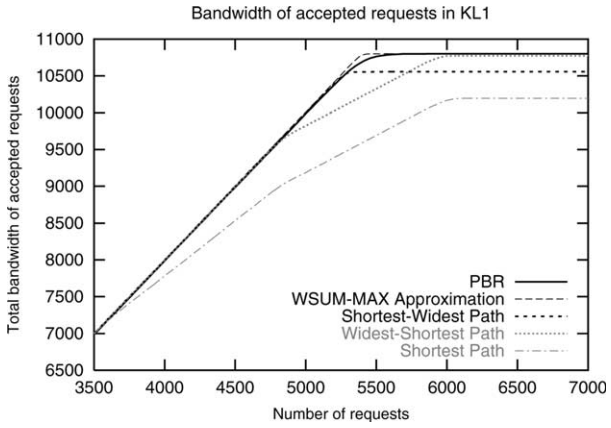

Fig. 5. Throughput of accepted requests using demands of 1–3 in KL1.

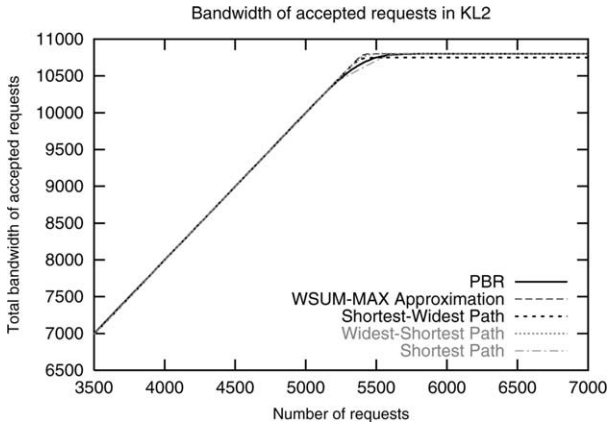

Fig. 7. Throughput of accepted requests using demands of 1–3 in KL2.

will choose the correct path for the first $n - 1$ flows of unit size from S_0 to D.

8.2. Experiment 1: uniform link costs

Experiment 1 has been conducted on KL1 network shown in Fig. 2. The network has been loaded with 7000 requests, having bandwidth demands uniformly distributed in the range of 1–3 units (only integer values are used). Also, the requests have been uniformly distributed among the ingress-egress pairs. Consequently, WSUM-MAX has been configured to use equal weights for the ingress–egress pairs and the traffic profile for PBR has been computed using the multi-commodity flow algorithm with 'infinite' supply for each ingress node. This computation resulted in flows of 2700 units of bandwidth between each ingress–egress pair. The link 'costs' are set to 1, such that the shortest-path algorithm is reduced to a minimum-hop algorithm.

The bandwidth of accepted requests of experiment 1 is shown in Fig. 5. For each of the algorithms, the bandwidth increases with the number of requests until a saturation point is reached at which no more requests can be accommodated, and the network is saturated. The first performance measure we use is the bandwidth of successfully routed requests after the saturation point has been reached. The SP shows the weakest performance, with a saturation point around 10,200 bandwidth units. The best performance is shown by PBR and WSUM-MAX with 10,800 units, followed very closely by WSP with 10,770 units, and SWP with 10,550 units. Note that because of the random process that is used to generate the requests, PBR reaches its saturation point slightly later than WSUM-MAX, after having routed exactly 2700 units of bandwidth between each ingress–egress pair. WSUM-MAX, on the other hand, is more flexible in the sense that it allocates the bandwidth on a first-come first-serve basis. Similar to the cases discussed for MIRA, this makes WSUM-MAX susceptible to creating bottlenecks.

A second performance measure looks at the number of blocked requests. Fig. 6 shows the number of blocked versus total requests. While SP starts to block requests after 3500 requests, PBR and WSUM-MAX start to block requests only after 5125 and 5340 requests, respectively.

In this and all other experiments with small request sizes, 20 test runs were carried out, and the results shown are the mean values obtained. The experiments have a 99.9% confidence interval of not exceeding 1% of the mean values. (For experiment 5, which uses much larger request sizes, the resulting spread is presented together with the other data.)

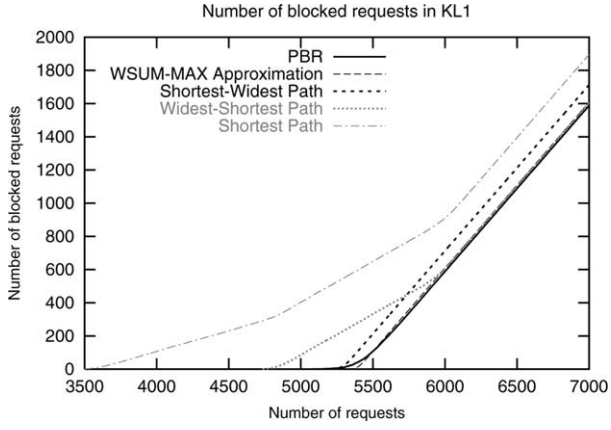

Fig. 6. Blocked requests using demands of 1–3 in KL1.

8.3. Experiment 2: link costs inversely proportional to link capacity

In the next experiment, we study the effect of static link costs on the performance. In network KL1, all links have a cost of 1. We obtain network KL2 by assigning different costs to the links. Following a common practice, we assign link costs inversely proportional to the link capacities. Links with capacity 1200 are assigned a cost of 4, and links of capacity 4800 are assigned a cost of 1. This results in network KL2. The remaining parameters are unchanged. Please note that path selection done by PBR (as well as WSUM-MAX and MIRA) is independent of the link costs,

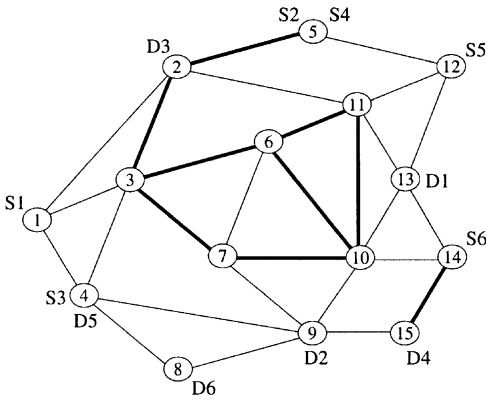

Fig. 8. Example network KL2 + with additional ingress–egress pairs.

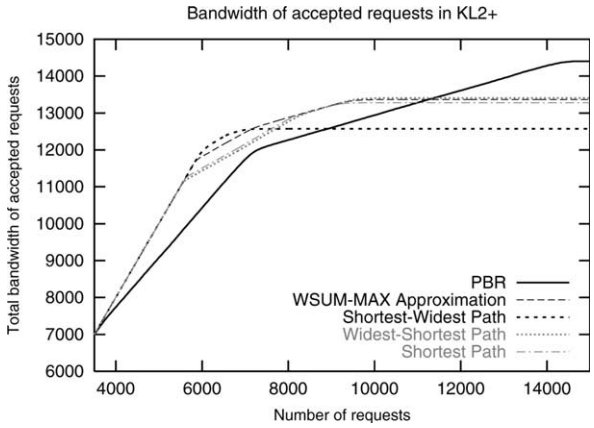

Fig. 9. Throughput of accepted requests using demands of 1–3 in KL2 + and a PBR profile that maximizes the total throughput.

and thus their performance remains unchanged to the previous experiment.

Fig. 7 shows the bandwidth of accepted requests in network KL2. Thanks to setting the link costs favorably for the link-cost-based algorithms (SP, SWP, WSP) and the generally low interference in this graph, even the less sophisticated algorithms perform very well. Except for SWP, all algorithms eventually achieve the theoretical maximum of 10,800 units. However, SP and also PBR achieve this maximum slightly later than the other algorithms. PBR's slight delay is due to its use of admission control, which would not really be necessary in this graph which is well tuned to the offered load, but is very helpful when the routing behavior should be predictable or is subject to side constraints such as SLAs.

8.4. Experiment 3: additional ingress–egress pairs

In a third experiment, we increase the number of ingress–egress pairs and therefore interference. Fig. 8 shows the example network after two additional ingress–egress pairs have been added to KL2. We refer to this network as KL2 + . Note that, unlike the corresponding MIRA topology from Ref. [14], the link costs are again inversely proportional to the link capacities, to reflect common practice and aid the cost-based algorithms (SP, SWP, WSP).

For the third experiment, 15,000 requests have been issued for network KL2 + . As in the previous experiments, the requests are uniformly distributed among the six ingress–egress pairs and the bandwidth demand is an integer value uniformly distributed in the range from 1 to 3. WSUM-MAX uses equal weights for the ingress–egress pairs. For PBR, the profile has been computed using the multi-commodity flow algorithm, resulting in a flow distribution as shown Table 2.

Fig. 9 shows the bandwidth of accepted requests in KL2 + . The largest total throughput is achieved by PBR. At the same time, PBR starts blocking requests for pairs S1 → D1 and S6 → D6 the earliest. The reason for this behavior is the fact that requests for all ingress–egress pairs are uniformly distributed, whereas the calculated maximum profile differs up to a factor of 4. Nevertheless, PBR achieves a total throughput of 14,400 units, whereas, WSP, WSUM-MAX and SP only achieve 13,400, 13,360, and 13,380, respectively.

In order to have PBR achieve a uniform distribution among the ingress–egress pairs, we compute a PBR profile by solving the multi-commodity flow problem with a supply of 2400 units for each of the six pairs. In the resulting profile, each pair supports 2400 units of bandwidth except pair S2 → D2, which is only able to carry 1200 units of bandwidth. The resulting aggregated throughput is shown in Fig. 10.

8.5. Experiment 4: weighted request distribution

The fourth experiment has been carried out on network KL2 + . Unlike in the previous experiments, the requests were not uniformly distributed, but followed the distribution given in Table 3. The relative weights used for WSUM-MAX, which match the profile shown in Table 2, are shown in the second row of Table 3. PBR has been used with the profile given in Table 2.

The results of this experiment are shown in Fig. 11. As expected, PBR achieves the maximum theoretical throughput of 14,400 units of bandwidth. WSUM-MAX follows

Table 2
Profile generated for KL2 +

	S1 → D1	S2 → D2	S3 → D3	S4 → D4	S5 → D5	S6 → D6
Flow value	1200	2400	4800	2400	2400	1200

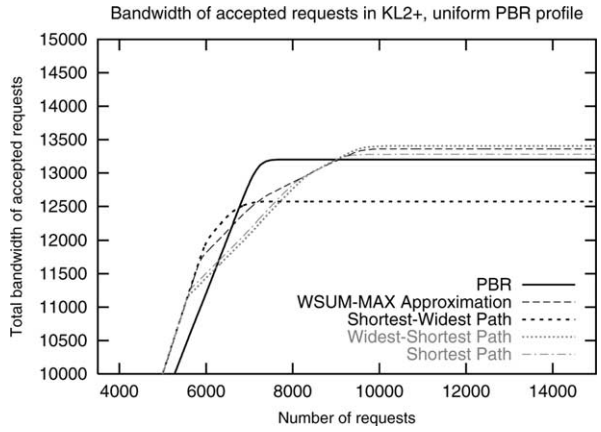

Fig. 10. Throughput of accepted requests using demands of 1–3 in KL2 + and a uniform PBR profile (bandwidth artificially limited to 13,200 units).

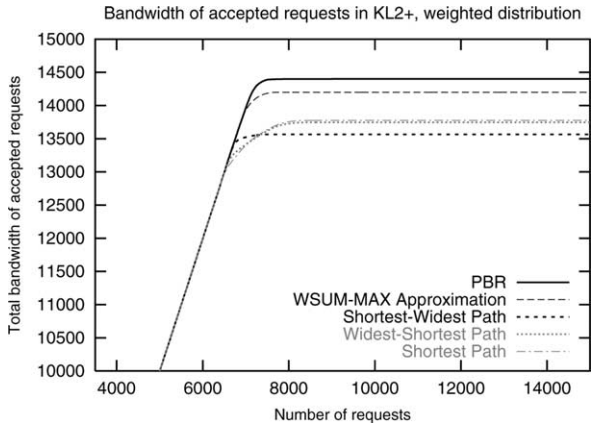

Fig. 11. Throughput of accepted requests using a weighted distribution of demands of 1–3 in KL2 + .

closely behind with a total throughput of 14,200 units of bandwidth. WSP and SP both achieve 13,750 units of bandwidth, whereas SWP only achieves slightly more than 13,500.

8.6. Experiment 5: large demands

The last experiment is a repetition of experiment 4 but using different demand sizes. The demand sizes have been scaled by a factor of 100 and thus cover 100–300 units of bandwidth (again only integer bandwidths are chosen from the range). The profile used for PBR and the relative weights used for WSUM-MAX remain unchanged. Fig. 12 shows the average values of 100 test runs. While the ranking of the algorithms remains the same, the performance difference increase. This is due to the fact that an increased request size also increases a flow's blocking effect through fragmentation of the remaining bandwidth.

For experiment 5, 100 test runs have been carried out, using a random process for selecting the ingress–egress pairs and the demand sizes. Thus, each test run used its own pattern of requests and demand sizes. Fig. 13 shows the variation of the total routed bandwidth and the number of blocked requests for the different algorithms after 60 and 100 requests have been issued, respectively. After 60 requests, all of the algorithms show the same behavior. At this point, some of the requests are being blocked, but the network is already saturated to a large degree. In this situation, PBR clearly exhibits the smallest variation regarding the amount of accepted bandwidth. This means that PBR's performance is largely independent of the request pattern used. This is in contrast to the other algorithms, in which the variation in total routed bandwidth is more than twice that of PBR. Additionally, the number of blocked requests is considerably lower in PBR than in any of the other algorithms.

8.7. Analysis of the simulation results

All algorithms (with the exception of WSP and sometimes SP) generally perform very well when the per-link costs have been carefully assigned and interference is low. While careful tuning of link loads is currently the norm among ISPs, this cumbersome process which has to be repeated whenever a link fails or is added can be eliminated by PBR.

For high interference and asymmetric load, PBR is able to sustain a (potentially much) higher load. We expect these networks to be the norm, especially for large ISPs. Doing the access control necessary to achieve such high loads, PBR may block a few requests violating the profile to ensure that it can admit more bandwidth from conforming ingress–egress pairs later.

We do not expect this blocking to have much impact in real-world networks. First, we expect profiles to be a meaningful prediction of the short-term future; those customers with large bandwidth demands will typically show some long-term stability. Second, the multi-commodity flow preprocessing is able to preallocate at least as much bandwidth as the other algorithms (this is true for all simulations, except where the multi-commodity flow algorithm has been forced not to use all bandwidth). The actual flows to be admitted thus gain significant headroom.

Table 3
Distribution of requests for experiment 4

	S1 → D1	S2 → D2	S3 → D3	S4 → D4	S5 → D5	S6 → D6
Distribution	1/12	1/6	1/3	1/6	1/6	1/12
Weights	0.5	1	2	1	1	0.5

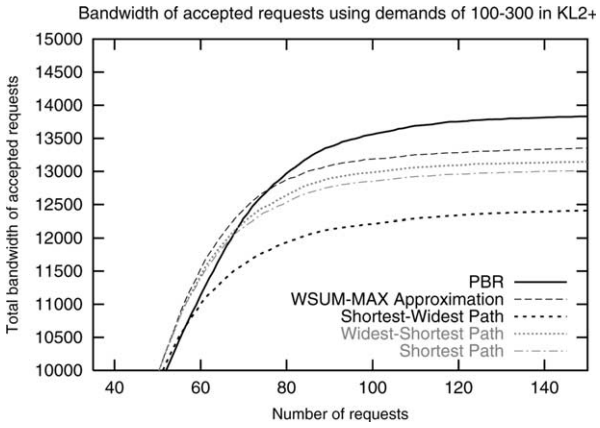

Fig. 12. Throughput of accepted requests using a weighted distribution of demands of 100–300 in KL2 + .

In case a network operator should have significantly unsteady load and thus limiting the usefulness of the profile, it can take proactive measures. For example, by allocating some percentage of the link capacity to the classless residue $R(e)$ as described in Section 7.2, an ISP gains some bandwidth that might be flexibly allocated. Alternatively, self-profiling as described in Section 7.5 might be applied to eliminate the need for manual configuration.

8.8. Route computation time

In addition to the theoretical complexity analysis, we also conducted measurements on our implementations of the algorithms. In particular, we measured the average execution time of several thousand path computations on network KL2 + . Although KL2 + is a rather small network, it is still sufficient to illustrate the differences in execution time; we expect the differences to increase substantially with bigger networks. All results reported below were run on a 1500 MHz Pentium 4. As the routing simulator [19] was written with readability and extensibility in mind, an optimized implementation is expected to significantly improve these results.

As could be expected SP, WSP, and PBR need about the same time to compute a single path. Average values for these algorithms are in the range of 110–125 μs, corresponding to 8000–9000 route requests per second. Our implementation of SWP requires two iterations of Dijkstra's algorithm [20], which results in an average execution time of 150 μs. Even though SWP executes Dijkstra's algorithm twice, the second execution is done on a much-reduced network that often contains just a single path, increasing the execution time by far less than a factor of two. The WSUM-MAX approximation computes several max-flows using Goldberg and Tarjan's preflow-push algorithm [21]. The average execution time is 3000 μs, which is more than a factor of 20 slower than the other algorithms. The initial computation of PBR's profile requires 30,000 μs and each individual path computation requires another 120 μs. Thus, the initial effort is amortized after about 10 path computations, after which PBR performs better than the WSUM-MAX approximation. As MIRA needs to determine the min-cut after calculating the max-flow, we expect it to be even slower than WSUM-MAX.

9. Concluding remarks and extensions

We presented a new PBR algorithm for dynamic routing of bandwidth-guaranteed paths. The online routing phase of

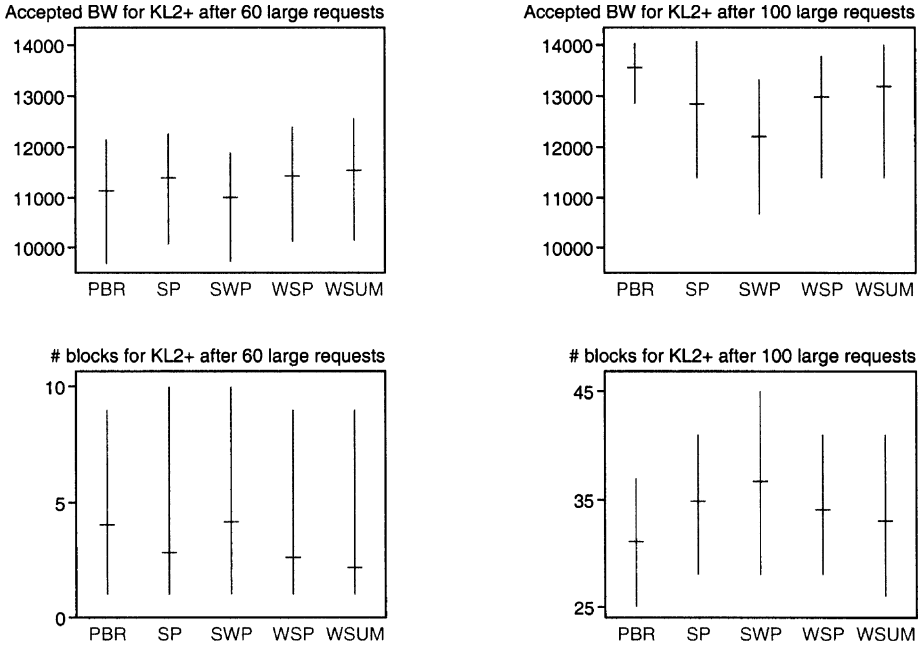

Fig. 13. Performance variation (minimum, average, maximum) using a weighted distribution of demands of 100–300 in KL2 + .

the algorithm is as simple and computationally as efficient as the commonly used minimum-hop routing or the WSP routing, and it is substantially faster than the recently proposed minimum interference routing algorithm [2]. Our algorithm improves network utilization, and accepts more flows than these other algorithms do, 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. CN and DS topologies), as opposed to single pairs identified by MIRA. We were unable to perform a direct comparison with MIRA itself, as not enough information was available. We consider WSUM-MAX to be a good approximation of MIRA, even though it may sometimes return better results.

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 three such extensions.

Minimum Service Level. Suppose a service provider wants to ensure that a bursty traffic class receives at least a guaranteed maximum 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.

Combination with Load Profiling. If load profiles [13] are available, they can be used to route large requests in order to minimize call blocking due to insufficient contiguous bandwidth. We expect it to achieve only minimal improvement under small requests or heavy overload. With large requests, load profiles should result in significantly postponing the first block seen on a source-destination pair.

9.1. Future impact

Our framework fits nicely with mechanisms that are currently being deployed and will be deployed in the future: Connection-oriented routing, including label-switching mechanisms such as MPLS, will see a major boost in the near future, due to the proliferation of VPNs and strict QoS demands associated with upcoming interactive media technologies, ranging from voice over IP and video conferencing to large-scale telepresence and interactive entertainment.

We further consider PBR to be highly scalable and adaptive, as the amount of work incurred per flow is linear in the size of the network, unlike other approaches, which grow quadratically to cubically with size. Even when network topologies or demand should change significantly, PBR adapts very well to new situations.

Optical advances, including fiber to the curb, will help create even larger networks with more resources. This will augment label switching with λ-switching, and increase the demand for routing mechanisms that are efficient in terms of both link bandwidth and route server CPU usage.

Finally, the increasing mobility of networked devices, personal networks, and other mobile networks (e.g. cars, trains, boats, or planes), fast handover and thus rerouting of flows is key. This is greatly simplified by the linear time behavior of the online portion of PBR.

Acknowledgements

Initial work performed at Washington University in St Louis, partially supported by NSF grant ANI 9813723. An earlier version of this paper was presented at QoFIS 2001 [1].

References

[1] S. Suri, M. Waldvogel, P.R. Warkhede, Profile-based routing: a new framework for MPLS traffic engineering, in: F. Boavida (Ed.), Quality of Future Internet Services, Lecture Notes in Computer Science, vol. 2156, Springer, Berlin, 2001, pp. 138–157.

[2] M. Kodialam, T.V. Lakshman, Minimum Interference Routing with Applications to MPLS Traffic Engineering, Proceedings of IEEE Infocom, Tel-Aviv, Israel, 2000.

[3] E.C. Rosen, A. Viswanathan, R. Callon, Multiprotocol label switching architecture, Internet Engineering Task Force RFC 3031 January (2001).

[4] B. Gleeson, A. Lin, J. Heinane, G. Armitage, A. Malis, A framework for IP based virtual private networks, Internet Engineering Task Force, RFC 2764 (2000).

[5] T.V. Lakshman, D. Stiliadis, High Speed Policy-based Packet Forwarding Using Efficient Multi-Dimensional Range Matching, Proceedings of ACM SIGCOMM '98, 1998, pp. 203–214.

[6] V. Srinivasan, G. Varghese, S. Suri, M. Waldvogel, Fast and Scalable Layer Four Switching, Proceedings of ACM SIGCOMM '98, 1998, pp. 191–202.

[7] V. Srinivasan, S. Suri, G. Varghese, Packet Classification Using Tuple

Space Search, Proceedings of ACM SIGCOMM '99, Cambridge, MA, USA, 1999, pp. 135–146.
[8] R. Braden, D. Clark, S. Shenker, RSVP: a new resource reservation protocol, IEEE Network 7 (9) (1993) 8–18.
[9] L. Andersson, P. Doolan, N. Feldman, A. Fredette, B. Thomas, LDP specification, Internet Engineering Task Force, RFC 3036 (2001).
[10] R. Guérin, H. Ahmadi, M. Naghshineh, Equivalent bandwidth and its application to bandwidth allocation in high-speed networks, IEEE Journal on Selected Areas in Communications 9 (7) (1991) 968–981.
[11] R. Guérin, A. Orda, D. Williams, QoS Routing Mechanisms and OSPF Extensions, Proceedings of Second Global Internet Miniconference, 1997.
[12] I. Iliadis, D. Bauer, A New Class of On-Line Minimal Interference Routing Algorithms, IBM Research Report 3379, IBM Zurich Research Laboratory, Rüschlikon, October, 2001.
[13] I. Matta, A. Bestavros, M. Krunz, Load profiling based routing for guaranteed bandwidth flows, European Transactions on Telecommunications 10 (2) (1999).
[14] K. Kar, M. Kodialam, T. Lakshman, Minimum interference routing of bandwidth guaranteed tunnels with MPLS traffic engineering applications, IEEE Journal of Selected Areas in Communications 18 (12) (2000) 2566–2579.
[15] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice-Hall, Upper Saddle River, NJ, USA, 1993.
[16] M.R. Garey, D.S. Johnson, Computers and Intractability: a Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, 1979.
[17] E.W. Dijkstra, A note on two problems in connection with graphs, Numerische Mathematik 1 (1959) 269–271.
[18] D. Bauer, A new minimum-interference routing algorithm based on flow maximization, IEE Electronics Letters 38 (8) (2002) 364–365.
[19] S. Cech, A Route Server for Constraint-Based Routing in MPLS, Master's Thesis, Universität Klagenfurt, Austria, January 2002.
[20] Q. Ma, P. Steenkiste, Onpath Selection for Traffic with Bandwidth Guarantees, Proceedings of IEEE International Conference on Networking Protocols (ICNP'97), 1997, pp. 191–202.
[21] A.V. Goldberg, R.E. Tarjan, A New Approach to the Maximum Flow Problem, Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, 1986, pp. 136–146.