Load optimal MPLS routing with N + M labels

David Applegate
AT&T Labs–Research
Shannon Laboratory
180 Park Avenue
Florham Park, NJ 07932
E-mail: david@research.att.com

Mikkel Thorup
AT&T Labs–Research
Shannon Laboratory
180 Park Avenue
Florham Park, NJ 07932
E-mail: mthorup@research.att.com

***Abstract*— MPLS is becoming an important protocol for intra-domain routing. MPLS routers are offered by the major vendors and many ISPs are deploying MPLS in their IP backbones, as well as in ATM and Frame Relay networks. For this period of possible transition to MPLS, it is urgent to increase our understanding of the power and limitation of MPLS.**

An attraction to MPLS is the flexibility it offers in engineering the routing of traffic in a network, e.g., to support higher demands without overloading any links. Mitra and Ramakrishnan [GLOBECOM'99] showed that optimal routing solutions may be found for a diverse set of traffic engineering goals. However, for a network with N nodes (routers) and M edges (links), their MPLS implementation may use $\Omega(N \times M)$ different labels. This is prohibitive since the number of labels is the number of entries needed in the router tables.

We present an algorithm reducing the number of MPLS labels to $N + M$ without increasing any link load. Our explicit $N + M$ bound makes it easy to limit the table size requirement for a planed network, and the linearity allows for tables implemented in fast memory. For differentiated services with K traffic classes with different load constraints, our bound increases to $K(N+M)$. Our stack-depth is only one, justifying implementations of MPLS with limited stack-depth.

I. Introduction

Multi-Protocol Label Switching (MPLS) was proposed as a standard in [1], and it is now becoming an important protocol for intra-domain routing. MPLS routers are offered by major vendors and many ISPs are deploying MPLS in their IP backbone, as well as in ATM and Frame Relay networks (see [2] for listings of vendors and users). For this period of possible transition to MPLS, it is urgent to increase our understanding of the power and limitation of MPLS.

Our general interest here is the use of MPLS in traffic engineering [3], [4], [5], [6]. As stated in [6], "traffic engineering is the process of controlling how traffic flows through ones network so as to optimize resource utilization and network performance".

An advantage to MPLS over traditional shortest path protocols such as OSPF [7] or IS-IS [8] is that it offers considerably more flexibility in the control of traffic [4], [6], and this can be used to derive optimal routing solutions to a vararity of traffic engineering goals related to maximizing the through put while avoiding over-utilized links [5].

However, an unnoticed problem in previous MPLS routing solutions (see e.g. [6], [5]) is that they require routing tables with a quadratic number of entries, and this limits their scaling to large networks. Our contribution is to get down to a linear number of entries without increasing any link loads. Thus table space is the concern in this paper. Generally, this should allow us to provision networks with smaller, cheaper, and faster routers. Moreover, the ability to implement an individual routing solution in very little space, allows us prepare for many different traffic classes and error scenarios.

A. The off-line constraint based routing problem

As in [5], [6], our main focus is on the *off-line constraint based routing problem* defined as follows. A network topology is presented as a graph $G = (V, E)$ whose nodes are the routers and whose edges are the links in the network. Each link $e \in E$ has a capacity C_e. Among the nodes, we have some sources $S \subseteq V$ (ingress routers) and some destinations $T \subseteq V$ (egress routers). Also, we have a demand matrix D specifying the amount of traffic $D_{(s,t)}$ to be routed from each source $s \in S$ to each destination $t \in T$. Our target is to distribute the traffic subject to the constraint that the total load on any link does not exceed its capacity.

In the rest of this paper, S, T, N, and M denote the number of sources, destinations, nodes, and edges, respectively, that is, $S = |S|$, $T = |T|$, $N = |V|$, and $M = |E|$. We generally think of our networks as sparse with many nodes acting as sources and destinations. Hence, even though we distinguish between S, T, N, and M in our bounds below, we think of all of them as being of the same order of magnitude.

The off-line constraint based routing problem is equally interesting for other routing protocols and several studies [9], [10], [11], [12] have been performed for traditional shortest path protocols such as OSPF [7] or IS-IS [8].

1) Interpreting the parameters: We note that the link capacities in our problem may be set somewhat lower than the official link capacities, e.g., 60%, in order to ensure Quality-of-Service (QoS). This kind of under-subscription also helps protecting against bursts [6].

Below we consider two different interpretations of the demand matrix, each leading to different optimization criteria.

As in [5], the demand matrix may represent service level agreements (SLAs), each committing support for a certain amount of traffic along a virtual leased line from a source to a

destination. We then have a bandwidth-broker type optimization problem where SLAs between different source-destination pairs are priced differently. Subject to the capacity constraints in the network, our goal is to maximize our earnings in terms of sold SLAs.

As in [6], we can also have an estimated demand matrix based on concrete measurements of traffic. In [6] they use capabilities in MPLS to measure the demands, and suggest picking the demand from a source to a destination as the 95-percentile measured over a week, and combine this with under-subscribed links to protect against bursts. Other approaches for measuring demands are presented in [13], [14]. Finally, methods have been suggested for predicting a demand matrix based on demographic data [15]. We note that some notion of a demand matrix is always present, at least implicitly, in the provisioning and configuration of communications network: we have to take into account that more demand is expected between New York and Washington than between two random cities in Alaska. The better we understand the demand matrix, the better we can share the network resources among the demands. With an estimated demand matrix, we may consider a best-effort type optimization problem of minimizing the overall packet loss or latency in the network. We then have a cost associated with each link which is some increasing convex function of its load. The goal is to minimize the total costs over all links [10], [11].

After our technical presentation, in §IV, we shall discuss various limitations and applications of our approach in connection with multiple traffic classes, dynamic reoptimization, and error-scenarios.

B. Constraint based routing with MPLS

As in the vendor paper [5] from Lucent and the ISP paper [6] from Global Crossing, we are assuming that we want to use MPLS to run a whole network, solving the constraint based routing problem. Here we are trying to understand if such an approach can scale for larger networks.

We note that MPLS may also be used in conjunction with, say, OSPF, using MPLS tunnels to define virtual OSPF links whose traffic follows a specified path in the network [1]. In that context, our work is not likely to be relevant unless it is a very elaborate system of tunnels that is called for.

A very significant feature of MPLS is that we can freely distribute the traffic on different paths from a source to a destination [6]. As described in [5], this freedom implies that most variants of the off-line constraint based routing problem can be formulated as linear or convex programming problems. Such problems can be solved optimally in polynomial time [16], and moreover, we can solve large problems with off-the-shelf commercial software packages [17] and open-source software packages [18]. In fact, these kinds of problems are known as minimum-cost multicommodity flow problems for which even faster specialized algorithms have been developed (see e.g. [19]). However, when implementing the optimal multicommodity flow solution using the flow-decomposition technique from [5, §2.5] in MPLS, we may end up with routing tables with $\Theta(\mathrm{T} \times \mathrm{M})$ entries, which is prohibitive for large networks (c.f. §III-B for negative examples making [5] realize the lower-bound).

We note that in the abstract and title, for simplicity, we used N in place of S and T. Both S and T may be as big as N, but here we want to qualify that we benefit if they are smaller.

An alternative is to use a greedy heuristic like the one from [6]. This heuristic reduces the table size to $\Theta(\mathrm{S} \times \mathrm{T})$ entries, which is still a lot. However, as a heuristic, the technique from [6] is not guaranteed to find a solution, even if one exists. In fact, the heuristic from [6] may only route a fraction $1/(\mathrm{N}-1)$ of the possible traffic (c.f. §III-A for negative examples for [6]).

1) Our result: The result of this paper is that we can always implement an optimal solution to the constraint based routing problem using at most $\mathrm{T} + \mathrm{M}$ entries in the routers. Our explicit $\mathrm{T} + \mathrm{M}$ bound makes it easy to limit the table size requirement for a given network, and the linearity allows for tables implemented in fast memory.

What we present is a algorithm that takes a given routing solution S as input. The algorithm then implements in MPLS another routing solution S' such that for each link e in the network, the load of e in S' is no bigger than in S. Thus, if we have any measure of costs which is non-decreasing in the individual link loads, S' is as good as S, if not better. If S is optimal w.r.t. such a cost function, we say that S is *load optimal*, and then S' preserves this load optimality.

Thus, we may use the algorithm from [5] as a front-end that produces a load optimal routing solution S. Unfortunately, in implementing S, [5] may use MPLS tables as large as $\Theta(\mathrm{T} \times \mathrm{M})$. However, using our algorithm as a back-end, we reroute S to a solution S' that preserves load optimality, but which is implemented with MPLS tables of size at most $\mathrm{T} + \mathrm{M}$.

C. Inside MPLS

We will now go into some of the details of MPLS routing [1].

1) ATM style: In [20], [5], [6], they use MPLS to set up so-called label switched paths (LSPs) from sources to destinations. They may set up multiple LSPs from a source to a given destination, and the source can split the traffic over these LSPs. These LSPs are similar to the virtual circuits in traditional ATM.

Packets are forwarded along the LSPs using a label in the header. Each router/node has a table that for each label tells what outgoing port to use for packets with that label, or to stop if the router is a destination for the label. Moreover, the router may swap the label with another label before forwarding the packet.

The label swapping is used as follows. We configure one LSP P at the time. On each edge (u, v) in P, we pick a label x that has not before been used on an edge entering v. If the next edge in P is (v, w) with label y, the router table at v will have an entry telling that incoming packets with label x

should be forwarded along (v, w) using the label y. If labels entering each node are picked in order $1, 2, ...$, the maximal overall label will be the maximal number of LSPs entering a single node.

The MPLS implementations of optimal routing solutions from [5] may use $\Theta(\mathrm{T} \times \mathrm{M})$ LSPs, and it may have that many LSPs entering a single router, hence the claimed bound on the table size. Similarly, the heuristic routing solutions from [6] may lead to $\Theta(\mathrm{S} \times \mathrm{T})$ LSPs and have that many LSPs entering a single router.

We note that [21] proposes a technique for implementing optimal routing solutions with a combination of MPLS and OSPF, using LSPs to define virtual links for OSPF. This does work nicely for some simple examples in [21]. It is not quite clear how the routing solutions are computed, but assuming similar techniques to those in [5], [21] may still end up with $\Theta(\mathrm{T} \times \mathrm{M})$ LSPs in general.

In this paper, based on some standard results on multi-commodity flows [22], we point out that we can actually always find and implement optimal routing solutions using $\Theta(\mathrm{S} \times \mathrm{T} + \mathrm{M})$ LSPs. This is relevant for traditional ATM networks that do not support the to-trees below.

2) The power of to-trees: As pointed out in [1], the label based forwarding mechanism of MPLS can also be used to route along multi-point-to-point trees. Here, a multi-point-to-point tree is a rooted tree R, rooted in some destination t, and with all edges oriented toward t. For short, we call R a *tree to* t, or, if we do not wish to specify the destination, a *to-tree*.

To-trees can be used to route from multiple sources to a single destination. The simplest implementation in MPLS identifies each of these to-trees R with a label ℓ_R. Each router in R sends packets labeled ℓ_R to the parent in R, except the root which is the final destination.

Our main result is that we can always find and implement a load optimal routing solution using $\mathrm{T} + \mathrm{M}$ to-trees.

Above, we do not exploit routers' ability swap the labels. Similar to LSPs for paths, we can implement to-trees with so-called LSP trees [1]. Here we just call them *LSTs* since the P in LSP stands for path, which we replace with T for tree. In an LST T, we may have many edges entering a node, and then all these edges use the same label. Picking the labels entering each node minimally, the maximal overall label becomes the maximal number of LSTs with an edge entering a single node.

As for the LSPs, we may end up with many fewer labels than LSTs, but it doesn't improve our worst-case bound of $\mathrm{T} + \mathrm{M}$. However, we consider $\mathrm{T} + \mathrm{M}$ to be a sufficiently small number that we do not need to worry about any further improvements. The basic point is that when we have provisioned a network with routing tables of a certain size, we do not gain much from only using part of the tables. Our result says that tables of size $\mathrm{T} + \mathrm{M}$ suffice.

3) The label stack: Above, we have completely ignored that MPLS packets actually may carry a stack of labels that the routers can push and pop from. This feature can be used in connection with link failures where one pushes a label on the stack representing a detour around the failed link [23].

Alternatively, the stack may be used, as in [6], for hierarchical routing. This may, or may not, reduce the label space. Also, we note that hierarchical optimization may easily lead to worse routing solution than global optimization. Along the same lines, some theoretical trade-offs between label space and stack size are presented in [24] for special classes of graphs. However, for general networks we do not know how to decrease the label space with a larger stack, unless we go for the extreme of just using port numbers as labels. The source would then have to provide a packet with the a stack with all the port numbers on the desired path to the destination.

D. Limited hard-ware

The fact that we need neither swapping nor the stack to get load optimal routing solutions with $\mathrm{T} + \mathrm{M}$ labels suggests the relevance of limited implementations of MPLS, say, with a restricted stack-depth of two like in ATM, leaving space for stacking a detour around failed links as described in [23].

E. Routing tables on line cards

Above, we have assumed that each router has only one routing table. However, conceivably, one could have independently configured routing tables on the line-cards of the incoming links. This can sometimes be used to reduce the label space, but it doesn't help in general with the quoted bounds.

F. Concrete label spaces

The above worst-case bound of $\mathrm{T} + \mathrm{M}$ is our main result. Yet it is interesting to try to get an impression of how many labels would be consumed in some concrete networks. These concrete examples are discussed in more detail in §III.

First, consider a simple network with S sources in one cluster that communicate with T destinations in another cluster. If these clusters are connected by a single link, then any LSP based approach is going to use at least $\mathrm{S} \times \mathrm{T}$ labels. Generally, we have to be concerned about such an $\Omega(\mathrm{S} \times \mathrm{T})$ lower-bound on the label space whenever we have a limited number of bottle-necks.

In order to get a feel for what would happen with more complex networks, we made our own implementations of the algorithm of Mitra and Ramakrishnan [5] and the greedy heuristic of Xiao et al. [6], as well as our own algorithm. We applied them to some 2-level graphs produced with the GT-ITM network generator [25]. These graphs had 300 nodes and roughly 1000 edges. Mitra and Ramakrishnan [5] algorithm got 92–113,000 paths/LSPs and 20–41,000 labels, the greedy heuristic of Xiao et al. [6] got 39–47,000 paths/LSPs and 11–16,000 labels, and our algorithm got 330–460 to-trees/LSTs and 300–370 labels. As mentioned, the heuristic of Xiao et al. [6] is not guaranteed to be able to support as much traffic, and indeed, it was found to support 5–10% less traffic. Thus, even with this limited sized network, our new algorithm gained a factor 30 in label space.

We stress that concrete experiments like the one above do not give any guarantees for the table sizes needed for a

network of a given size. Network topologies are not random, so we cannot make conclusions from one to another on a simple experimental basis. In a provisioning phase, in order to experimentally ensure sufficient table sizes in the routers, one needs to make simulations for the concrete network, and these simulations are violated by subsequent changes to the topology. A safer approach is use our worst-case bound of $\mathrm{T} + \mathrm{M}$, and say, multiply it by two, thus being prepared for a doubling in size of the network. This worst-case bound is luckily sufficiently small that we can use it in practice.

G. Contents

The rest of the paper is divided into two main sections. In §II, we present our algorithm for implementing load optimal routing with $\mathrm{T} + \mathrm{M}$ MPLS labels. In §III, we present our experiments including more details on the algorithms from [5], [6]. In §IV, we discuss various applications and limitations of our approach.

II. Techniques

Recall our problem: we are given a routing solution S realizing some demand matrix D, that is, for each source destination pair $(s,t) \in S \times T$, the traffic routed from s to t is $D_{(s,t)}$. Our goal is to find an efficient MPLS implementation of a solution S' which is at least as good as S in that no link gets larger load. Thus, if S is produced as in [5] to be optimal with respect to criteria correlating positively with demands and negatively with link loads, then this optimality is inherited by S'.

As mentioned, we will solve this problem, rerouting S to some as good S', but only using $\mathrm{T} + \mathrm{M}$ to-trees and MPLS labels. In order to contextualize our result, first, in §II-A, we demonstrate that if no rerouting is needed, the number of MPLS labels needed is cubic in the worst-case. Next, in II-B, we turn to rerouting. In §II-B.1, we point out that with rerouting, LSP implementations need a quadratic number of labels in the worst-case. Thus, we really need both the rerouting and the to-trees for our linear bound. The result is formally stated in §II-B.2, and proved in §II-C-II-E.

A. The need for rerouting

We will now demonstrate that rerouting is needed since realizing a prescribed distribution of flow between each source-destination pair may require cubic table sizes.

1) With LSPs: Some unfortunate examples are illustrated in Fig. 1. First consider (a). Each source wants to send the same amount of traffic to each destination, and for each destination it spreads the traffic evenly over the P parallel links. The thick lines illustrate the distribution of flow from a particular source to a particular destination. This routing solution does give a best possible distribution of loads. Moreover, this kind of symmetric solution is a possible outcome of the techniques in [5, §2.2.2.] if one, as suggested in [5], uses an interior point method [26] to solve the linear program.

To implement this solution with LSPs as in [5, §2.5], for each source-destination pair, we need an LSP for each parallel link, so we need $\mathrm{S} \times \mathrm{T} \times \mathrm{P}$ LSPs. Moreover, since these all enter the bottleneck v, we end up with $\mathrm{S} \times \mathrm{T} \times \mathrm{P}$ labels.

The reader may object that (a) is contrived with the high node degrees, but (b) illustrates the same principle with lower degrees, using binary trees to and from the bottlenecks v and w, and with a regular bipartite graph in place of the parallel edges. Again the thick edges illustrate the distribution of flow from a particular source to a particular destination. In (b), the bottleneck v has only one incoming edge, and hence it does not make a difference to have independent routing tables on the incoming edges.

The flow decomposition from [5, §2.5] can lead to at most $\mathrm{S} \times \mathrm{T} \times \mathrm{M}$ LSPs, so we conclude that the worst-case number of LSPs and labels resulting from [5] is $\Theta(\mathrm{S} \times \mathrm{T} \times \mathrm{M})$. In the introduction, we actually quoted [5] for a better bound of $\Theta(\mathrm{T} \times \mathrm{M})$ labels. This bound stems from a variant of their approach mentioned in [5, §2.4]. We shall return to both variants in §III-B

2) With to-trees: Using to-trees, we can easily implement the routing from Fig. 1 (a) and (b) using $\mathrm{T} \times \mathrm{M}$ to-trees, for each tree can take all sources over one of the parallel links. It is therefore natural to ask if one in general can do better with to-trees. The answer is no, as illustrated in Fig. 1 (c). Each source insists on using its own parallel link from u to v, and hence no to-tree can route from more than one source to one destination. Moreover, as in (a), each source-destination pair needs a to-tree for each parallel link from v to w. Thus, we need at least $\mathrm{S} \times \mathrm{T} \times \mathrm{P}$ to-trees. In Fig. 1 (d), the same point is made with low-degree nodes, and where the bottleneck node u has only a single incoming edge, thus showing that independent routing tables in the line-cards of incoming edges do not make a difference. The examples actually show:

Proposition 1: To implement prescribed flows for each source destination pairs with MPLS, allowing label swapping, but not using the stacks, we need routing tables of size $\Theta(\mathrm{S} \times \mathrm{T} \times \mathrm{M})$ in the worst case.

Proof: As mentioned in §II-A.1, the upper bound is already realized with LSPs in [5]. The lower-bound is a slight refinement of the number of to-trees needed for Fig. 1 (c). More precisely, packets arriving u with the same label will proceed on the same path, so to proceed on different paths, they need different labels, hence different entries in the routing table at u. However, we need $\mathrm{S} \times \mathrm{T} \times \mathrm{M}$ different paths from u to the destinations, so the result follows with $\mathrm{P} \geq \max\{\mathrm{S}, \mathrm{T}\}$. ∎

Thus, to get a sub-cubic number of labels we need some rerouting.

B. Allowing rerouting

When we allow rerouting of a routing solution S, our problem of matching S is simply the constraint-based routing problem where the capacity C_e of an edge e is its load in S, and thanks to the existence of S, we know that this problem is feasible, i.e., that it has a solution.

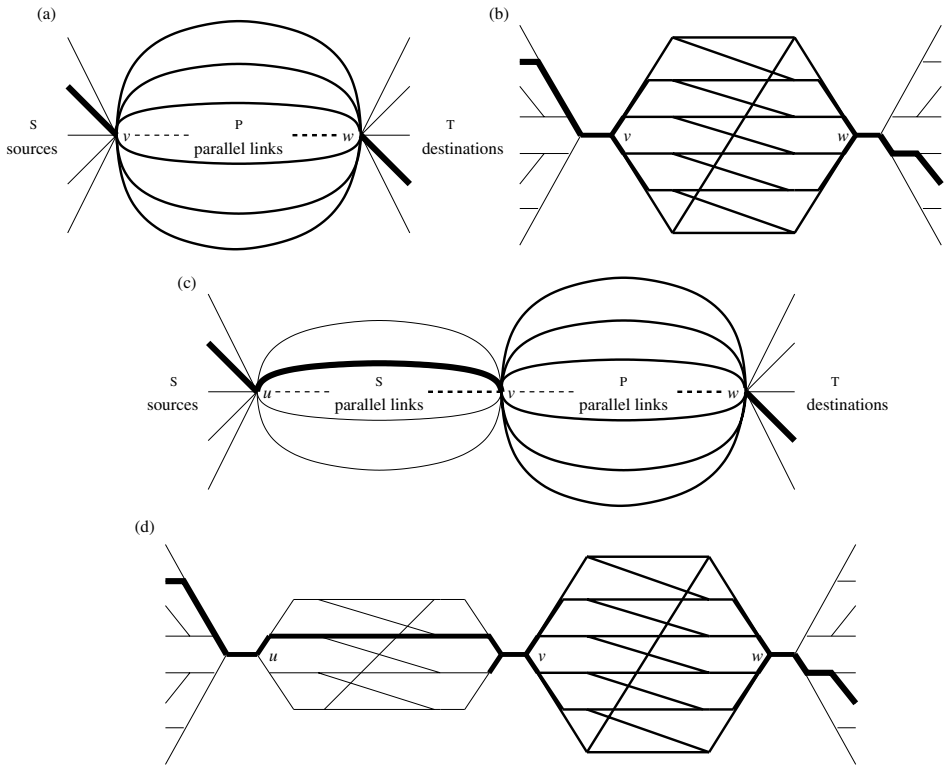

Fig. 1. Routing solutions to be rerouted.

1) With LSPs: Now, as described in [22, §17.5], using a so-called basic solution to a path flow formulation of the above constraint-based routing problem, we will get to a total of at most $\mathrm{S} \times \mathrm{T} + \mathrm{M}$ paths routing the demands between all the source-destination pairs. These paths are then implemented as LSPs with MPLS, replacing the cubic bound from §II-A with a quadratic one. This is asymptotically optimal, for obviously we need an LSP for each source-destination pair. Also, considering Fig. 1 (a), to spread the loads, we need an LSP for each parallel edge, hence at least $\max\{\mathrm{S} \times \mathrm{T}, \mathrm{P}\} = \Omega(\mathrm{S} \times \mathrm{T} + \mathrm{M})$ LSPs. Since all of these enter the bottleneck node u, u will need that many entries in its routing table. Summing up,

Proposition 2: We can solve a feasible constraint-based routing problem using $\mathrm{S} \times \mathrm{T} + \mathrm{M}$ LSPs. Moreover, using LSPs, we cannot in general get asymptotically smaller routing tables. ■

Thus, the quadratic bound is optimal for LSPs.

2) With to-trees: The result of this paper is that we can get down to a linear number of labels using to-trees and rerouting. More precisely, we will show

Theorem 3: We can solve a feasible constraint-based routing problem using $\mathrm{T} + \mathrm{M}$ to-trees. Using MPLS with swapping, but not the stacks, we cannot in general get asymptotically smaller routing tables.

The lower-bound is trivial. For example, in Fig. 1 (a), at the bottle-neck node v, we need a label for each destination and we we need a label for each parallel link, hence $\max\{\mathrm{T}, \mathrm{P}\} = \Omega(\mathrm{T} + \mathrm{M})$ entries for the routing table at v.

As for Proposition 2, the upper-bound uses basic solutions as a starting point. However, whereas the quadratic bound in Proposition 2 follows from a simple bound based on the number of equations in a path based linear programming formulation of the problem [22, §17.5], our linear bound needs a more subtle combinatorial analysis presented in §II-E.

C. The algorithm

We will now show how to construct the to-trees claimed in Theorem 3. The construction is, in itself, fairly natural. We defer to § II-E the subtle analysis showing that the algorithm only produces $\mathrm{T} + \mathrm{M}$ to-trees.

First we formulate the multicommodity flow problem corresponding to the constraint-based routing problem. This multicommodity flow problem is a special type of linear programming problem. For a more detailed discussion of multicommodity flow problems and this formulation, see [22, §17]. For each destination $t \in T$ and each edge $e = (u, v) \in E$, we wish to determine the flow of traffic $F_e^t \geq 0$ on e toward t from the sources. Note that in this formulation, we do not distinguish traffic flows from different sources if they are for the same destination. For each destination t and node $v \neq t$, we have the flow constraint:

$$\sum_{(v,w)\in E} F^t_{(v,w)} - \sum_{(u,v)\in E} F^t_{(u,v)} = D_{(v,t)}. \qquad (1)$$

Here, $D_{(v,t)} = 0$ if $v \notin S$. This flow conservation constraint says that the total traffic leaving v destined for t should be equal to the total traffic entering v destined for t plus the traffic demand from v to t. Moreover, for each edge $e \in E$, we have the capacity constraint:

$$\sum_{t \in T} F_e^t \leq C_e \tag{2}$$

This says that the total flow on each edge e is at most C_e, which in turn was the flow on e in the original solution S. Finally, to avoid generating cycles in the flows to any destination, we ask to

$$\textbf{minimize} \sum_{t \in T, e \in E} F_e^t \tag{3}$$

We now obtain a basic solution to this linear programming problem. This can be done using any linear programming solver based on the simplex method, such as the commercial linear programming package CPLEX [17], or the open-source package GLPK [18]. Several alternative methods for obtaining a basic solution will be discussed in the next section, where we also discuss the special properties of a basic solution that we take advantage of.

We will now pick a destination $t \in T$, and distribute all the flow to t on trees to t. This is done iteratively. In each iteration, some of the flow to t is moved to some tree to t. The flow variables and demands are reduced accordingly, and we continue till all demands and flows are down to zero.

Algorithm A: **Flow2Trees**(t) Takes all the flow to t and distributes it on trees to t.

A.1. while there is a sources $s \in S$ with demand to t ($D_{(s,t)} > 0$):

A.1.1. using only edges e with flow to t ($F_e^t > 0$), construct a tree R to t spanning all sources with demand to t.

A.1.2. move as much flow as possible to R.

We now need to show how to perform the steps in the while-loop. This includes showing that (1) is preserved as an invariant.

First, to find the tree R to t in step A.1.1, we do as follows. For each node v, if v has an outgoing edge with flow to t, we include an arbitrary such edge as its parent pointer in R.

To see that R gets the desired properties, note by (1) that a node has outgoing flow to t if it has incoming flow to t or demand to t. This implies that we get parent edges for all nodes $v \neq t$ with demand to t or with an incoming parent edge. Since the flow to t contains no cycles, we conclude that R is a tree to t spanning all nodes with demand to t.

Next we have to clarify what we mean by moving as much flow as possible to R. Note that R is oriented toward t, so t is implicit in R. In each node v in R, we satisfy a non-negative demand $D_v^R \leq D_{(v,t)}$ to t, and each edge e in R will carry a non-negative flow $F_e^R \leq F_e^t$ to t. This should be done subject to the standard flow constraint at each node $v \neq t$:

$$\sum_{(v,w) \in R} F_{(v,w)}^R - \sum_{(u,v) \in R} F_{(u,v)}^R = D_v^R. \tag{4}$$

Above, there is exactly one edge $(v, w) \in R$; namely the parent pointer from v.

We want to send as much flow as possible in R, so our goal is to

$$\textbf{maximize} \sum_{v \in R} D_v^R \tag{5}$$

The above tree flow problem could be solved as a linear program, but it is much more efficient to solve it with the recursive procedure below. To get a maximal flow in R to t, it is called with TreeFlow(t, ∞, R).

Algorithm B: **TreeFlow**(v, c, R) sends a maximal flow toward t, though at most c, using the sub-tree descending from v in R. The flow value is returned.

B.1. $f \leftarrow D_{(v,t)}$.

B.2. if $f \geq c$, $f \leftarrow c$.

B.3. $D_v^R \leftarrow f$.

B.4. iterate u through the children of v in R.

B.4.1. $g \leftarrow$ TreeFlow$(u, \min\{c - f, F_{(u,v)}^t\}, R)$.

B.4.2. $F_{(u,v)}^R \leftarrow g$.

B.4.3. $f \leftarrow f + g$.

B.5. return f

It can easily be verified that the flow has the desired properties, that is, for each node $v \neq t$ in R, $0 \leq D_v^R \leq D_{(v,t)}$, for each edge e in R, $0 \leq F_e^R \leq F_e^t$, and finally, (4) and (5) are satisfied.

Having found the flow in R, for each node $v \neq t$ in R, we subtract D_v^R from $D_{(v,t)}$ and for each e in R, we subtract F_e^R from F_e^t. Neither the $D_{(v,t)}$ nor the F_e^t can turn negative, and since the flow in R satisfies (4), the subtractions preserve our invariant (1).

Since the flow in R is maximal, it will saturate at least one flow edge, so in each iteration, we will lose at least one edge with flow to t. Thus Algorithm A terminates in at most M iterations. Summing over all destinations, this leads to a crude bound of T × M in-trees. A much more refined analysis in §II-D-II-E will bring this bound down to T + M.

For the MPLS implementation, we create a label ℓ_R for each tree R. For each node v in R, the router table will map ℓ_R to its parent pointer in R, or to stay if $v = t$. Also, each source s in R, we use label ℓ_R for a D_s^R part of the traffic to t.

D. Basic Solutions

The algorithm above called for a basic solution to the linear program defined by equations (1) and (2). The concept of a basic solution is fundamental to the theory of linear programming. Here, we will note a few properties of basic solutions, and some methods for obtaining one. For a fuller treatment, see [27] or [28].

Associate with each variable the column vector of that variable's coefficients in each constraint of the linear program. A basic solution is specified by a basis, that is, a subset of the variables for which the associated columns form a basis of the

vector space spanned by all the columns. The basic solution is obtained from the basis by setting the value of all variables not in the basis to 0, and then solving the remaining full-rank system for the values of the variables in the basis.

¿From this, we obtain two important properties of basic solutions:

(i) The number of variables in the basis is at most the number of constraints, in our case, $\mathrm{T} \times (\mathrm{N} - 1) + \mathrm{M}$.

(ii) All non-zero variables in the basic solution are in the basis.

For multicommodity flow problems, one can also show (see, for example, exercise 17.9 in [22]):

(iii) For every destination t, the basic flow variables to t contain a disoriented spanning tree S_t, that is, the set of edges e with F_e^t in the basis contain a tree S_t ignoring the orientation of the edges.

There are several ways to obtain a basic solution to our multicommodity flow problem. A linear programming solver based on the simplex method will automatically produce basic solutions, so we can use a commercial package like CPLEX [17], or an open-source package like GLPK [18]. Alternatively, since we only used (3) to avoid flow cycles, we can break ties with high probability by multiplying each flow variable by an independent random number in (3). We still avoid cycles, and now we get a unique basic solution with any exact linear programming solver (for example ADP [26]) or minimum-cost multicommodity flow solver (for example EMNET [29] or MCNF85 [30]). On the theoretical side, for a polynomial worst-case running time, we can use an interior-point algorithm [31], [32], followed by a crossover algorithm [33] which converts the interior-point solution to a basic solution. Finally, for a strongly polynomial worst-case running time, we can use [34].

In the experiments reported in §III, we used CPLEX, and for networks with 300 routers and roughly 1000 edges, we identified our MPLS labeling in less than 24 minutes.

E. Analysis

In this section, we are going to show that the algorithm from §II-C will generate at most $\mathrm{T}+\mathrm{M}$ to-trees and labels. We will use the properties of basic solutions mentioned above.

First we note a standard application of property (i) and (ii); namely that $(\mathrm{N}-1) \times \mathrm{T} + \mathrm{M}$ bounds the total number of flow edges to all destinations, and since each tree R to t from Algorithm A eliminates at least one flow edge to t, we get at most $(\mathrm{N}-1) \times \mathrm{T} + \mathrm{M}$ to-trees in total. Though still quadratic, this beats the crude factor $\mathrm{T} \times \mathrm{M}$ that we mentioned at the end of §II-C.

However, we claim that we can eliminate the factor $(\mathrm{N}-1)$, getting a clean linear bound of $\mathrm{T} + \mathrm{M}$. To show this, we need to employ (iii) and a more subtle argument.

Let M_t^* be the number of basic flow variables to t, M_t be the number of flow edges to t, and N_t be the number of nodes with outgoing flow to t. Here t is itself counted in N_t.

Lemma 4: $\mathrm{N} - \mathrm{N}_t \leq \mathrm{M}_t^* - \mathrm{M}_t$

Proof: Consider the disoriented spanning tree S_t from (iii). For each node $v \neq t$, let $e(v,t)$ be the edge from v on the disoriented path to t in S_t. Thus, we assign a unique basic flow variable to each node $v \neq t$.

If a node v is not accounted for in N_t, then, by flow-conservation, it has no incident flow edges to t, and hence $e(v,t)$ is not accounted for in M_t. However, by (ii), all flow edges to t are basic flow variables, so we conclude that $\mathrm{N} - \mathrm{N}_t \leq \mathrm{M}_t^* - \mathrm{M}_t$. ■

Lemma 5: Algorithm A creates at most $\mathrm{M}_t - \mathrm{N}_t + 2$ trees to t.

Proof: The proof is by induction over M_t. Consider a tree R to t found in step A.1.1, spanning all nodes with demand to t. If this is the last iteration in the sense that R carries all the flow and demand to t, then R is the set of flow edges to t. Since R is a tree, $\mathrm{M}_t = \mathrm{N}_t - 1$, so $\mathrm{M}_t - \mathrm{N}_t + 2 = 1$, matching that R is the last tree.

Now assume R is not the last tree. We subtract the flow and demands from R in A.1.2. Let M'_t and N'_t by the new values of M_t and N_t. Since we have used one tree, we need to argue that

$$\mathrm{N}_t - \mathrm{N}'_t \leq \mathrm{M}_t - \mathrm{M}'_t - 1. \tag{6}$$

Let X be the nodes losing their outgoing flow to t — then $|X| = \mathrm{N}_t - \mathrm{N}'_t$. In particular, each node in X loses its parent edge R as a flow edge to t. We need to show that at least one additional flow edge is lost.

Since the R was not the final tree, there is at least one source s with additional demand to t, that is, $D_s^R < D_{(s,t)}$. However, since the flow in R is maximal by (5), this implies that the path P in R from s to t contains a saturated edge e with $F_e^R = F_e^t$, hence which will be lost as a flow edge to t when F_e^R is subtracted from F_e^t. If P does not intersect X, e is the additional lost flow edge needed for (6).

Now suppose P intersects X. By flow conservation (1), s cannot be in X. Consider the first edge f in P entering a node in X. Then f is not a parent edge of a node in X. Moreover, since the nodes in X have no outgoing flow to t left, by flow conservation (1), the edge f cannot have any flow to t left. Thus f is an additional lost flow edge. This completes the proof of (6). ■

Proof: [of Theorem 3] We now need to prove our main result, that the total number of to-trees produced is at most $\mathrm{T} + \mathrm{M}$.

¿From (i) we have that $\sum_t \mathrm{M}_t^* \leq (\mathrm{N}-1) \times \mathrm{T} + \mathrm{M}$. However, the two preceding lemmas imply that the number of to-trees used by Algorithm A to cover the flow to a specific destination t is at most

$$\mathrm{M}_t - \mathrm{N}_t + 2 \leq \mathrm{M}_t^* - \mathrm{N} + 2$$

Adding this up over all destinations t, we get

$$\begin{aligned} \sum_t (\mathrm{M}_t^* - \mathrm{N} + 2) &\leq ((\mathrm{N}-1) \times \mathrm{T} + \mathrm{M}) \\ &\quad + \mathrm{T} \times (-\mathrm{N} + 2) \\ &= \mathrm{T} + \mathrm{M} \end{aligned}$$

to-trees in total. This completes the proof of Theorem 3. ■

III. Experiments

We conducted some experiments to get an impression of how well different MPLS configuration schemes would do in practice on some reasonably large networks, that is, not just some small toy networks that we construct by hand and depict in the text.

In order to qualify the difference between heuristics and optimal algorithms, we consider scenarios where not all demands can be satisfied. We are then both interested in how much demand is dropped and how many labels are used.

A. Greedy

First we have a greedy heuristic like the one described by one of Xiao et al. [6] (they don't give all details, so we have to make some guesses). The essential format is that demands are satisfied in order of decreasing size. When routing a demand, we only consider links with sufficient residual capacity to satisfy the demand. We then send the demand on a widest shortest path. Here, the *residual capacity* is the capacity minus the current load, and a *widest shortest path* is a hop-count shortest path along which the minimal residual capacity is largest. Using widest shortest paths was suggested in [35]. If there is no such path, the demand is dropped. We shall refer to this heuristic as *greedy no-split*. Since there are at most one path per demand, we get at most $\mathrm{S} \times \mathrm{T}$ paths, and with a single bottle-neck link, we get that many labels.

In [6], they talk a lot about the advantage of splitting, even though they don't use it in their main algorithm. We therefore decided to consider a variant that considered all links with any residual capacity, send as much as possible on a widest shortest path, and repeat the process with the remainder of the demand. We then only dropped demand from s to t if there was no path left with residual capacity from s to t. We call this version *greedy split*. Since each path either completes a demand or fills a link, this heuristic uses at most $\mathrm{S} \times \mathrm{T} + \mathrm{M}$. The example discussed just before Proposition 2 forces greedy split to use $\Theta(\mathrm{S} \times \mathrm{T} + \mathrm{M})$ labels.

Recall again, that we call the above a heuristic because it is not guaranteed to route the demands, even if it is possible. A concrete example foiling greedy is a single path of vertices $v_1 \cdots v_n$. For $i = 1, ..., \mathrm{N} - 1$, the link (v_i, v_{i+1}) has capacity 1, and this is matched by a demand of 1 from v_i to v_{i+1}. However, in addition, we have a demand of 1 from v_1 to v_{N}. If the last demand get placed first, it blocks all other demands, thus reducing the throughput to 1, but if we take the other demands first, the throughput is $\mathrm{N} - 1$. Thus, either version of greedy will only satisfy a fraction $1/(\mathrm{N} - 1)$ of the nodes. To make sure that greedy picks the wrong demand first, we can just reduce all the other demands to 0.999999999.

B. Interior

Next we consider the optimal solution based on interior point methods, as suggested by Mitra and Ramakrishnan [5]. In their optimization model, dropping as little demand as possible is equivalent to assigning the same unit value to each demand. In [5, §2.2.2] they use interior point methods to find optimal flows for each individual demands. They then decompose each flow into paths [5, §2.5]. We implement this step using greedy split from above. When applied to such a flow, greedy split makes an exact decomposition with no loss. It makes at most one path per flow edge, of which we have at most M, so with $\mathrm{S} \times \mathrm{T}$ demands, we end up with at most $\mathrm{S} \times \mathrm{T} \times \mathrm{M}$ paths. The example from II-B.1 shows that in the worst-case, they may end up $\Theta(\mathrm{S} \times \mathrm{T} \times \mathrm{M})$ labels. We call this optimal algorithm *interior demand*.

As a remark in [5, §2.4], Mitra and Ramakrishnan point out that they can reduce the complexity of the linear program, if they instead get one flow from all sources to each destination. This destination based approach essentially used the same linear program as we used in §II-C. Again we decompose this flow with greedy split, but since we now have only T flows, we only get $\Theta(\mathrm{T} \times \mathrm{M})$ paths and labels, and it is this bound we quoted them for in the introduction. We call this algorithm, *interior destination*.

We note here that Mitra and Ramakrishnan never considered the number of labels. We also note that they make a mistaken remark that we with interior destination cannot value different demands differently. Using the terminology from our own linear program from §II-C, in (1), we replace '$=$' with '$\leq$', thus allowing us to not satisfy a demand. The actual traffic sent from s to t is then $\sum_{(s,w)\in E} F^t_{(s,w)} - \sum_{(u,s)\in E} F^t_{(u,s)}$ so if that traffic has value $W_{(s,t)}$, our objective is to

$$\textbf{maximize} \sum_{s\in S, t\in T} \left(W_{(s,t)} \sum_{(s,w)\in E} F^t_{(s,w)} - \sum_{(u,s)\in E} F^t_{(u,s)} \right) \tag{7}$$

If all demands have the same unit value, as in our experiments, (7) reduces to

$$\textbf{maximize} \sum_{t\in T} \sum_{(u,t)\in E} F^t_{(u,t)} \tag{8}$$

C. Basic

Finally, we have our own algorithm. To minimize the dropped demands, we simply used the linear program described just above using (3) as a secondary optimization criteria. However, as discussed in II-D we got an optimal basic solution, that we then decomposed into at most $\mathrm{T} + \mathrm{M}$ to-trees as described in II-C. We call the algorithm *basic to-tree*.

To qualify how much the to-trees help, we also tried decomposing the basic solution into paths using greedy split. We call this algorithm *basic paths*. It is not too difficult to show that this can give us at most $\mathrm{S} \times \mathrm{T} + \mathrm{M}$ paths, like the path based multi-commodity flow formulation mentioned in §II-A.1. The proof is like a simplified version of the analysis in §II-E. The example discussed just before Proposition 2 forces basic path to use $\Theta(\mathrm{S} \times \mathrm{T} + \mathrm{M})$ labels.

D. Networks

As in [10], for our experiments, we used synthetic 2-level networks produced using the generator GT-ITM [25], based on a model of Calvert, Bhattacharjee, Daor, and Zegura [36], [37]. The 2-level networks are divided into clusters. We have *local access* arcs inside the clusters and *long distance* arcs between clusters. The local arcs got capacity 250 whereas the long distance arcs got capacity 1000. We considered networks with 300 nodes and roughly 1000 links. We constructed two networks, one with 10 and one with 20 clusters.

The 2-level networks do not provide demands. However, the model places nodes in a unit square, thus getting a distance $\delta(x, y)$ between each pair of nodes. Inspired by classical entropy models for urban traffic [38], demands were modeled as follows. For each node x, we pick two random numbers $o_x, d_x \in [0, 1]$. Further, for each pair (x, y) of nodes we pick a random number $c_{(x,y)} \in [0, 1]$. Now, if the Euclidean distance (L_2) between x and y is $\delta(x, y)$, the demand between x and y is

$$\alpha o_x d_y c_{(x,y)} e^{-\delta(x,y)/2\Delta} \tag{9}$$

Here α is a parameter and Δ is the largest Euclidean distance between any pair of nodes. Above, the o_x and d_x model that different nodes can be more or less active senders and receivers, thus modeling hot spots on the net. Because we are multiplying three random variables, we have a quite large variation in the demands. The factor $e^{-\delta(x,y)/2\Delta}$ implies that we have relatively more demand between close pairs of nodes, yet the distance on its own never has an impact bigger than a factor $\sqrt{e} = 1.648...$. It should be mentioned that the same model without the distance factor also has been used in [15] for Internet traffic. We scaled the demand so that optimal solution would drop 20%. Here, scaling the demands by s means that we multiply each demand by s. Then, for the heuristics, we could see how much more they would drop. These demand sets are called *plain*.

We noticed, however, that many links were hardly utilized, which is not surprising as the capacities where chosen independent of the demands. This is, however, very unrealistic. In reality, networks are designed to match the demands. To get a more reasonable set of capacities, we did as follows. First, ignoring capacities, we sent all demands over hop-count shortest paths, splitting traffic evenly at nodes with multiple outgoing links on shortest paths to a destinations, as in OSPF routing with unit weights and even splitting. We scaled the demands so that the maximum utilization became 96. Corresponding to the current capacities for OCx links, each link was given capacity the smallest of $x \in \{1, 3, 12, 48, 192\}$ which was greater than the scaled link load. Note here, that links with the maximum utilization of 96 get a link capacity of 192. Now we have a network with capacities that are reasonably in relation to the demands. Finally, we multiplied each demand by 1.5, modelling growth in the demand after the network capacities were designed. The challenge was then to drop as little as possible. These demand sets are called *ospf*.

E. Results

The results of all our experiments are presented in Table I. Clearly the numbers are not as bad as suggested by our worst-case analysis, but the gains from our basic to-trees are very substantial. We note that we gain about a factor 30 in number of labels over the greedy heuristics, and drop about 30% less demand. Compared with interior, which is optimal like ours, i.e., has the same loss of demands, we gain about a factor 100 in labels over interior destination, and about a factor 300 over interior demands.

We also note that our basic path only used about twice as many labels as the greedy heuristics while dropping 30% less demand.

Somewhat surprising, for these examples, the drop rates of greedy split and nosplit are always within 0.1% of each other.

The maximum time spent on finding any of our to-tree solutions was 24 minutes.

IV. Applications and limitations

We will now discuss some applications and limitations of our MPLS implementations.

A. Dividing traffic into classes

Our free rerouting of traffic may not always be desirable in connection differentiated services. Suppose we have two classes of customers: gold customers that were promised a guaranteed bandwidth, and normal customers. We may be given a routing solution with the required constraint that gold traffic uses at most 60% of the capacity of each link. The normal traffic is assumed to have lower priority in the routers so that it does not disturb the gold traffic.

Obviously, we cannot just mix the flows from the different classes, but to find an MPLS implementation with few labels, we can reroute each class independently. Then neither class increases its load on any link, and hence all classes should be happy with rerouting. The price we pay is that we use $\text{K} \times (\text{T} + \text{M})$ to-trees and labels to route K traffic classes with different capacity constraints.

We note that finding a first "good" routing solution S for differentiated services may be a non-trivial matter. Our claim is only that we given S can find an as good solution S' that we can implement with $\text{K} \times (\text{T} + \text{M})$ MPLS labels.

B. Dynamic reoptimization

Generally, we imagine that our optimization can be applied, say, on a daily basis, or whenever we are about to run out of labels. It is important to note here, that MPLS can transition much more gracefully than, say, OSPF. One can in the background prepare a new set of labels, and configure them in the router tables without disturbing the traffic. When that is done, one can transfer the traffic to the new label set, say, just 10% at a time, waiting for the traffic to stabilize after each 10%. This way one is sure that no link-load becomes more than 10% more than it would be in either configuration. Note that even if we switch all the traffic at once, we should

Plain with 300 nodes, 1026 edges and 20 clusters

	greedy		interior		basic	
	nosplit	split	demands	destinations	paths	to-trees
labels	12709	12716	81113	20877	14462	307
structures	41302	41313	256626	92622	70608	339
dropped	28.2%	28.2%	20.0%	20.0%	20.0%	20.0%

OSPF with 300 nodes, 1026 edges and 20 clusters

	greedy		interior		basic	
	nosplit	split	demands	destinations	paths	to-trees
labels	12189	12217	76730	30502	21144	364
structures	46988	47035	239520	112821	82910	459
dropped	14.2%	14.2%	7.2%	7.2%	7.2%	7.2%

Plain with 300 nodes, 1056 edges and 10 clusters

	greedy		interior		basic	
	nosplit	split	demands	destinations	paths	to-trees
labels	11569	11577	141775	30239	21497	303
structures	39969	39984	336103	97466	70763	319
dropped	28.7%	28.7%	20.0%	20.0%	20.0%	20.0%

OSPF with 300 nodes, 1056 edges and 10 clusters

	greedy		interior		basic	
	nosplit	split	demands	destinations	paths	to-trees
labels	15104	15132	92008	40214	27456	350
structures	44130	44181	237489	110224	77988	452
dropped	18.4%	18.4%	12.8%	12.8%	12.8%	12.8%

Asymptotic worst-case

	greedy		interior		basic	
	nosplit	split	demands	destinations	paths	to-trees
labels	$S \times T$	$S \times T + M$	$S \times T \times M$	$T \times M$	$S \times T + M$	$T + M$

TABLE I
MPLS CONFIGURATION SCHEMES

be pretty safe. In the worst-case, a link could get the sum of its loads in the two configurations. However, if the demands switch in random order, and a link is not dominated by a single demand, the link loads will just move nicely between the loads in the two configuarions.

For contrast, with a shorest path protocol like OSPF, the configuration is a single weight for each link, and while transitioning from one weight setting to another, the shortest paths could be all over the place, creating utter chaos in the network.

C. Error-scenarios

In this paper, we have generally ignored link-failures and the like. One approach suggested, but not implemented in [6] is to say that no link can carry more than half of any demand. Implementing this with few labels appears difficult. However, the above is not the only defense against link-failures. Two points of criticism is that losing even half may be too much, and even worse, insisting that no link carries more than half the demand may create a very bad routing solution. In the worst-case with some bottle-neck link, the requirement would only allow us to satisfy half the possible demands.

An alternative approach, mentioned earlier, is link rerouting where one pushes a label on the stack representing a detour around the failed link [23]. This approach fits perfectly with ours in that it only requires one label per link.

An alternative solution for a few particularly worry-some or common failure-situations is to have alternative sets of labels configured, and switch to these labels if needed. Because we only use $T + M$ labels to describe a complete routing solution, it is feasible to pre-configure many alternative sets of labels that we can switch to in different scenarios. For example, in our experiments, we gained a factor of 30 in number of labels. That means that we can now pre-configure what to do in 30 different error scenarios without using more space than we used to do for a single routing solution.

REFERENCES

[1] E. C. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," 2001, Network Working Group, Request for Comments, http://www.ietf.org/rfc/rfc3031.txt.

[2] The MPLS Resource Center, http://www.mplsrc.com/faq3.shtml#MPLS Deployment.

[3] D. O. Awduche, "MPLS and traffic engineering in IP networks," *IEEE Communications Magazine*, vol. 37, no. 12, pp. 42–47, Dec. 1999.

[4] D. O. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus, "Requirements for traffic engineering over MPLS," September 1999, Network Working Group, Request for Comments, http://www.ietf.org/rfc/rfc2702.txt.

[5] D. Mitra and K. Ramakrishnan, "A case study of multiservice, multipriority traffic engineering design for data networks," in *Proc. IEEE GLOBECOM*, 1999, pp. 1077–1083.

[6] X. Xiao, A. Hannan, B. Bailey, and L. Ni, "Traffic engineering with MPLS in the Internet," *IEEE Network Magazine*, pp. 28–33, Mar. 2000.

[7] J. T. Moy, "OSPF version 2," July 1991, Network Working Group, Request for Comments: 1247, http://www.ietf.org/rfc/rfc1247.txt.

[8] R. Callon, "Use of OSI IS-IS for routing in TCP/IP and dual environments," December 1990, Network Working Group, Request for Comments: 1195, http://www.ietf.org/rfc/rfc1195.txt.

[9] A. Bley, M. Grötchel, and R. Wessäly, "Design of broadband virtual private networks: Model and heuristics for the B-WiN," in *Proc. DIMACS Workshop on Robust Communication Networks and Survivability, AMS-DIMACS Series 53*, 1998, pp. 1–16.

[10] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proc. 19th IEEE Conf. on Computer Communications (INFOCOM)*, 2000, pp. 519–528.

[11] K. Ramakrishnan and M. Rodrigues, "Optimal routing in shortest-path data networks," *Lucent Bell Labs Technical Journal*, vol. 6, no. 1, 2001.

[12] F. Lin and J. Wang, "Minimax open shortest path first routing algorithms in networks supporing the SMDS services," in *Proc. IEEE International Conference on Communications (ICC)*, vol. 2, 1993, pp. 666–670.

[13] J. Cao, D. David, S. Weil, and B. Yu, "Time-varying network tomography: Router link data," *Journal of the American Statistical Association*, vol. 95, pp. 1063–1075, 2000.

[14] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving traffic demands for operational IP networks: Methodology and experience," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 265–279, 2001.

[15] A. Dwivedi and R. Wagner, "Traffic model for USA long-distance optimal network," in *Proc. Optical Fiber Communication Conference (OFC)*, 2000, pp. 156–158, TuK1.

[16] L. G. Khachiyan, "A polynomial time algorithm for linear programming," *Dokl. Akad. Nauk SSSR*, vol. 244, pp. 1093–1096, 1979.

[17] ILOG CPLEX, http://www.ilog.com/products/cplex/.

[18] A. Makhorin, "Gnu linear programming kit," http://www.gnu.org/software/glpk/glpk.html.

[19] N. Garg and J. Könemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," in *Proc. IEEE Foundations of Computer Science*, 1998, pp. 300–309.

[20] K. Long, Z. Zhang, and S. Cheng, "Load balancing algorithms in MPLS traffic engineering," in *Proc. IEEE Workshop on High Performance Switching and Routing*, 2001, pp. 175–179.

[21] Y. Wang and L. Zhang, "A scalable and hybrid IP network traffic engineering approach," 2001, Network Working Group, Internet Draft (work in progress), http://www.ietf.org/proceedings/02mar/I-D/draft-wang-te-hybrid-approach-00.txt.

[22] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, New Jersey: Prentice Hall, 1993.

[23] T. Chen and T. Oh, "Reliable services in MPLS," *IEEE Communications Magazine*, pp. 58–62, Dec. 1999.

[24] A. Gupta, A. Kumar, and R. Rastogi, "Traveling with a Pez dispenser (or, routing issues in MPLS)," in *Proc. IEEE Foundations of Computer Science*, 2001.

[25] E. W. Zegura, "GT-ITM: Georgia Tech internetwork topology models (software)," http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz, 1996.

[26] N. Karmarkar and K. Ramakrishnan, "Computational results of an interior point algorithm for large scale linear programming," *Mathematical Programming*, vol. 52, pp. 555–586, 1991.

[27] V. Chvátal, *Linear Programming*. New York, New York: W. H. Freeman, 1980.

[28] A. Schrijver, *Theory of linear and integer programming*. New York, New York: John Wiley & Sons, 1986.

[29] R. D. McBride, "Progress made in solving the multicommodity flow problem," *SIAM Journal on Optimization*, vol. 8, pp. 947–955, 1998.

[30] J. Kennington, "A primal partitioning code for solving multicommodity flow problems (version 1)," Department of Industrial Engineering and Operations Research, Southern Methodist University, Tech. Rep. 79009, 1979.

[31] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorica*, vol. 4, pp. 373–395, 1984.

[32] P. M. Vaidya, "Speeding-up linear programming using fast matrix multiplication," in *Proc. 30th Symp. on Foundations of Computer Science (FOCS)*, 1989, pp. 332–337.

[33] N. Megiddo, "On finding primal- and dual-optimal bases," *ORSA Journal on Computing*, vol. 3, no. 1, pp. 63–65, 1991.

[34] Éva Tardos, "A strongly polynomial algorithm to solve combinatorial linear programs," *Operations Research*, vol. 34, pp. 250–256, 1986.

[35] R. Guerin, A. Orda, and D. Williams, "QoS routing mechanisms and OSPF extensions," in *Proc. 2nd Global Internet Miniconference*, 1996.

[36] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. 15th IEEE Conf. on Computer Communications (INFOCOM)*, 1996, pp. 594–602.

[37] K. Calvert, M. Doar, and E. W. Zegura, "Modeling Internet topology," *IEEE Communications Magazine*, vol. 35, no. 6, pp. 160–163, 1997.

[38] P. L. Toint, "Transportation modelling and operations research: A fruitful connection," in *Operations Research and Decision Aid Methodologies in Traffic and Transportation Management*, ser. NATO ASI series: Ser. F, Computer and systems sciences, M. Labbé, G. Laporte, K. Tanczos, and P. L. Toint, Eds., vol. 166. Springer, 1998, pp. 1–27.