Exploring the Trade-off Between Label Size and Stack Depth in MPLS Routing

Anupam Gupta
Department of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213
Email : anupamg@cs.cmu.edu

Amit Kumar
Bell Laboratories
600 Mountain Avenue
Murray Hill NJ 07974
Email : amitk@research.bell-labs.com

Rajeev Rastogi
Bell Laboratories
600 Mountain Avenue
Murray Hill NJ 07974
Email : rastogi@research.bell-labs.com

***Abstract*— *Multiprotocol Label Switching* or MPLS technology is being increasingly deployed by several of the largest Internet service providers to solve problems such as traffic engineering and to offer IP services like *Virtual Private Networks* (VPNs). In MPLS, the analysis of the packet (network layer) header is performed just once, and each packet is assigned a *stack of labels*, which is examined by subsequent routers when making forwarding decisions.**

Despite the fact that MPLS is becoming widespread on the Internet, we know essentially very little about the performance one can achieve with it, and about the intrinsic trade-offs in its use of resources. In this paper, we undertake a comprehensive study of the label size versus stack depth trade-off for MPLS routing protocols on lines and trees. We show that in addition to LSP tunneling, label stacks can also be used to dramatically reduce the number of labels required for setting up MPLS LSPs in a network. Based on this observation, we develop routing algorithms and prove lower bounds for two basic problems: (1) FIXED LABEL ROUTING**: Given a fixed number of labels, we want to minimize the stack depth, and (2)** FIXED STACK ROUTING**: Given a bound on the stack depth, we want to minimize the number of labels used. Our simulation results validate our approach, demonstrating that our novel protocols enable MPLS routing on large trees with few labels and small stack sizes. Thus, our MPLS routing algorithms are applicable to a number of practical scenarios involving the provisioning of VPNs and multicast trees.**

I. INTRODUCTION

In most conventional packet-based network routing protocols, a packet makes its way from source to destination in essentially the following way. When a router gets the packet, it analyzes the packet header and decides the next hop for it. These decisions are made locally and independently of other routers, based solely on the analysis of the packet header, which contains the destination address. For example, routers using conventional IP forwarding typically look for a longest-prefix match to the entries in the routing table to decide the next hop. In general, *each* router has to extract out the information relevant to it from the (much longer) packet header. Furthermore, routers are not designed to use information about the source of the packets from these headers for forwarding purposes.

An alternative proposed to this routing model by the IETF is called *MultiProtocol Label Switching* or MPLS [7], [17]. In this, the analysis of the packet (network layer) header is performed just once, and causes the packet to be assigned a *stack of labels*, where the labels are usually much smaller than the packet headers themselves [24], [23]. At each subsequent hop, the router examines the label at the the *top* of the label stack, and makes the decision for the next hop based solely on that label. It can then pop this label off the stack if it so desires, and push on zero or more labels onto the stack, before sending it on its merry way. (We shall refer to this as *label replacement*, and the path followed by the packet as a *Label Switched Path* (LSP).) Note that there is no further analysis of the network layer header by any of the subsequent routers.

There are a number of advantages of this over conventional network layer forwarding, the obvious one being the above-mentioned elimination of header analysis at each hop. A more significant benefit, however, is that since we analyze the header and assign the stack to the packet when it enters the network, the ingress router may use any additional information about the packet to route packets differently to satisfy different QoS requirements. For example, data for time-sensitive applications may be sent along faster but more expensive LSPs than regular data. Also, the ingress router can encode information about the source as well as the destination in the labels, which cannot be done with conventional forwarding. Apart from these factors improving network performance, explicitly routed MPLS LSPs also make it much easier to do *traffic engineering* compared to conventional routing schemes, since the entire route taken by the packet can be specified very naturally on the stack [2]. All these reasons have made MPLS very popular among network and router designers, and companies like Cisco, Juniper, Lucent and Nortel have been developing routers which support MPLS protocols [4], [20].

Despite the fact that MPLS is becoming widespread on the Internet, we essentially know very little about the performance one can achieve with it, and about the intrinsic trade-offs in its use of resources. For instance, in [24], the label stack was introduced into the MPLS framework to allow multiple LSPs to be aggregated into a single LSP tunnel. However, an important observation that we make in this paper, is that label stacks can also be used to dramatically reduce the number of labels required for setting up MPLS LSPs in a network. To the best of our knowledge, this benefit of label stacks has not been pointed out before, and this is one of the primary

contributions of our work. In addition, we provide answers to several pertinent questions like: What is the depth of the stack required for routing in an n-node network, and how does this interact with the label size?

We note here that there are a number of scalability and performance related reasons for reducing the size of the label space. First, there is a growing interest among several of the largest service providers (e.g., AT&T) to use MPLS for providing *Virtual Private Network* (VPN) services. Offering these MPLS-based VPN services to thousands of customers would require the service provider to set up and manage thousands of MPLS LSPs connecting the VPN endpoints (this is specially true for Layer 2 MPLS VPNs and VPN services based on the *overlay* model which are predominant today [7]). Clearly, since only 20 bits of each 32-bit label stack entry are available for encoding the label, the label space cannot exceed 2^{20}; the implication here is that for scalability, the label space must be conserved to the greatest extent possible. A second reason for having a small label space is to reduce the size of the *forwarding table* used by each label switching router to make label replacement and forwarding decisions for each incoming label. A smaller forwarding table helps to lower memory requirements at routers and also enables them to switch packets faster.

From the above discussion, it follows that smaller label sizes are critical for achieving better scalability and performance. However, as we show in this paper, smaller label sizes are obtained at the expense of deeper stacks. Deep stacks are undesirable since each stack entry is 32 bits, and longer stacks increase the space requirements in IP packet headers. Thus, the two goals of smaller labels and smaller stacks oppose each other, and the trade-offs involved are non-trivial. The protocols for routing a set of MPLS LSPs that we develop in this paper explore these trade-offs, and attempt to strike a balance between smaller labels and smaller stacks. Previous papers on routing do not address such questions, and it is not clear whether the information theoretic bounds are close to the truth.

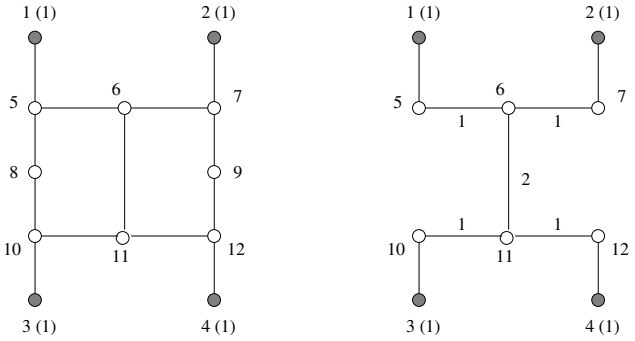
Note that a very important restriction while designing these routing protocols is that the routers can *only look at the top of the stack* to decide the next hop (as well as the set of labels to push on the stack). We quote the following paragraph from [24]:

> The processing is always based on the top label, without regard for the possibility that some number of other labels may have been "above it" in the past, or that some number of other labels may be below it at present.

Further, routers maintain a distinct label switching forwarding table for each interface, and thus the next hop decision is made based on the incoming edge on which the packet was received and the label on the top of the stack.

A. Motivating Example – VPN Provisioning in the Hose Model

We illustrate the label size/stack depth trade-off based on a practical scenario involving provisioning a single MPLS VPN in the *hose model* [6], [15], [16]. (In practice, a service provider may provision thousands of such VPNs). In the hose model, each VPN endpoint specifies a pair of bandwidths – an *ingress* bandwidth and an *egress* bandwidth. The ingress bandwidth for an endpoint specifies the maximum incoming traffic from all the other VPN endpoints into the endpoint, while the egress bandwidth is the maximum amount of traffic the endpoint can send to the other VPN endpoints. In [16], the authors showed that using a tree structure to connect VPN endpoints results in efficient utilization of network bandwidth since it enables bandwidth to be shared between VPN endpoints. The authors also propose algorithms for computing VPN trees that minimize the reserved bandwidth, and suggest that MPLS be used to set up LSPs between each pair of VPN endpoints along edges of the VPN tree. Further, since the paths connecting a pair of endpoints in the tree may not correspond to the shortest path between the endpoints, the authors point out that path setup will need to rely on the explicit routing capabilities of either RSVP-TE or CR-LDP [7]. However, the authors do not address the important problems of how labels are allocated to the LSPs connecting the VPN endpoints, or how the label stacks for the various LSPs are initialized and manipulated. Below, we use a concrete example to illustrate the issues that arise in developing routing protocols for the LSPs that connect the VPN endpoints.



(a) Network graph (b) Optimal VPN tree

Fig. 1. Routing along paths in a VPN tree using MPLS

Consider the network graph depicted in Figure 1(a). The four VPN endpoints 1, 2, 3 and 4 are shown in the figure using shaded nodes, and each VPN endpoint has equal ingress and egress bandwidths of 1. Figure 1(b) illustrates the optimal VPN tree connecting the four endpoints and the bandwidth reserved on each edge of the tree. For instance, 2 units of bandwidth need to be reserved in each direction on edge $(6, 11)$ since the combined bandwidth requirement for endpoints 1 and 2 is 2, and the combined bandwidth for endpoints 3 and 4 is also 2. Note that the path connecting endpoints 2 and 4 in the tree is not the shortest path between them (the shortest path between 2 and 4 consists of the following four edges: $(4, 12), (12, 9), (9, 7), (7, 2)$). The same holds for VPN endpoints 1 and 3. The problem is to develop MPLS routing protocols for establishing the LSPs or routing paths (along

edges of the VPN tree in Figure 1(b)) between each pair of VPN endpoints, and that use the minimum number of labels.

We first consider the problem of devising a MPLS routing protocol for the routing paths with a stack depth of only 1. Our protocol requires 4 labels, one label L_i for each VPN endpoint i. Further, every network node simply forwards each incoming packet with label L_i along the outgoing edge in the direction of endpoint i (without popping the label). For example, when node 6 in Figure 1(b) encounters a packet with either label L_3 or L_4, it forwards the packet with the same label (L_3 or L_4) along edge $(6, 11)$. Similarly, packets with labels L_1 and L_2 are forwarded by 6 along edges $(6, 5)$ and $(6, 7)$, respectively. The following table depicts, for nodes 6 and 11, the actions and outgoing edges for packets depending on the incoming edge and label.

Node	Incoming Edge	Incoming Label	Outgoing Edge	Action
6	$(11, 6), (7, 6)$	L_1	$(6, 5)$	None
	$(5, 6), (11, 6)$	L_2	$(6, 7)$	None
	$(5, 6), (7, 6)$	L_3, L_4	$(6, 11)$	None
11	$(6, 11), (12, 11)$	L_3	$(11, 10)$	None
	$(6, 11), (10, 11)$	L_4	$(11, 12)$	None
	$(10, 11), (12, 11)$	L_1, L_2	$(11, 6)$	None

It is straightforward to observe that an endpoint i can send a packet to endpoint j by simply pushing a single label L_j onto the stack. Thus, in order to implement the routing paths between VPN endpoints with a stack depth of 1, our protocol requires 4 labels, one per endpoint[1].

We next show that by increasing the stack depth to 2, we can implement all routing paths with only 2 labels L_1 and L_2. To see this, consider the routing protocol contained in the following table, and consisting, for all nodes, the outgoing edge and stack-related actions for packets ($*$ below denotes the wild card entry that matches any label).

Node	Incoming Edge	Incoming Label	Outgoing Edge	Action
5	$(1, 5)$	$*$	$(5, 6)$	None
	$(6, 5)$	$*$	$(5, 1)$	None
6	$(5, 6)$	L_1	$(6, 7)$	None
	$(7, 6)$	L_1	$(6, 5)$	None
	$(5, 6), (7, 6)$	L_2	$(6, 11)$	Pop
	$(11, 6)$	L_1	$(6, 5)$	None
	$(11, 6)$	L_2	$(6, 7)$	None
7	$(6, 7)$	$*$	$(7, 2)$	None
	$(2, 7)$	$*$	$(7, 6)$	None
10	$(10, 11)$	$*$	$(3, 10)$	None
	$(3, 10)$	$*$	$(10, 11)$	None
11	$(10, 11)$	L_1	$(11, 12)$	None
	$(12, 11)$	L_1	$(11, 10)$	None
	$(10, 11), (12, 11)$	L_2	$(11, 6)$	Pop
	$(6, 11)$	L_1	$(11, 10)$	None
	$(6, 11)$	L_2	$(11, 12)$	None
12	$(11, 12)$	$*$	$(12, 4)$	None
	$(4, 12)$	$*$	$(12, 11)$	None

With the above routing protocol, it is possible for every pair of VPN endpoints to communicate with a maximum stack depth of 2. For instance, to send a packet to endpoint 2, endpoint 1 simply pushes a single label L_1 onto the stack – the routing protocol specifies that a packet with label L_1 and entering nodes 5, 6 and 7 along edges $(1, 5)$, $(5, 6)$ and $(6, 7)$, respectively, is forwarded along the edges $(5, 6)$, $(6, 7)$ and $(7, 2)$, respectively. Similarly, endpoint 1 can send a packet to endpoint 4 by pushing label L_2 onto the stack twice. In this case, node 6, when it receives the packet on edge $(5, 6)$, pops the topmost label L_2 and forwards the packet along edge $(6, 11)$, and node 11, on seeing the second label L_2 at the top of a packet arriving on edge $(6, 11)$, forwards the packet toward endpoint 4.

Thus, for the example tree in Figure 1(b), it follows that increasing the stack depth from 1 to 2 causes a reduction in the label size from 4 to 2, when implementing all the routing paths for the tree.

B. System Model

Before describing the precise problems involving the label size/stack depth trade-off that we tackle in this paper, let us formalize the model. Each *packet* carries a *stack* of *labels*. The labels are drawn from a set Σ of size L, which is identified with the set $\{1, 2, \cdots, L\}$.

The network is an undirected graph $G = (V, E)$, where each node is a *router* and runs a routing protocol. When a packet reaches a router v on edge $e = \{u, v\}$, the router *pops* the top of the stack and examines it. (If the stack is empty, the packet should be destined for v.) The protocol at vertex v is just a function $f : E_v \times \Sigma \rightarrow (E_v \times \Sigma^*)$, where E_v is the set of edges incident to v. If $f(e, top(Stack)) = (e', \sigma)$, the router *pushes* the string σ on the stack, and then sends the packet along edge e'.

Note that there is no bound on the number of labels that can be pushed on and hence, for ease of exposition, we force the top of the stack be *popped* off when reaching a router. The quantity of interest is the maximum stack depth required for routing between any two vertices, which we denote by s. An (L, s) protocol is one which uses $O(L)$ labels, and has maximum stack-depth $O(s)$.

C. Problem Formulation

In this paper, we devise routing protocols for sending packets between a set of n nodes along a specified set of routing paths, one for each pair of nodes. We consider the following two restrictions on the set of routing paths.

- ROUTING ON A LINE. In this case, the n nodes are along a path P_n. The set of routing paths for routing packets essentially consists of all subpaths of P_n, each subpath carrying packets between the two endpoints of the path. Thus, a packet between an arbitrary pair of nodes u, v on P_n follows a route along a subpath of P_n.
- ROUTING ON A TREE. In this case, the n nodes communicate only along edges of a tree T connecting the nodes. Thus, the set of routing paths consists of all the (unique) paths in T between every pair of nodes in T.

Note that the above formulation for trees is more general than the VPN tree example presented in Section I-A. In

[1]It can be shown that with a stack depth of 1, the set of routing paths cannot be implemented with less than 3 labels.

Section I-A, we only considered routing paths between the leaves of the tree (that is, VPN endpoints 1, 2, 3 and 4 in Figure 1(b)), while in the above tree routing problem, the routing paths considered consist of the (unique) paths in the tree between all pairs of nodes (that is paths between all the nodes chosen from $1, \ldots, 12$). Clearly, the restricted model in which we restrict the set of paths to only those that connect a subset of nodes in the tree (e.g., leaf nodes), has numerous practical applications that include implementation of *multicast trees* [13] and VPNs [6], [15], [16] (as described in Section I-A). While the results that we present in this paper (lower bounds as well as routing protocols) are for general routing on a tree, they are also applicable to scenarios in which the set of paths is restricted to be only between specific endpoints.

A routing protocol for a line/tree essentially specifies the actions performed by each node on the stacks of incoming packets such that for each path in the line/tree, packets between the endpoints traverse the path. A good routing protocol is one that uses the minimum possible set of labels (since this would enhance router performance), while pushing the minimum possible number of labels on the stack for each packet (since this would keep the packet sizes small). Unfortunately, there is a trade-off involved here, and the two goals of minimizing label sizes and minimizing stack depths are conflicting. This was shown earlier in Section I-A for the example tree in Figure 1(b). With a stack depth of 1, 4 labels, one per VPN endpoint, were needed to route packets between the leaves of the tree. However, with a stack depth of 2, only 2 labels were required.

There are two natural problems that can be formulated based on this interplay between the label size L and the maximum stack depth s:

- FIXED STACK ROUTING: In this problem, we are given a bound s on the depth allowed for the stack, and we want to find a routing protocol that minimizes the number of labels L used.
- FIXED LABEL ROUTING: This is the dual problem of FIXED STACK ROUTING, in which we are given a fixed set of L labels, and we want to give a routing protocol that minimizes the maximum stack depth s used by it.

There are some obvious observations that can be made: In any graph G, if the stack depth is bounded by 1, we clearly need n labels, else we will not be able to even distinguish between the n nodes of the graph. A simple extension of this is that if the stack depth is s, we need at least $L \geq n^{1/s}$ labels; and that a label set of size L requires a stack depth of $\log n / \log L$. We will refer to this as the *information-theoretic* bound, and in the rest of the paper, this will be the holy grail towards which we shall strive.

However, in some cases, we can show that the information-theoretic bound is provably weak, and that no routing protocol can achieve these trade-offs. This should not be very surprising to the reader, since the information-theoretic bound just ensures that we have enough space to encode the destination of the packet; it does not account for the extremely restricted way in which we access the information we encode in the stack.

The nature of this restricted form of access is, however, reflected in the next lower bound: If the graph is a *tree* T, the number of labels is at least $\Delta - 1$, where Δ is the maximum degree of a vertex in T. Indeed, let v be a vertex of degree Δ, then a packet reaching v with a non-empty stack must decide which edge to go out on, and there are $\Delta - 1$ possibilities.

D. Our Contributions

In this paper, we undertake a comprehensive study of the label size versus stack depth trade-off for MPLS routing protocols on lines and trees. Our study involves both proving lower bounds for fixed label and fixed stack routing, as well as developing new routing protocols for a variety of situations. Recall from Section I-B that an (L, s) protocol is one which uses $O(L)$ labels, and has maximum stack-depth $O(s)$. The main contributions of our work can be summarized as follows.

- NOVEL PROTOCOLS FOR FIXED STACK ROUTING: In Section II, we first present a routing protocol for routing on a line with maximum stack depth of s that uses only $sn^{1/s}$ labels, which is quite close to the information-theoretic bound of $n^{1/s}$. The situation for trees is more involved, and we show that the information-theoretic bound can be very weak. Specifically, we prove that for a stack depth of 2, as many as $\Omega(n^{2/3})$ labels may be required for a tree with $\Delta = n^{1/3}$, whereas the information theoretic bound is only $\Omega(n^{1/2})$. To handle this seemingly dire situation, we propose two solutions: on one hand, we give a protocol that given a target stack depth s, uses only $O(\Delta + sn^{1/s})$ labels but violates the bound on the stack depth by a factor of 2. On the other hand, if we are not allowed to violate the bound on the stack depth, we show that it is possible to route on a tree with $\Delta sn^{1/s}$ labels.
- PROTOCOLS FOR FIXED LABEL ROUTING: As a first step, we study routing on the path P_n. We give a routing protocol using L labels requiring stack depth $O(\log_L n)$ only, which is within a constant factor of the information-theoretic bound. These protocols serve as building-blocks when we go to arbitrary trees. We use them in conjunction with a variant of the so-called *caterpillar decomposition* [19], [18] of trees into paths to get a $(\Delta + L, \frac{\log^2 n}{\log L})$ routing protocol. (Recall that if the maximum degree of a tree is Δ, then we clearly require at least $\Delta - 1$ labels.) Note that the latter protocol can give us stack depth $O(\log^2 n / \log\log n)$ with $\Delta + O(\log n)$ labels: we improve this protocol to get a $(\Delta + \log\log n, \log n)$ protocol as well. A number of the fixed label protocols in Section III are based on concepts presented by us in [14].
- SIMULATION RESULTS DEMONSTRATING THE EFFECTIVENESS OF OUR PROTOCOLS: In order to gauge the effectiveness of our protocols, we conducted a series of simulation experiments on a broad range of network graphs generated using the Waxman topology

model [25]. Our simulation results in Section VI validate our approach, demonstrating that our novel protocols enable routing on large trees with few labels and small stack sizes. For instance, with our protocols, it is possible to route on a tree containing 1000 nodes with only 70 labels and a stack of depth 3! Clearly, these savings in labels for a single tree have the potential to translate into fairly substantial reductions in label consumption when thousands of such trees are provisioned by service providers (e.g., for MPLS-based VPNs).

Furthermore, in Section IV, we show that developing *optimal* routing protocols frequently requires being able to solve some interesting combinatorial optimization problems, most of which turn out to be NP-hard.

On an unrelated note, we would like to point out that we have not made significant efforts to optimize the constants in our analyses. However, the constants involved are small, and the algorithms underlying our protocols can be easily implemented in practice; a claim which is vindicated by the simulation results of Section VI.

II. Fixed stack routing

In this section, we consider the problem of routing on a line or a tree when there is an upper bound on the stack depth s. The objective is to minimize the number of labels used for this routing.

A. The n-vertex path P_n

We first consider the special case where all the n vertices lie on a single path, and let them be called $0, 1, 2, \ldots, (n-1)$ from left to right. It is clear to see that there is a trivial lower bound of $n^{1/s}$ on the number of labels, since the stack (of depth s) should be able to encode n distinct addresses. As an upper bound, we prove the following theorem:

Theorem 1 *Given a bound of s on the maximum stack depth, there is a protocol for routing on the line that uses at most $sn^{1/s}$ labels.*

In this protocol, each label consists of a tuple $\langle i, p\rangle$. Here i is a number between 1 and $n^{1/s}$, and p is a *position* between 1 and s. Using these labels, we can use a stack of depth at most s to encode any number smaller than n: we just look at its representation in base $\lceil n^{1/s} \rceil$, and for each non-zero digit, we push the label corresponding to the tuple $\langle$*value of the digit, position from the right*$\rangle$. If the stack is created so that the positions increase from top to bottom, it is simple to perform decrement operations by just popping the top label and (potentially) pushing some labels on top of the stack.

For example, if $n = 256$ and $s = 4$, the encoding of 178 is 2302, the representation of 178 in base $4 = (256)^{1/4}$. The stack corresponding to this is $[\langle 2, 4\rangle, \langle 3, 3\rangle, \langle 2, 1\rangle]$, where $\langle 2, 1\rangle$ is the label at the top. To decrement this, the top is popped, and the label $\langle 1, 1\rangle$ is pushed onto the stack, which now corresponds to $(2301)_4 = 177$.

Now if a vertex i wants to send a packet to a vertex j to (say) its right, it encodes $j - i - 1$ on the stack as above, and sends the packet to its neighbor to the right. (Note that since j is to the right of i, $j > i$ and hence the value $(j - i - 1)$ is between 0 and $n - 1$.) When a vertex receives a packet which has a non-empty stack, it decrements the value as described above. It is easy to see that the vertex j will get the packet with an empty stack, and hence accept it.

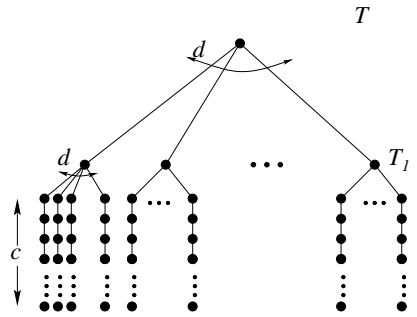


Fig. 2. Proof of Theorem 2.

B. Routing on trees

In this section, we turn our attention to routing using MPLS on trees, trying to minimize the number of labels given a bound s on the stack depth. In contrast to the path where $O(n^{1/2})$ labels sufficed, we show in Theorem 2 that as many as $\Omega(n^{2/3})$ labels may be required, if the stack depth is bounded by 2.

To handle this seemingly dire situation, we propose two solutions: on one hand, we give a protocol, that given a target stack depth s, uses only $O(\Delta + sn^{1/s})$ labels but violates the bound on the stack depth by a factor of 2. If we are not allowed to violate the bound on the stack depth, we can manage with $\Delta sn^{1/s}$ labels.

1) Lower Bounds: In this section, we will show that the information theoretic bound of $O(n^{1/2})$ is very weak for the case when the stack depth is bounded by 2, and is in fact off by a polynomial factor.

Theorem 2 *There are trees on n vertices for which the minimum number of labels required with stack depth 2 is $\Omega(n^{2/3})$.*

Proof: Consider the tree T in Figure 2. T is obtained as follows: consider a rooted tree of depth 2 such that each internal node has d children. Now, attach a path of length c at all the leaves of this tree. So, T has $n = O(cd^2)$ vertices. Let us denote the set of vertices in these paths of length c as V'. We shall refer to the children of the root as *level one nodes*. We will only need to consider the case when the root wants to send packets to V'.

Fix a routing protocol which uses a stack depth of at most 2 and L labels. For each node in V', the root sets up a stack. Let $\mathcal{S}$ be the set of all these stacks. For every depth 2 stack in $\mathcal{S}$, consider the first vertex where the label in the top of the stack gets popped off. (If the packet has a stack of one label only, we imagine the root popping off the top; and there can be at most d packets starting with an empty stack.) Since there are L labels, there must be at most L such distinct vertices — call

these special set of nodes V''. Further, there must be a level one node which has at most L/d descendants in V'' — let u be this node and T_1 be the subtree rooted at u. Now, u has d different paths below it. At most L/d paths can contain a node from V''. So, u must use the lower label in the stack to route to the remaining $d - L/d$ paths (because, no stack pop operations occur in these paths and so, the stack depth for all packets destined to any node in these paths must be 1). But there are $c(d - L/d)$ vertices in these paths. Therefore, it follows that $L \geq c(d - L/d)$. This gives us that $L \geq cd^2/(c + d)$. Now setting $c = d = n^{1/3}$ gives us the claimed bound. ■

This shows that we cannot even hope for a result like $\Delta + sn^{1/s}$, which we had for the path. In the next subsection, we will give two results, one of which gets a good performance but violates the bound on the stack depth, while the other respects the bound but uses more labels.

2) Upper Bounds: The crucial fact at the heart of the positive results is that for any tree T and any set S of vertices in T, there is a *separator* vertex v such that deleting v breaks T into several parts, none of these connected subtrees containing more than $|S|/2$ vertices from A. Recursively finding these separator vertices in these subtrees, the following fact can be proved:

Proposition 3 *For any tree T, a subset S with n vertices in T, there is a subset A of at most $3n^{1/s}$ vertices of T whose deletion causes each connected subtree to have at most $n^{(s-1)/s}$ vertices of S.*

Before we prove this, let us prove a lemma:

Lemma 4 *Given a forest F, and a subset S of n vertices in F, by deleting at most one vertex, we can get two disjoint subforests F_1 and F_2 of F such that each of these contain at most $2/3$ of the vertices of S.*

Proof: Suppose each connected component of F has at most $n/3$ vertices from S; then we start collecting the smallest components of F until get between $n/3$ and $2n/3$ vertices of S. (We cannot directly go from below $n/3$ to above $2n/3$, since each piece is small.) This gives us F_1, and the rest of the trees form F_2.

If one of the pieces has between $n/3$ and $2n/3$ vertices from S, this can be designated as F_1, and the rest will be F_2. Finally, if one of the pieces has more that $2n/3$ nodes from S, we find the aforementioned separator vertex and delete it. Now each remaining tree will have at most $n/3$ vertices from S, and we can form F_1 and F_2 as in the first case. ■

Proof: (Proposition 3) To get this, we maintain a family of vertex-disjoint subforests of T, successively breaking some subforest until each has fewer than $t = n^{(s-1)/s}$ from S. Clearly, this will imply that each tree in any forest has fewer than t vertices, thus proving the result.

At the beginning, the family just has T itself. If the family has some forest F_i with $n_i > t$ vertices from S, we use the above lemma, and break it into two subforests $F_{i,1}$ and $F_{i,2}$ such that each resulting subforest $F_{i,j}$ has between $n_i/3$ and $2n_i/3$ vertices from S. We now remove F_i from the family and add $F_{i,1}$ and $F_{i,2}$ to it. The process stops when all forests have less than t vertices from S.

To count the number of vertices deleted, let us model the above process by a tree, whose vertices are the various subforests involved in the above process. Let T be the root of this tree, and the children of F_i be the two forests $F_{i,j}$ created by applying lemma 4 to it. Note that each forest has exactly two children, and hence this is a binary tree. Furthermore, each forest at a leaf of this tree has at least $t/3$ vertices from S, else its parent would not have been split. Hence there are at most $3n/t = 3n^{1/s}$ leaves in this tree, which also bounds the number of internal nodes. However, each internal node corresponds to at most one deleted vertex, which shows that the set A of deleted vertices has size at most $3n^{1/s}$. ■

Using this, we can prove the first result. The statement of the theorem is stronger than required, but useful for pushing the induction through.

Theorem 5 *Given any tree T, a subset S of n vertices in T, and a value s, there is a protocol to send messages to vertices in S that uses stack depth at most $(2s - 1)$ and $\Delta + 3sn^{1/s}$ labels.*

Proof: The proof of this proceeds by induction on s and n. If $s = 1$, for any n, we can just use n distinct labels, one for each vertex of S. To send a message to $v \in S$, the vertex u simply places the label corresponding to v on the stack, each intermediate vertex on the unique path sending it on to the next, and the penultimate vertex on this path popping the label off the stack before forwarding the message to v.

For $s > 1$, we use Proposition 3 to get the set A of $3n^{1/s}$ separator vertices. We will now use Theorem 5 inductively in two different ways. Let the vertex u want to send a message to v, and suppose there is a vertex $x \in A$ that lies on the (unique) $u - v$ path closest to v. We can then use the above protocol for the base case (with $3n^{1/s}$ labels) to send the message to x. The next label on the stack is one of the Δ labels, which tells x which one of its (at most) Δ edges should it send the packet out on. Now the packet is in one of the subtrees of size $\leq n^{(s-1)/s}$, and inductively, we can use stack depth at most $2(s-1) - 1$ and $\Delta + 3(s-1)n^{1/s}$ labels to route within this subtree.

Note that though the $3(s-1)n^{1/s}$ labels have to be distinct from those already used, the Δ labels can be reused. Putting everything together, we get the claimed bound of $\Delta + 3sn^{1/s}$ labels and stack depth $2s - 1$. ■

Note that in the above proof, we can combine the label indicating which vertex x to go to, and the label indicating the edge to take out of x into one label. This increases the size of the labels to $\Delta \times sn^{1/s}$, and hence gives the second positive result of the section:

Theorem 6 *Given a tree T with n vertices, and maximum degree Δ, it is possible to achieve stack depth s with $\Delta sn^{1/s}$ labels.*

Note that Theorem 5 can be massaged to give another procedure for routing on the line with maximum stack depth s and $sn^{1/s}$ labels, giving another proof of Theorem 1.

III. Fixed Label Routing

In this section, we are given a fixed number L of labels, and we want to devise protocols that use as little stack depth as possible. We begin by showing that we can achieve very good bounds on a line; furthermore, the algorithm we develop for a line is simple and easy to implement. We then extend this algorithm to get good routing schemes on trees. The algorithms described in this section are based on ideas presented by us in [14].

A. Routing on a Line

We begin by considering the case when we just have two labels. The information theoretic bound says that $\log_2 n$ bits are required. This can clearly be achieved when the routers can look at all the bits of the address. However, in the MPLS model, where each router can look at just a single bit (without even knowing its bit position), it is perhaps surprising that we can perform routing with $3\log_2 n$ stack depth.

Since the direction of travel of the packet is decided by which edge it enters the vertex, it is enough to give a procedure to send a packet from left to right. Let the vertices on the path P_n be numbered $0, 1, \ldots, n-1$, and let the two labels be 0 and 1. Direct all edges in P_n from left to right, and assign label 0 to these edges. Now add some new directed edges E' to this graph, each edge in E' being also directed from left to right, such that each vertex v has at most one edge in E' going out of it. Assign each edge in E' the label 1. It can be shown that there is a way of constructing E' such that the edge set $E' \cup P_n$ satisfies the following two properties :

- *Low-diameter Property:* For any two vertices $u < v$, there is a directed path from u to v of length at most $3\log n$.
- *Nesting Property :* Let $u < u' < u''$ be three distinct vertices on the line ordered from left to right. If (u, u'') and (u', v') are two directed edges in E', then v' does not lie to the right of u'', i.e., $v' \leq u''$. Essentially, no two edges in E' *cross* each other; either they span disjoint portions of the line, or one is contained within the other.

One such example for $n = 16$ is shown in Figure 3, where the solid edges are labeled 0, and the dotted edges are labeled 1. Note the recursive structure of the construction: to build a graph G_{2^k} on 2^k nodes, take 2 copies of the graph $G_{2^{k-1}}$ on 2^{k-1} nodes and attach them in series. (A graph on 2 nodes is just a single arc.) This gives a graph on $2^k - 1$ vertices. Now we take a new vertex and attach it to the leftmost vertex by an arc labeled 0, and to the rightmost vertex by an arc labeled 1. This new vertex becomes vertex 0 in the new graph G_{2^k}, and the other vertices get suitably renumbered. In general, a graph G_n on n nodes is obtained by taking a graph on $2^{\lceil \log_2 n \rceil}$ nodes, and retaining only the leftmost n nodes.

The nesting property ensures the following fact in $P_n \cup E'$ for shortest paths defined in terms of the number of hops, which we state without proof.

Lemma 7 *Let $u < u' < v' < v$ be four distinct nodes on the line P_n. If the shortest path P from u to v in $P_n \cup E'$ contains v', then the shortest path from u' to v contains v'.*

We next describe the actual routing protocol. Given a node u and a stack of labels $l_0, \ldots, l_r$ (l_0 being on the top), we define the *path defined by the stack* by the sequence of edges obtained by starting from u and following the edges labeled $l_0, \ldots, l_r$. If u wants to send a packet to node v ($u < v$), the stack is initialized so that the path defined by the stack is a shortest path from u to v. Furthermore, we maintain the invariant that when a node u' receives the packet, the path defined by the stack at that point is a shortest path from u' to v. Now the Low-diameter property ensures that the stack depth is at most $3\log n$.

We now show how to maintain the invariant. Let the packet be at u' and let the edges labeled 0 and 1 originating from u' be $e_0 = (u', u'')$ and $e_1 = (u', u''')$ respectively. (If there is no label 1 edge from u', the argument gets even simpler). Note that edge $e_0 \in P$ and $e_1 \in E'$. Thus, a packet can be forwarded along e_0 but not along e_1. Suppose the top of the stack contains label 0. Then u' simply pops this label and sends the packet to u'', which must be the next vertex on the path. Since the path defined by the stack when it was at u' contained u'', it is easy to show that the path defined by the stack when it is at u'' is also a shortest path from u'' to v. Otherwise, the top of the stack has a 1. In this case, u' pops this label and pushes a set of labels which encode a shortest path from u'' to u'''. Lemma 7 ensures that the shortest path from u'' to v contains u''' as an intermediate node, which implies that the path defined by the stack when it reaches u'' is also a shortest path from u'' to v, maintaining the invariant.

In fact, the above process to forward a packet so as to maintain the invariant is extremely simple. As always, if a router gets a packet, and the stack is not empty, it performs the actions described below and sends it out on the other edge. Each router pops off a 0 if it sees one on top of the stack; the difference is in the handling of the 1's. If the router has outdegree 1, it just pops off the 1 (and in fact, such a vertex will never see a 1); if it has outdegree 2, it replaces it by two 1's.

The following theorem follows from the above discussion:

Theorem 8 *There is a protocol for routing on the n-vertex path which uses 2 labels and stack depth at most $3\log n$.*

It is trivial to encode the top $O(\log L)$ labels on the stack in a label of size L, and hence we can use the above protocol to get the following theorem:

Theorem 9 *There is a protocol for routing on the n-vertex path which uses L labels and stack depth at most $O(\log_L n)$,*

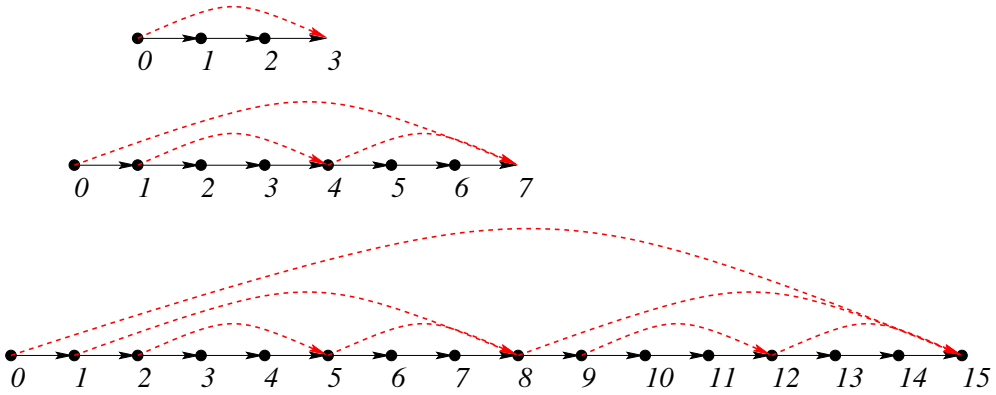


Fig. 3. How to build a 16 node example.

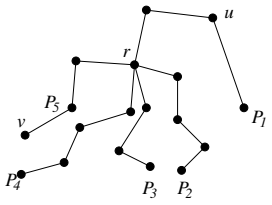


Fig. 4. A spider.

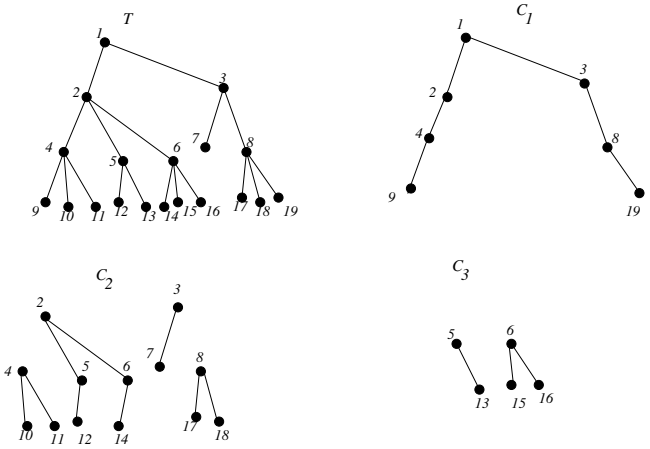


Fig. 5. A spider decomposition.

which is within a constant factor of the information-theoretic bound.

B. Routing on a Tree

The task of defining a routing protocol on a tree like a path or a star is much facilitated by their structural simplicity; however, things tend to get more complicated as the "complexity" of the tree grows. A very useful notion of complexity of a tree is the *spider dimension* $\kappa(T)$ of a tree.

Suppose our tree was a set of paths joined together at a single vertex, which we call a *spider*. (E.g., look at Figure 4.) To send a packet on this tree from u to v, we could use the protocol for routing on the line to send the packet from u to the root r, and then from r to v. The depth of the stack would be (about) twice the stack depth for routing on a single path. Note that this generalizes very well: suppose we could decompose the tree into paths such that the number of paths seen on the unique path between any two points was bounded by k, we could use the same idea to route, and the stack depth would be at most k times the stack depth to route on a line. (We will see later that we can do better than a factor k increase in stack depth.)

This idea is at the heart of a *spider decomposition*: it decomposes the tree into spiders such that at most $\kappa(T)$ different spiders are seen while traveling between any two vertices of the tree. Given a tree T with l leaves, a spider decomposition of T is a set of spiders $C_1, \ldots, C_t$ such that each spider C_i is a subgraph of T, and each edge of E appears in exactly one of the spiders. The *spider dimension*[2] of a tree $\kappa(T)$ is the smallest number t such that there exists a spider decomposition of T into t spiders. Figure 5 gives an example. There is a well known theorem which states that the spider dimension of any tree is at most $\log l \leq \log n$. To find this decomposition, take a rooted tree with l leaves and find a set of spiders such that deleting their edges leaves a forest with each subtree having at most $l/2$ leaves. (The details can be found in [18], [19], and the algorithm can be very easily implemented.)

This immediately gives us the following theorem:

Theorem 10 *Given a tree T with maximum degree Δ, there exists a routing protocol for T which uses $\Delta + L$ labels, and in which the stack depth is at most $(\log_L n)\ \kappa(T) = O(\log^2 n / \log L)$.*

For the special case of $L = \log n$, Theorem 10 gives us a stack depth of $O(\log^2 n / \log\log n)$. For this case of $L = \log n$, we can get a stack depth of $O(\log n)$ using the algorithm in the proof of Theorem 5. However, this is not the best we can do, and we can improve on this to get a protocol with $L = 2\log\log n$ and stack depth $O(\log n)$. Admittedly, this does not fit into the model where L is specified; however, $2\log\log n$ is smaller than 12 for all reasonably sized networks,

[2]We use this term at the risk of confusing the arthropod food-chain, since these are often called *caterpillar decompositions* in the literature.

and these results show that if we are dealing with bounded degree trees, we can achieve a logarithmic stack depth with very few labels.

Theorem 11 *There exists a* $(\Delta + \log\log n, \log n)$ *routing protocol for trees.*

The proof of Theorem 11 looks closely at the properties of spider decompositions, and uses these to reduce the number of labels used. The detailed proof, which we omit for lack of space, appears in a separate paper. We would like to point out that despite the slight complexity of the proof, the protocol is fairly simple and can be easily and efficiently implemented; we give simulation results for the above algorithm in Section VI.

IV. Hardness

The above approaches suggest a few natural optimization problems, all of which turn out to be NP-hard.

The first problem we define is Minimum Labels, which is a special case of Fixed Depth Routing. In this, we are given a graph G, a source node r, and a subset of vertices S. The objective is to find smallest size set of labels such that r can send messages to all the vertices in S on shortest-paths using stacks of depth at most 2. We state the following theorem without proof.

Theorem 12 *The problem* Minimum Labels *is NP-hard.*

The second natural problem, which is inspired by our constructions in Section II-B, is Minimum Shattering Set. In this problem, the input is a graph G and a number K, and the objective is to find the smallest set of vertices S in G whose removal breaks the graph into connected components, each of size at most K. Any feasible solution S to this problem can be used to route using stack depth $s = 2$ and $K + |S|$ labels.

This problem also turns out to be NP-hard, since the case where $K = 1$ is the Vertex Cover problem, which is NP-hard. Furthermore, if the number of vertices in G is n, the case $K = n/2$ is the NP-hard Graph Bisection problem, for which the best known algorithms can only guarantee a solution within $O(\log^2 n)$ of the optimum value.

V. Related Work

Distributed packet routing problems in networks has been widely studied, e.g., see [9], [10], [22], [5], or [11] for a survey of some of the issues and techniques. In these papers, the emphasis has been to reduce the sizes of the routing tables and the sizes of the packet headers while performing near-shortest path routing. Our work is incomparable to this line of work. In MPLS, setting up the initial stack may require more memory than conventional routing problems, but once the stack is set up, the memory needed by each router to just forward the packets is very small.

There has also been lot of work on finding sparse *spanners* of graphs [1], [3]. However, these results are interesting only when the graph is not sparse, whereas the problems we address in this paper are non-trivial even for bounded degree graphs.

Another different (but related) large corpus of work has studied the problem of distance labeling of graphs [26], [21], [12]. Distance labeling problem involves assigning short labels to vertices, so that an algorithm given the labels of any two vertices in the graph can deduce the shortest distance between them. (Note that the algorithm does not have any other knowledge of the graph). Although this appears to be similar to problem, they turn out to be technically quite disparate.

To begin with, the distance labeling problem is trivial when the input is a path, but finding good MPLS routing schemes for the path is already non-trivial. In the case of trees, the proof that all trees have $O(\log^2 n)$ size distance labels [21] relies on balanced vertex separators. This concept can be used to give a $(\Delta + \log n, \log n)$ MPLS routing scheme on trees, but there is no obvious way to improve this result. However, our techniques allow us to get a better $(\Delta + \log\log n, \log n)$ MPLS scheme.

VI. Experimental Results

We now describe the simulation results obtained from implementing our routing protocols. The major findings of our study can be summarized as follows :

- If we restrict ourselves to unit stack depth, then the number of labels needed can be as large as the number of nodes. Our study confirms the fact that having a stack depth greater than 1 can lead to a significant reduction in the number of labels needed. For instance, on trees with 1000 nodes, we found that increasing the stack depth from 1 to 3 decreased the number of labels from 1000 to about 70, which is more than a 90% reduction in the number of labels. Although this difference may not seem significant for a single provisioned tree network (after all, a router can easily maintain 1000 labels), but consider a scenario where several hundred VPN trees are provisioned over the same network. Here, such savings for a single VPN network can translate to significant savings of labels.
- If we are constrained by the number of labels for a provisioned tree (again this can happen if there are many such trees which need to be provisioned using MPLS), then our experimental results show that we can deal with this by increasing the stack depth slightly.

A. Network Generation Models

We tested our algorithms on two different network models. One network generator was based on the work by Waxman [25], the other by Fatoutsos et. al. [8]. For both models, we generated graphs containing 1000 nodes in our experiments. We then constructed a spanning tree by chosing a node at random and building the shortest path tree out of it. Observe that the shortest path tree is recommended for provisioning VPN networks in Kumar et. al. [16].

- **Waxman Model. [25]** In this model, nodes are placed on a plane, and the probability for two nodes to be connected by a link decreases exponentially with the Euclidean

distance between them. In our experiments, we used two different sets of values for the parameters which control the density of short edges in the network, α, and the average node degree, β. In one set of experiments, we set these values to 0.4 and 0.02 respectively, and in the other set, we set these to 0.6 and 0.04 respectively.

- **Power Law Model. [8]** In this model, the node connectivity follows a power-law rule: very few nodes have high connectivity, and the number of nodes with lower connectivity increases exponentially as the connectivity decreases. This model is based on Internet measurements, where a node is an autonomous system. Again, we used two different values for the average edge density parameter, m. We set m to 2 and 4.

B. Experimental Results

We carried out two sets of experiments. In one case, we studied the effect of varying stack depth on the number of labels needed. In the other case, we studied the effect of varying the number of labels on the stack depth needed.

- **Fixed Stack Routing.** The algorithms described in Section II work well when the stack depth is very small. We implemented the algorithm inherent in the proof of Theorem 5. This algorithm runs in $O(s^2n)$ time, where s is the stack depth. Considering s as a constant, we see that this algorithm is also very efficient. Figure 6 depicts the results obtained by running this algorithm. ¿From the results, it follows that even small increases in the stack size can result in huge savings (close to 90%) in the number of labels. For instance, increasing the stack depth from 1 to 3 causes the number of labels needed for routing on the tree to decrease from 1000 to barely 70. Also, observe that initial increases in the stack depth result in bigger reductions in label sizes.
- **Fixed Label Routing.** The algorithm inherent in Theorem 10 allows us to control the number of labels. Hence, it is suitable when we have very limited number of labels. Observe that the maximum degree of the tree, Δ is a lower bound on the number of labels needed. For the sake of comparing different sets of experiments, we subtract the quantity Δ from the number of labels needed in the plots (the actual value of Δ in the different cases is mentioned on the plots). Figure 7 illustrates the results obtained. Note that even small initial increases in label sizes (beyond maximum degree) result in substantial decreases in the stack depth required for routing.

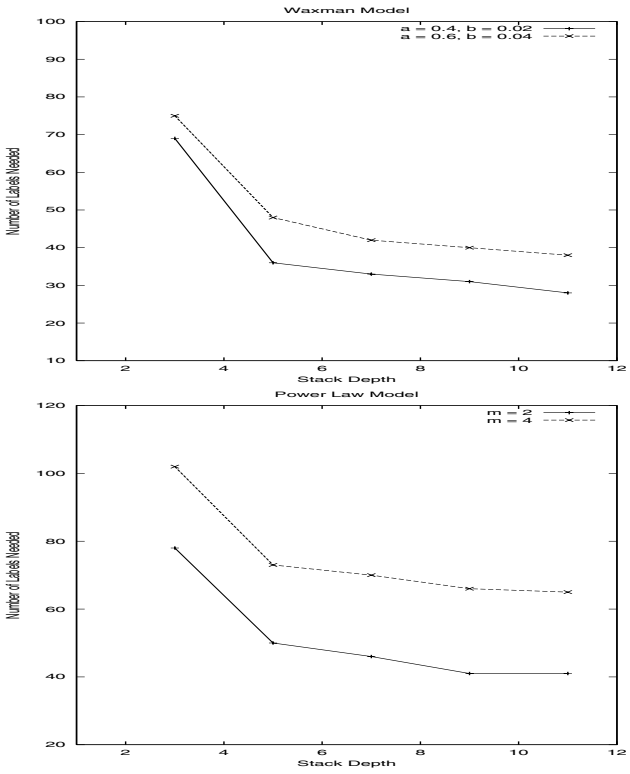


Fig. 6. Effect of stack depth on number of labels.

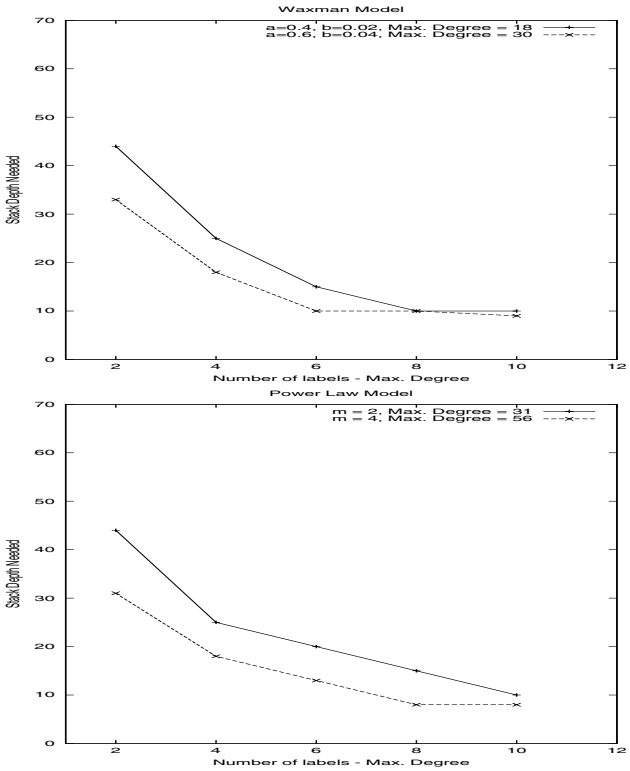


Fig. 7. Effect of number of labels on stack depth.

VII. Concluding Remarks

In this paper, we conducted a comprehensive theoretical study of the label size versus stack depth trade-off for MPLS routing protocols on lines and trees. Specifically, we developed routing algorithms and proved lower bounds for two basic problems: (1) Fixed Stack Routing: Given a bound on the stack depth, we want to minimize the number of labels used, and (2) Fixed Label Routing: Given a fixed number of labels, we want to minimize the stack depth. The protocols proposed in the paper minimize resource utilization, and are efficient, practical, and simple to implement. Further, our

simulation results indicate that the protocols perform well in practice and enable MPLS routing on large trees with few labels and small stack sizes. Consequently, our protocols have numerous practical applications that include implementation of *multicast trees* [13] and *virtual private networks* [6], [15], [16] using MPLS as the underlying signalling mechanism.

REFERENCES

[1] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and Jose Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

[2] Daniel O. Awduche. MPLS and traffic engineering in IP networks. *IEEE Communications Magazine*, December 1999.

[3] Barun Chandra, Gautam Das, Giri Narasimhan, and Jose Soares. New sparseness results on graph spanners. *International Journal of Computational Geometry & Applications*, 5(1-2):125–144, 1995.

[4] CISCO MPLS web page. `http://www.cisco.com/warp/public/732/Tech/mpls/`.

[5] Lenore Cowen. Compact routing with minimum stretch. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Disc= rete Algorithms*, pages 255–260, 1999.

[6] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merwe. A flexible model for resource management in virtual private networks. In *Proceedings ACM SIGCOMM*, 1998.

[7] Bruce Davie and Yakov Rekhter. *MPLS: Technology and Applications*. Morgan Kaufmann Publishers, 2000.

[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In Proceedings ACM SIGCOMM, 1999.

[9] Greg N. Frederickson and Ravi Janardan. Designing networks with compact routing tables. *Algorithmica*, 3:171–190, 1988.

[10] Greg N. Frederickson and Ravi Janardan. Efficient message routing in planar networks. *SIAM Journal on Computing*, 18(4):843–857, 1989.

[11] Cyril Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News - Distributed Computing Column*, 32(1):36–52, March 2001.

[12] Cyril Gavoille, David Peleg, Stephane Perennes, and Ran Raz. Distance labeling in graphs. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 210–219, 2001.

[13] S. Keshav. An engineering approach to networking. Addison-Wesley Professional Computing Series, 1997.

[14] A. Gupta, A. Kumar and R. Rastogi. Routing Issues in MPLS (Or, How to Travel with a Pez Dispenser). In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science* (FOCS), 2001.

[15] A. Gupta, J. Kleinberg, A. Kumar, R. Rastogi and B. Yener. Provisioning a virtual private network: A network design problem for multicommodity flow. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, 2001.

[16] A. Kumar, R. Rastogi, A. Silberschatz and B. Yener. Algorithms for provisioning virtual private networks in the hose model. In *Proceedings ACM SIGCOMM*, 2001.

[17] MPLS Charter. `http://www.ietf.org/html.charters/mpls-charter.html`.

[18] Nathan Linial, Avner Magen, and Michael Saks. Trees and Euclidean metrics. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 169–177, 1998.

[19] Jiří Matoušek. On embedding trees into uniformly convex Banach spaces. *Israel Journal of Mathematics*, 114:221–237, 1999.

[20] Nortel MPLS web page. `http://www.nortelnetworks.com/corporate/technology/mpls/index.html`.

[21] David Peleg. Proximity-preserving labeling schemes and thei applications. In *25th Workshop on Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science 1665, pages 30–41, 1999.

[22] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. Assoc. Comput. Mach.*, 36(3):510–530, 1989.

[23] Eric C. Rosen, Dan Tappan, Yakov Rekhter, Guy Federkow, Dino Farinacci, Tony Li, and Alex Conta. MPLS label stack encoding (RFC 3032). `http://www.ietf.org/rfc/rfc3032.txt`, January 2001.

[24] Eric C. Rosen, Arun Viswanathan, and Ross Callon. MultiProtocol Label Switching architecture (RFC 3031). `http://www.ietf.org/rfc/rfc3031.txt`, January 2001.

[25] B.M. Waxman. "Routing of Multipoint Connections". *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, December 1988.

[26] Peter Winkler. Proof of the squashed cube conjecture. *Combinatorica*, 3(1):135–139, 1983.