Traveling with a Pez* Dispenser

(Or, Routing Issues in MPLS)

†

Anupam Gupta‡ Amit Kumar§ Rajeev Rastogi¶

Abstract

In most conventional network routing protocols, a packet makes its way from source to destination essentially thus: when a router gets the packet, it analyses the packet header and decides the next hop for it, and sends on the packet. These decisions are usually based on the router's analysis of the contents of the packet header, particularly the destination address, and *each* router has to extract out the information relevant to it from the (much longer) packet header. An alternative proposed to this routing model is *MultiProtocol Label Switching* or MPLS [8, 14]. 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 [25, 24]. 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 on the packet. This scheme has a number of advantages, over conventional routing protocols; the primary ones being a reduced amount of header analysis allowing faster switching, and also traffic engineering.

Despite the fact that MPLS is becoming widespread on the Internet, we know essentially nothing at a theoretical level about the performance one can achieve with it, and about the intrinsic trade-offs in its use of resources. In this paper, we initiate a theoretical study of the protocol and give routing algorithms and lower bounds in a variety of situations. We first study the routing problem on the line, where it is already non-trivial; we give routing protocols that are within constants of the information-theoretic bound for this case. We then extend our results for paths to trees, and thence onto more general graphs. The technique used for this last step is that of finding a *tree cover* of a graph, i.e., a small set of subtrees of the graph such that for each pair of vertices, one of the trees contains an (almost-)shortest path between them. Our results showing tree covers of logarithmic size for for planar graphs and graphs with bounded separators seem to be of independent interest as well.

*Registered trademark of Pez Candy, Inc.

†A preliminary abstract of this work appeared in the Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science [13].

‡Lucent Bell Labs, 600 Mountain Avenue, Murray Hill NJ 07974. Email: anupamg@research.bell-labs.com. Part of this research was done when this author was visiting Cornell University.

§Department of Computer Science, Cornell University, Ithaca NY 14853. Email: amitk@cs.cornell.edu. Supported in part by Lucent Bell Labs and the ONR Young Investigator Award of Jon Kleinberg.

¶Lucent Bell Labs, 600 Mountain Avenue, Murray Hill NJ 07974. Email: rastogi@research.bell-labs.com.

1 Introduction

In most conventional network routing protocols, a packet makes its way from source to destination in essentially the following way. When a router gets the packet, it analyses the packet header and decides the next hop for it. These decisions are made locally and independently of other routers, based solely on the identity of the incoming edge, and 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.

An alternative proposed to this routing model by the IETF is called *MultiProtocol Label Switching* or MPLS [8, 14]. 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 [25, 24]. 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.*) 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. This allows us to replace routers by simpler fast switches which are capable of doing label lookup and replacement. Furthermore, 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 channels 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, it is also much easier to do *traffic engineering* or network control using MPLS than 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 [6, 19].

Despite the fact that MPLS is becoming widespread on the Internet, we know essentially nothing at a theoretical level about the performance one can achieve with it, and about the intrinsic tradeoffs in its use of resources. For instance, a pertinent question is the following: 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 want small-sized labels, since bandwidth reservation in networks is often done by creating a (virtual) channel for each label. A small number of labels ensures that the traffic is not split too much, which usually implies a better bandwidth utilization. Furthermore, having a small label space makes the forwarding procedures simple and hence faster. On the other hand, we want a small stack size as well, so as to keep the space requirements in the headers small. Obviously, these goals oppose each other, and their tradeoffs seem non-trivial. Previous papers on routing do not address such questions, and it is not clear whether the information theoretic bounds are close to the truth.

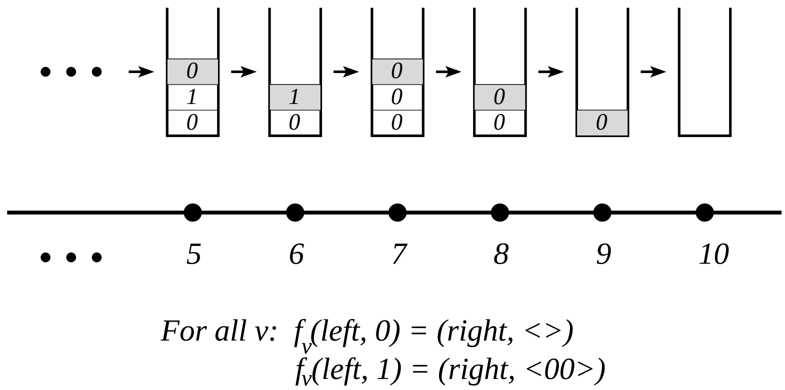

Figure 1: An example of MPLS routing.

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). As an example, consider the following question: if we are given a constant-degree graph, it is not clear whether shortest-path routing is at all possible when each router looks at only one label of length $O(\log \log n)$ bits, instead of having access to the entire network header of $O(\log n)$ bits. Again, this is clearly a question that needs to be addressed.

In this paper, we initiate a theoretical study of the protocol, and give routing algorithms and lower bounds in a variety of situations. We first study the routing problems on the line. We then build up our results from paths through trees to more general graphs. The basic technique to go to general graphs is that of finding a *tree cover*, which is a small set of subtrees of the graph such that for each pair of vertices, one of the trees contains an (almost-)shortest path between them. The concept of tree covers is interesting in its own right.

The Model: Before we give our results, let us formalize the model. Each *packet* carries a *stack* S 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. If the protocol does not depend of the node on which it is running, the protocol is called *uniform*. When a packet reaches a router v on edge $e = \{u, v\}$, the router *pops* the top $t(S)$ 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 $f_v : E_v \times \Sigma \to (E_v \times \Sigma^*)$, where E_v is the set of edges incident to v. If $f_v(e, t(S)) = (e', \sigma)$, the router *pushes* the string σ on the stack, and then sends the packet along edge e'.

A toy example is given in Figure 1 with $\Sigma = \{0, 1\}$. All functions f_v are the same; and only the relevant subset of the actions is shown here. The packet in the example is destined for the vertex labeled 10. Each node bases its decision on the top (shaded) label and the incoming edge.

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 $O(s)$.

Our Results: As a first step, we study routing on the path P_n, where we show a large gap between uniform and non-uniform protocols. We show that uniform protocols on the line with L

labels require $s = \Theta(Ln^{1/L})$. However, we give a non-uniform protocol using L labels requiring stack depth $O(\log_L n)$ only. Note that this 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 the so-called *caterpillar decomposition* [18, 15] of trees into paths to get a $(\Delta + k, kn^{1/k} \log n)$ uniform protocol, and a $(\Delta + k, \frac{\log^2 n}{\log k})$ non-uniform protocol. In the case of uniform protocols, we prove an almost matching lower bound when k is $O(\log n)$. (Note 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 non-uniform protocol to get a $(\Delta + \log\log n, \log n)$ protocol as well.

Finally, we turn to the case of general graphs. Here, we use the protocols for trees as our basic tools. We define a *tree cover* of a graph, which is a small set of subtrees of the graph such that for each pair of vertices, one of the trees contains an (almost-)shortest path between them. (See Definition 4.1 for a formal definition.) If we have a graph with a tree cover with t trees, we can run the tree routing algorithm on the appropriate tree. Note that we lose just a factor of t in the number of labels by this idea. Unfortunately, it can be shown that general graphs do not have $(\log n)$-sized tree covers unless the trees are allowed to stretch distances by $\Omega(\log n)$.

Since a non-constant stretch is inadmissible in our applications, we look at special classes of graphs, and as our first result, show that graph families with $r(n)$-sized balanced vertex-separators have $O(r(n) \log n)$ sized tree covers with no stretch. This result also gives $O(\sqrt{n})$-sized tree covers for planar graphs, which we show tight by exhibiting a simple length-assignment to the edges of the n-vertex grid. However, we then go on to show that allowing a small stretch (of 3) improves matters considerably: we can find a $O(\log n)$ sized tree cover for all planar graphs. The proof of this fact uses the Lipton Tarjan planar separator theorem [16] in a novel way, which we feel may have other implications.

As the above discussion indicates, our algorithms are extremely modular in nature, and hence improvements in routing strategies for (say) the path will result in improvements for trees and graphs. Furthermore, though we have made no significant efforts to optimize constants, the constants involved are small, and hence the algorithms can be implemented in practice.

Previous Work: Distributed packet routing problems in networks has been widely studied, e.g., see [9, 10, 23, 22, 7], 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. For example, in traditional routing on planar networks, the best result known for minimizing the total memory (i.e., summed over all the routers) is $\tilde{O}(n^{4/3})$. In our case, setting up the stack requires more memory, but for just forwarding the packet, the total memory required is $\tilde{O}(n)$. Furthermore, many previous results giving small storage allow the vertices to be labeled by the algorithm, whereas we make no assumptions on the vertex names.

There has also been substantial work on finding sparse *spanners* of graphs [1, 5]. 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 focussed on the problem of distance labeling of graphs [27, 20, 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). The techniques used in both this problem as well as ours are similar, often involving finding good separators of graphs; however, the scope of the problems are quite different. Indeed, distance labeling precludes any knowledge of the global structure of the graph, and hence the label sizes are usually in the range of $\Theta(\log n)$. In the case of MPLS, however, the graph structure is known, and the challenge lies in local routing algorithms that look at a much smaller set of bits, i.e., $O(\log \log n)$, or even a constant number of bits, to decide where the packet should be sent. This makes MPLS routing for even the path to be non-trivial, whereas the distance labeling problem is quite simple for this case. Furthermore, merely having an small stretch distance labeling scheme does not appear to give us routing protocols in which packets travel along near-shortest paths.

However, we note that some of our MPLS results can be used to improve known results on distance labelings. In the case of planar graphs, we can use our ideas to get a stretch-3 distance labelings of size $O(\log^2 n)$ for planar graphs. Previously, no sub-polynomial labeling schemes were known for planar graphs (even with constant distortion) [12].

A recent paper of Thorup and Zwick [26] gives constructions of a slightly different variety of tree covers. Though their definitions differ from ours, they can also be used for MPLS routing. Their results imply that for general graphs, there exist tree covers of size $\tilde{O}(n^{1/k})$ with stretch $O(k)$. This gives an MPLS routing scheme with $\tilde{O}(n^{1/k})$ labels, poly-logarithmic stack depth and stretch $O(k)$. We, however, concentrate on cases where it is possible to get poly-logarithmic stack depth and labels, and constant stretch.

2 Routing on the line

In this section, we give shortest-path routing schemes for the path graph P_n. This is the basic building block which we shall use to route on trees in the next section. We give two routing strategies, depending on whether nodes are allowed to have different routing protocols or not. We show that if the routers must run the same protocol, then the stack depth goes as $\Theta(Ln^{1/L})$; however, if they are allowed to use the information of their own position, then a very simple strategy allows us to have $s = O(\log_L n)$, which is within constants of the best possible.

2.1 Uniform protocols

In this case, we assume that each router must run the same protocol. To achieve the upper bound of $O(Ln^{1/L})$, we have the following simple strategy: nothing is pushed onto the stack when a 1 is seen, and seeing an $i > 1$ causes $n^{1/L}$ copies of $(i-1)$ to be pushed onto the stack. It is easy to see that the total stack depth need only be $Ln^{1/L}$. The following theorem, whose proof is in the appendix, shows that the construction is tight up to constants.

Theorem 2.1 *Any uniform routing protocol must require a stack depth of $\Omega(Ln^{1/L})$. This bound can be achieved by the scheme outlined above.*

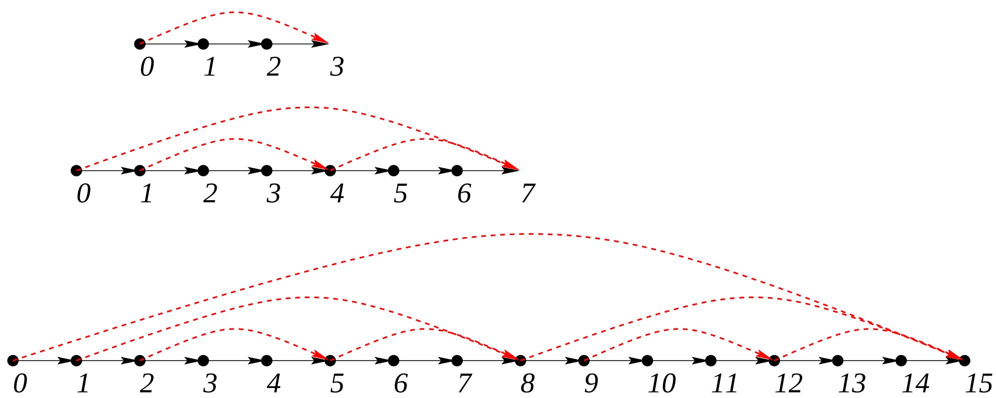

Figure 2: Specifying the routing protocol for the path P_{16}.

2.2 Non-uniform protocols

Interestingly, the case for non-uniform protocols, where each vertex can run a different protocol, the relationship between s and L is much closer to the information-theoretic bound. We will consider the case when $L = 2$: in this case, it is easy to see that the stack depth must be $\Omega(\log n)$ for us to encode n distinct addresses. 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 2, 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:

Lemma 2.2 *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'.*

Proof: Suppose the shortest path P' from u' to v does not contain v'. Let $e = (w, w')$ be an edge in P' such that $w < v' < w'$ (clearly, such an edge must exist). We claim that P must contain w. If not, since $u < w < v$, P must contain an edge $e' = (x, x')$ such that $x < w < x'$. Further, $x' < v'$ since P contains v'. But now e and e' violate the nesting property, and hence P contains w. Also, the portion of P from w to v must be the shortest path from w to v. Since we know that P' contains w, replacing the portion of P' after w by that of P, we again get a shortest path P'' from u' to v which contains v'. This proves the lemma. ∎

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 2.2 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 2.3 *There is a non-uniform 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 2.4 *There is a non-uniform protocol for routing on the n-vertex path which uses L labels and stack depth at most $O(\log_L n)$, which is within a constant factor of the information-theoretic bound.*

3 An algorithm for trees

In this section, we consider the problem of routing on trees. Since we already have developed algorithms for the line that are within constants of the best possible, we first show how to use them to get protocols for trees. We then refine these to get better tradeoffs.

Let the tree be T, and let it be rooted at r. All the algorithms use the so-called caterpillar decomposition of a tree into edge-disjoint paths. The *caterpillar dimension* [18, 15] of a rooted tree T, henceforth denoted by $\kappa(T)$, is defined thus: For a tree with a single vertex, $\kappa(T) = 0$. Else, $\kappa(T) \leq k+1$ if there exist paths $P_1, P_2, \ldots, P_t$ beginning at the root and pairwise edge-disjoint such that each component T_j of $T - E(P_1) - E(P_2) - \ldots - E(P_t)$ has $\kappa(T_j) \leq k$, where $T - E(P_1) - E(P_2) - \ldots - E(P_t)$ denotes the tree T with the edges of the P_i's removed, and the components T_j are rooted at the unique vertex lying on some P_i. The collection of edge-disjoint paths in the above recursive definition form a partition of E, and are called the *caterpillar decomposition* of T. It is simple to see that the unique path between any two vertices of T intersects at most $2\kappa(T)$ of these paths. It can also be shown that $\kappa(T)$ is at most $\log n$ (see, e.g., [18]).

Now, given a pair of vertices to route between, there are $O(\log n)$ paths to travel on, and $O(\log n)$ changes of paths to specify. Hence, if we have a (L, s) routing protocol for the line, we could get a $(\Delta(T) + L, s\kappa(T))$ protocol for the tree. Plugging in the values from the previous section, we get the following theorem. (See the Appendix B for a formal definition of uniform protocols for trees.)

Theorem 3.1 *Given a tree T with maximum degree Δ, there exists a $(\Delta + k, kn^{1/k}\kappa(T))$ uniform routing protocol and a $(\Delta + k, (\log_k n)\ \kappa(T))$ non-uniform routing protocol for T.*

In the Appendix, we also prove the following almost matching lower bound for $k = \log n$.

Theorem 3.2 *There exists a binary tree T such that any uniform routing protocol with $O(\log n)$ labels requires stack depth $\Omega(\left(\frac{\log^2 n}{\log\log n}\right)$.*

Note that for $k = 2$, we have a $(\Delta + 2, \log^2 n)$ non-uniform protocol, and for $k = \log n$ and constant Δ, the worst case guarantees for both these algorithms are approximately $(\log n, \log^2 n)$. The results of the next section show how to get a much better result in the non-uniform case.

3.1 Improved Non-Uniform Protocols

Interestingly enough, we can improve the non-uniform routing algorithm of the previous section, keeping the stack depth at $O(\log n)$, and get a label size of $O(\log\log n)$. Let $k = \lceil \log_2 n \rceil$. We will prove the following lemma by induction on n (where c is the constant in Theorem 2.3):

Lemma 3.3 *We can route a packet from the root r to any node in T by using at most $2\log k + \Delta$ labels, and stack depth at most $6ck$.*

As before, Δ of the labels are used to decide which branch to take when changing paths. For the proof, we shall indicate how the rest of the $2\log k = 2\log\log n$ labels can be used for the rest of the routing.

Proof: The base case follows trivially from Theorem 2.3. To show the inductive step, we use the following fact, which we refer to as the *halving* property for caterpillar decompositions. One can

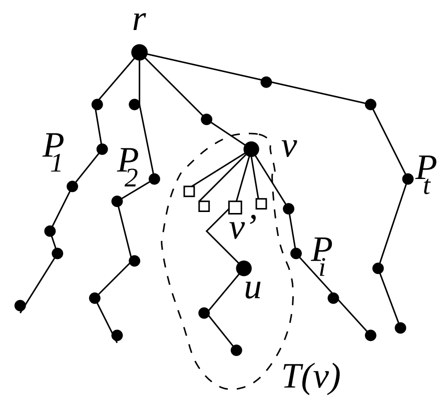

Figure 3: Proof of Lemma 3.3. Vertices marked by squares belong to V'.

find a decomposition of size $O(\log n)$ with the following property: If $P_1, \ldots, P_t$ be all the paths originating at the root r, then for any vertex $v \in P_i$, any connected component of $T - \{v\}$ not containing a node of P_i has at most $\lfloor n/2 \rfloor$ nodes. (The proofs of [18, 15] show such a construction.) We will assume this property of our caterpillar decomposition.

Let us fix a path P_i in the caterpillar containing r. We show how to route a packet from r to any descendant of a node in $P_i - \{r\}$. If we show that the conditions of the lemma hold for this path, the lemma in general follows from the fact that the paths $P_1, \ldots, P_t$ are disjoint except at r.

Consider a vertex $v \in P_i - \{r\}$, and let V' be the children of v which are not in P_i (see Figure 3). Define $T(v)$ rooted at v to be the subtree containing v, the subset of its children V' just defined, and all the descendents of V'. Observe that if $v \neq w \in P_i$, then $T(v)$ and $T(w)$ are disjoint. We define $t(v)$, the *index* of a node v, to be $\lceil \log_2 |T(v)| \rceil$. Let $I(j)$ be the set of nodes in $P_i - \{r\}$ with index j. Note that if $t(v) = j$, then $|T(v)| \geq 2^{t(v)-1}$, and since all these trees $T(v)$ are disjoint, it follows that there are at most 2^{k-j+1} nodes in $I(j)$.

We now form $\log k$ *supergroups*, each supergroup formed by the union of several $I(j)$'s. For each $p = 0, \ldots, \log k$, the supergroup $\mathcal{I}(p)$ is the union of the groups $I(k - 2^{p+1} + 2), \ldots, I(k - 2^p + 1)$. The number of nodes in $\mathcal{I}(p)$ is maximum when all these nodes come from $I(k - 2^{p+1} + 2)$, and so $\mathcal{I}(p)$ contains at most $2^{2^{p+1}+1}$ nodes. We divide the labels L into $\log k$ sets, $L_1, \ldots, L_{\log k}$, with each L_i containing 2 labels. The labels in L_p are used to route from r to nodes only in $\mathcal{I}(p)$. If any node in P_i that does not lie in $\mathcal{I}(p)$ sees a label in L_p on top of the stack, it merely forwards the packet to its child on P_i. Theorem 2.3 now implies that we can use the labels in L_p to route from r to all nodes in $\mathcal{I}(p)$ using a stack depth of at most $c(2^{p+1} + 1)$. Note that this requires only $2 \log k$ labels.

Now consider the case when r wants to send a packet to $u \in T(v)$. Suppose $v \in I(j)$ and $I(j) \in \mathcal{I}(p)$. The top part of the stack contains labels that send the packet from r to v, requiring a stack depth of at most $c(2^{p+1} + 1)$. Let $v' \in V'$ be the child of v such that u is a descendant of v'; the next symbol on the stack is one of the Δ labels that cause v to send the packet to v'. The remaining part of the stack specifies how to route from v' to u. Let T' be the subtree rooted at v', and j' be the smallest integer such that $j' = \lceil \log |T'| \rceil$. Clearly, $j' \leq k - 2^p + 1$; also, the halving property of the caterpillar decomposition implies that $j' \leq k - 1$. By induction, the stack depth needed is at most $6cj'$. Hence the total stack depth needed is at most $2c2^p + c + 1 + 6cj' \leq 2c2^p + c + 2c(k - 2^p + 1) + 1 + 4c(k-1) = 6ck + c + 2c + 1 - 4c \leq 6ck$. This proves the desired result. ■

Though we have been showing how to route packets starting at the root, the same algorithm can actually send packets from any vertex to its descendent with stack depth $O(\log n)$. Note that to send a packet from v to an arbitrary vertex u, if we could first route to the least common ancestor of u and v, we would have reduced the problem to the solved case of routing to a descendent. However, this is fairly simple: to route to $\text{lca}(u, v)$, we note that the packet is always traveling in the upwards direction, and hence this is isomorphic to the problem of routing on a line, which we can do with 2 labels and $O(\log n)$ stack depth using the scheme of Theorem 2.3. This concludes the proof of the following theorem:

Theorem 3.4 *There exists a $(\Delta + \log\log n, \log n)$ non-uniform routing protocol for trees.*

4 Covering graphs by trees

There are several problems to extending the above scheme to route in arbitrary graphs: the shortest paths between vertices are not unique, they intersect in non-trivial ways, and hence it is difficult to come up with a useful notion of a path decomposition. However, if we could find a set of k subtrees, such that for each pair of vertices, there was a tree in this set that maintained the shortest path distance between them, we could use this for routing. This would just involve specifying which of these trees we were routing on, which would cause the number of labels to increase by a factor of k. Of course, we could relax the distance condition to allow distances to be stretched by a small factor even in the best tree. Motivated by this, we define a *tree cover* of a graph:

Definition 4.1 *Given a graph $G = (V, E)$, a* tree cover *(with* stretch *D) of G is a family $\mathcal{F}$ of subtrees $\{T_1, T_2, \ldots, T_k\}$ of G such that for every $u, v \in V$, there is a tree $T_i \in \mathcal{F}$ such that $d_{T_i}(u, v) \leq D\, d_G(u, v)$.*

The following theorem follows immediately from the discussion above.

Theorem 4.2 *Let there be an (L, s) protocol for routing on trees. Let $\mathcal{F}$ be a tree cover of G with stretch D. Then, there is an $(L|\mathcal{F}|, s)$ protocol for G. This protocol has stretch D, i.e., given any pair of vertices $u, v \in V$, this protocol routes from u to v on a path which has length at most D times the shortest path between u and v.*

Note that, since each tree is a subtree of G, $d_G(u, v) \leq d_{T_i}(u, v)$. When $D = 1$, we often say that there is no stretch; furthermore, in this case, we will often omit mentioning the stretch.

Tree covers have been previously defined and used for conventional routing applications in [4, 3] (see also [21]). Note that this definition of tree covers is slightly different from previous definitions, since it does not place a restriction on the number of trees in which a vertex appears, but instead places a uniform restriction on the number of trees in the family.

Of course, it is easy to see that the size of a tree cover may be large: if we require a stretch 1 tree cover for the complete graph K_n, the union of the T_i must cover every edge, and hence $\Omega(n)$ trees are required. By the trick of replacing the edges incident to a vertex by a (weighted) binary tree, it can be seen that a lower bound of $\Omega(n)$ holds even for degree-3 graphs.

As for lower bounds for covers with stretch: there are explicit constructions of graphs with $\Omega(n^{1+4/(3g-6)})$ edges which have girth g [17]. For these graphs, if we want a stretch less than g, the union of our T_i must also contain every edge of such a graphs. Hence we can get a lower bound of $\Omega(n^{4/(3D-6)})$ for covers of stretch $D-1$. A case of particular interest is when $D=4$, for complete bipartite graphs show that stretch-3 covers may require $\Omega(n)$ trees. (A trick similar to that alluded to above shows a similar result for bounded-degree graphs.)

In view of these general negative results, the question of interest is to find families of graphs for which we can find small tree covers. In this section, we study the problem of finding small tree covers for families of graphs with small sized vertex separators. For example, for planar graphs, we know that separators of size $O(\sqrt{n})$ exist, while bounded tree-width graphs have constant-sized separators.

4.1 Unit weighted grid

Before we present the constructions for small-separator and planar graphs, let us give the following simple result :

Theorem 4.3 *The unit-weighted grid has tree covers of size $O(\log n)$.*

Proof: Let the vertices be referred to in the usual way as (i,j) for $1 \leq i,j \leq \sqrt{n}$. Now consider the tree T defined by the union of the paths $P = \{(\sqrt{n}/2, j) | 1 \leq j \leq \sqrt{n})\}$ and $P_j = \{(i,j) | 1 \leq i \leq \sqrt{n})\}$ for all j. It is easy to check that for any two vertices that lie in different halves of the grid defined by the vertical path, the shortest path lies in T.

Now to find paths between vertices which lie in the same half, we recurse on both the smaller grids. (Note that a similar construction would work for rectangular grids as well, and so the recursion is well-defined.) Inductively, we get two families of at most $t = \log(n/2)$ forests, one for each part; let them be $F'_1, F'_2, \ldots, F'_t$, and $F''_1, F''_2, \ldots, F''_t$ respectively. Note that defining $F_i = F'_i \cup F''_i$ gives us t forests of the original graph G (since F'_i and F''_i are vertex disjoint), and hence adding the tree T to these $\log n - 1$ forests gives us the desired $\log n$-sized non-stretch tree cover. Note that it was not important that two components were created by the separator; the technique would have worked with many components as well. ■

Note that it is possible to design a better routing scheme for grids. Given two vertices $u = (i,j)$ and $v = (i',j')$, there is a shortest path between them that goes from u to $w = (i,j')$ and then from w to v. The protocol specifies how much distance to go without changing the first coordinate, and then how far to go without changing the second coordinate.

4.2 Graphs with Small Separators

Using some of the ideas from the previous section, we give a tree cover of size $O(r(n)\log n)$ for families of graphs which admit $r(n)$-sized hierarchical separators. (I.e., these are graphs which can be separated into pieces of size at most $2n/3$ by removing at most $r(n)$ vertices, and any connected component G_i thus obtained has a separator of size $r(|G_i|)$, and so on.) It is well-known that for planar graphs, $r(n) = O(\sqrt{n})$, and for treewidth-k graphs, $r(n) = k$. (We shall make the reasonable assumption that $r(n)$ is monotonically increasing.)

The idea is very simple: we first find a separator S of G having size at most $r(n)$. For each of the vertices $s \in S$, we take the shortest-path tree T_s rooted at S.

Lemma 4.4 *For any pair of vertices $u, v \in T$ for which the shortest path P connecting them intersects S, there is a tree T_s which contains the shortest path between u and v.*

Proof: For any such pair of vertices u and v, let $P \cap S$ contain the vertex s. Then P must be the concatenation of the shortest path from s to u, and that from s to v. But then both these paths lie in T_s, and hence the claim is proved. (We are implicitly assuming in this proof that there are unique shortest paths; this assumption is purely for convenience and can be discharged in the usual ways.) ■

We are now left with $G - S$, which has components of size at most $2n/3$, and we just have to construct trees to maintain distances between vertices that lie within these components. Recursively, each of these can be done by a family of size $r(2n/3)\log_{3/2}(2n/3) \leq r(n)(\log_{3/2} n - 1)$, and by pairing them up and adding the set of $r(n)$ trees created at this level, we get the claimed cover of $r(n)\log_{3/2} n$ subtrees.

Note that for planar graphs, plugging in $r(n) = O(\sqrt{n})$ and being slightly more careful in the above analysis gives us a tree cover of size $O(\sqrt{n})$.

4.3 Lower bounds

In this section, we show that the result of the previous section for planar graphs is existentially tight.

Theorem 4.5 *There exist length assignments to the edges of the grid so that any tree cover (with stretch 1) is of size $\Omega(\sqrt{n})$.*

Proof: Let $G = (V, E)$ be an $n = t \times t$ square grid, where the vertices are (i, j), $1 \leq i, j \leq t$ in the obvious manner. Let ϵ be a small enough positive number ($\epsilon = \frac{1}{n}$ will suffice). Let e be an edge joining vertices (i, j) and (i', j'). Then let us define c_e, the length of edge e to be $1 + \frac{1}{n}(\min(i, i') + (1 + \epsilon)\min(j, j'))$.

The basic intuition behind assigning these edge-lengths c_e is just symmetry breaking, which is formalized in the following lemma. (We defer the proof of this fact to the appendix.)

Lemma 4.6 *Given any two vertices in G, there is a unique shortest path between them. Furthermore, this shortest path has at most one bend.*

Let T be a spanning tree of G, and let $\mathcal{S}_T$ be the set of pairs of vertices (u, v) in V such that T contains a shortest path between u and v (with respect to the edge costs c_e). In Appendix C, we show the following key lemma:

Lemma 4.7 *For any spanning tree of the above grid, $|\mathcal{S}_T|$ is $O(t^3)$.*

Since there are $\Omega(t^4)$ pairs of vertices, this shows that we require $\Omega(t) = \Omega(\sqrt{n})$ trees in the cover, completing the proof. ■

5 Tree Covers for Planar Graphs

In this section, we will show that all planar graphs have stretch-3 tree covers of size $O(\log n)$. This is in sharp contrast to the results of the previous sections that planar graphs do not have $o(\sqrt{n})$ sized covers in general if no stretch is allowed, and that general bounded degree graphs do not have $o(n^{2/3D})$ sized stretch-D tree covers.

5.1 Isometric Separators

We can refine the ideas in Section 4.2 to get a $O(\log n)$ sized family for all planar graphs. Let us first make a few definitions: given a graph $G = (V, E)$, a *k-part isometric separator* is a family $\mathcal{S}$ of k subtrees $S_1 = (V_1, E_1), \ldots, S_k = (V_k, E_k)$ of G such that

1. $S = \cup_i V_i$ is a 1/3-2/3 separator of G.

2. For each i and each pair of vertices $u, v \in S_i$, $d_{S_i}(u, v) = d_G(u, v)$. I.e., the each of the subtrees S_i contain the shortest paths between their constituent vertices, and hence are isometric to the restriction of G on V_i.

Note that we do not care about the total number of vertices in S; just the number of isometric subtrees.

For instance, any graph having a 1/3-2/3 separator of size $r(n)$ has a trivial $r(n)$-part isometric separator, where each S_i contains just a single vertex. However, if we look at the proof of the planar separator theorem [16], it can be inferred that any planar graph has a 2-part isometric separator. Now an extension of the ideas in the previous sections shows the following theorem:

Theorem 5.1 *For any graph $G = (V, E)$ with $r(n)$-part isometric separators, there exists a tree cover with stretch 3 having $O(r(n) \log n)$ trees.*

Proof: The following algorithm is very similar in spirit to that in Section 4.2. For each of the trees S_i, we contract the vertices of S_i and construct a shortest-path tree in the resulting graph, and then expand back the tree S_i. The resulting tree is call T_i. Note that T_i contains S_i, and the union of the shortest paths from every other vertex in $V - V_i$ to the subtree S_i. This gives us $r(n)$ trees, and we now recurse on the two parts in a by now familiar fashion. It is clear that this process gives us at most $r(n) \log_{3/2} n$ trees.

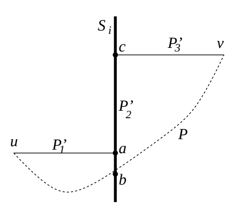

Figure 4: Proof of small stretch in theorem 5.1

What remains to be shown is that, for each pair of vertices, there is a tree which maintains distances between them to within a factor of 3. The proof mimics that of Theorem 4.4. Consider a pair of vertices u, v for which the shortest path P between u and v intersects some S_i (at point

b, say). The path P' between u and v in T_i can be divided into three sections P'_1, P'_2, P'_3, where P'_1 is the shortest path from u to S_i, P'_3 is the shortest path from v to S_i, and P'_2 is the unique path in S_i connecting the points a and c at which P'_1 and P'_3 meet S_i. (See Figure 4 for an illustration.)

For nodes x, y, let $[x, y]$ denote the shortest-path between x and y in G. Now since $[u, a]$ and $[v, b]$ are the shortest paths to S_i, $d_G(u, a) \leq d_G(u, b)$, and $d_G(v, c) \leq d_G(v, b)$. Furthermore, by the fact that $[a, c]$ is the shortest path, $d_G(a, c) \leq d_G(a, u) + d_G(u, v) + d_G(v, c)$. But the length of the path

$$\begin{aligned} d_{T_i}(u, v) &= d_G(u, a) + d_G(a, c) + d_G(c, v) \\ &\leq d_G(u, a) + (d_G(a, u) + d_G(u, v) + d_G(v, c)) + d_G(c, v) \\ &\leq 2(d_G(u, b) + d_G(v, b)) + d_G(u, v) = 3d_G(u, v), \end{aligned}$$

which proves the claim. ∎

Now using the fact that planar graphs have 2-part isometric separators gives us the following theorem:

Theorem 5.2 *There exists a stretch-3 tree cover of size $O(\log n)$ for all planar graphs*

Corollary 5.3 *Given an (L, s) routing scheme for trees, there is an $(L \log n, s)$ routing scheme for planar graphs. This routing protocol has stretch at most 3.*

Proving such a result for broader classes of graphs still remains open. One of the problems with extending the above approach is that isometric separators are not known for many classes of graphs, even for graphs with small sized separators.

5.2 An Application To Small Distance Labelings

In this section, we give another application of isometric separators. A stretch-D *distance labeling* scheme is a way of assigning a *label* $l(v)$ to each vertex v, and specifying a scheme f such that given a graph G, $1 \leq f(l(u), l(v))/d_G(u, v) \leq D$ for all pairs of vertices $u, v \in G$. This has been studied in [27, 20, 12].

Theorem 5.4 *For any planar graph $G = (V, E)$ with diameter $diam(G)$, a stretch-3 distance labeling scheme with labels of size $O(\log^2 n)$ bits exists.*

This result should be contrasted with the result of Gavoille et al. [12] that $\Omega(n^{1/3})$ bits are required when no stretch is allowed. We should note that it is possible to get a quick-and-dirty $O(\log^3 n)$ bit result, by taking the $O(\log n)$ tree cover of Theorem 5.2, and using the distance labeling scheme of Peleg [20] to embed each tree with $O(\log^2 n)$ bits.

Proof of Theorem 5.4: For each vertex, we generate $O(\log n)$ coordinates thus: we look at 2-part isometric separator S_0 of G, which consists of 2 shortest paths P_0 and P'_0, and let a_0 and a'_0 be an endpoint of each of these paths. We will define 2 coordinates for each path. For P_0, the first coordinate records the distance of v from P_0, and the second records the distance of v_0, the closest vertex on P_0 from v. Two coordinates are similarly defined for P'_0. After this, we look at the graph obtained by removing S_0, and record the connected component in which v lies in a fifth coordinate (where we have number the components by some consistent canonical order). We now recurse on this component containing v. Note that if v was in the separator, the rest of the label would have 0's.

For the decoding function $f(u,v)$, we look at the first level in which the two vertices lie in different components. For each of the recursive levels till that point, and for each pair of coordinates corresponding to either shortest-path at that level, we do the following: we add the distance of u and v from the the path, and to this we add the absolute value of the difference of their distances from the chosen endpoint. Finally, we take the minimum among all these values. Using an argument similar to the one used in Theorem 5.2, it is not difficult to show that this minimum is within 3 of the distance between u and v. ■

Acknowledgments

Many thanks to Jon Kleinberg, Bruce Shepherd, Éva Tardos, Peter Winkler, and Francis Zane for discussions and helpful comments. Thanks to Peter also for suggesting the title of the paper.

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] Baruch Awerbuch, Shay Kutten, and David Peleg. On buffer-economical store-and-forward deadlock prevention. In *Proceedings of the IEEE INFOCOM*, pages 410–414, 1991.

[4] David Peleg Baruch Awerbuch. Routing with polynomial communication-space trade-off. *SIAM Journal of Discrete Mathematics*, 5(2):151–162, 1992.

[5] 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.

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

[7] Lenore Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38(1):170–183, 2001. (Preliminary version in *10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 255–260, 1999).

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

[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] Anupam Gupta, Amit Kumar, and Rajeev Rastogi. Routing issues in MPLS (or how to travel with a Pez dispenser). In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 148–157, 2001.

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

[15] 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.

[16] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.

[17] G. A. Margulis. Explicit group-theoretic constructions of combinatorial schemes and their applications in the construction of expanders and concentrators. *Problems in Information Transmission*, 24(1):51–60, 1988.

[18] Jiří Matoušek. On embedding trees into uniformly convex Banach spaces. *Israel Journal of Mathematics*, 114:221–237, 1999. (Czech version in : *Lipschitz distance of metric spaces*, C.Sc. degree thesis, Charles University, 1990).

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

[20] 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.

[21] David Peleg. *Distributed computing*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. A locality-sensitive approach.

[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] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 43–52, 1998.

[24] 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.

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

[26] Mikkel Thorup and Uri Zwick. Approximate distance oracles. In *Proceedings of the 33nd Annual ACM Symposium on Theory of Computing*, 2001. To appear.

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

A Proofs of Section 2

Proof of Theorem 2.1: In the following discussion, let the labels be given by the integers $\{1, 2, \ldots, L)$. Consider a graph with the labels as vertices, and draw an edge from j to i if seeing label i causes j (among others) to be pushed on the stack. Note that any label that lies on a directed cycle is not useful, since the stack can never empty if this label reaches the top of the stack. Hence, let us look at the set of vertices that do not lie on cycles: they form a DAG.

Let us look at a topological sort of this DAG, which (say) places the labels ascending order. Then each label i just corresponds to placing some specific number of labels $1, \ldots, i-1$ on the stack, and hence the ordering of the labels on the stack does not make a difference. Let k_i be the number of copies of label i on the stack; hence $k_1 + k_2 + \cdots + k_L \leq s$. Since, the ordering of these labels does not matter, it follows that the number of solutions to this equation, $\binom{s+L}{L}$, must be at least n. Hence $s = \Omega(Ln^{1/L})$, proving that the above strategy was optimal up to constants. ■

B Proofs of Section 3

B.1 A note on uniform protocols for trees

Let us formally define a uniform protocol on a tree. Clearly, we cannot expect each vertex to behave identically on each label (as on the line), because different vertices may have different degrees.

We assume that there are Δ special labels, called L_Δ, which are used only for going a distance of one hop from a vertex, essentially by specifying which of the edges going out of it should be taken. Let L be the set of other labels. For each edge $e = \{u, v\}$, the vertex v specifies another edge $e' = \{v, w\}$, such that any packet arriving at v on edge e having a label from L on top of the stack is forwarded along e' only. Hence each vertex associates an *exit* edge with each edge e. The action of a vertex when it sees a label $l \in L$ on top of the stack is identical: it places an identical set of labels on top of the stack and sends the packet along the appropriate exit edge. This is the sense in which the protocol is uniform.

B.2 Uniform lower bounds for trees

Proof of Theorem 3.2: We show that any uniform protocol running on a tree $T = (V, E)$ using only $O(\Delta + \log n)$ labels must use $\Omega\left(\frac{\log^2 n}{\log\log n}\right)$ stack depth. Our lower bound example will be a binary tree. Let $T = (V, E)$ be any binary tree. It is not difficult to show that in a binary tree, L_Δ needs to have size 1 only. So, let $L_\Delta = \{l_\Delta\}$.

Given two nodes $u, v \in T$, let $S[u, v]$ denote the stack depth needed to route a packet from u to v. Given a label l, define $S_l[u, v]$ as the stack depth needed to route a packet from u to v such that when the packet reaches v, the top of the stack contains l. If we want to specify a protocol $\mathcal{P}$ for routing, then we use the terms $S_l[u, v](\mathcal{P})$ and $S[u, v](\mathcal{P})$. The following lemma follows from the definition of a uniform protocol.

Lemma B.1 *Let $v \in T$ be a node of degree 3 and let C_1, C_2, C_3 be the components of $T - \{v\}$. Let v_i be the neighbor of v in C_i. Then there exists a $j \in \{2,3\}$ such that given any $x_1 \in C_1$ and $x_j \in C_j$, $S[x_1, x_j] \geq S_{l_\Delta}[x_1, v] + S[v, x_j] - 1$.*

Proof: Let the neighbors of v be v_1, v_2, v_3, where $v_i \in C_i$. Consider the edge $e = \{v_1, v\}$. Suppose v specifies the exit edge for e containing a label in L to be the edge $\{v, v_2\}$. Now if we want to send a packet from x_1 to x_3, it must contain l_Δ on top of stack when it reaches v. Hence the part of this stack which takes the packet from x_1 to v contributes to $S_{l_\Delta}[x_1, v]$. The part of the stack below l_Δ can actually route from v_1 to x_3. Adding l_Δ on top of it gives a routing scheme from v to x_3. This proves the lemma. ∎

Given two vertices u, v in T, we say that they are connected by a *straight path* if all the internal vertices in the unique path connecting u and v have degree 2. Note that the total number of labels is fixed to be $O(\log n)$. Fix a uniform routing protocol $\mathcal{P}$ on T such that there does not exist another protocol $\mathcal{P}'$ with the following property: for every pair of vertices u, v and label l, $S_l[u,v](\mathcal{P}') \leq S_l[u,v](\mathcal{P})$, $S[u,v](\mathcal{P}') \leq S[u,v](\mathcal{P})$ and there is a pair u, v and label l such that $S_l[u,v](\mathcal{P}') < S_l[u,v](\mathcal{P})$.

Lemma B.2 *Let T contain a straight path of length n' joining vertices u and v. There exists an x, $n'/2 \leq x \leq n'$, such that if u', v' are any two vertices in T connected by a straight path of length x, then $S_{l_\Delta}[u', v']$ is $\Omega(\log n' / \log\log n')$.*

Proof: Let P be the path joining u and v. Let V' be the vertices in P whose distance from u is between $n'/2$ and n'. We claim that there is a vertex $w \in V'$ such that $S_{l_\Delta}[u, w]$ is $s' = \Omega(\log n' / \log\log n)$. Indeed, a simple information theoretic argument, and the fact that we have only $O(\log n)$ labels implies this fact. Let x be the distance of u from w.

Suppose u' and v' are two vertices such that there is a straight path joining them of length x. Suppose $S_{l_\Delta}[u', v'] < s'$. Then the uniformity of $\mathcal{P}$ implies that keeping other things the same, we can make $S_{l_\Delta}[u, w] < s'$. But this contradicts the definition of the protocol $\mathcal{P}$, and proves the lemma. ∎

Our lower bound instance T will contain a disjoint family of trees. Since we will route within these trees and not between them, it suffices to prove a lower bound in this case. Given a number x, let T_x denote the complete binary tree of depth $1/6 \log n$ and having x subdivisions on each edge. T is the union of T_x, for $x = n^{1/3}, \ldots, 2n^{1/3}$. A *branching node* in T_x will be a node of degree 3. It is easy to check that T contains at most n nodes.

Note that T contains a straight path of length $2n^{1/3}$ between two vertices. So, by Lemma B.2, there is $n^{1/3} \leq x \leq 2n^{1/3}$ such that if u, v are two branching nodes in T_x joined by a straight path, then $S_{l_\Delta}[u, v]$ is $\Omega(\log n' / \log\log n)$. Now, iteratively using Lemma B.1, we can demonstrate a path from the root to a leaf y of T_x such that routing from the root of T_x to y requires stack depth $\Omega\left(\frac{\log^2 n}{\log\log n}\right)$. ∎

C Proofs of Section 4

Proof of Lemma 4.6: We will prove the lemma for the slightly different edge weights $c_e = \min(i, i') + (1+\epsilon)\min(j, j')$, but it is easy to alter the proof to give the result for the original edge-lengths.

Consider the vertices in Figure 5. We claim that cost path P_1 is less than that of P_2. Indeed, cost of $P_1 = (i+1)+(1+\epsilon)j+i+(1+\epsilon)j = 2i+2(1+\epsilon)j+1$, while cost of $P_2 = i+(1+\epsilon)(j+1)+i+(1+\epsilon)j = 2i+2(1+\epsilon)j+1+\epsilon$. Similarly, cost of $P_3 = i+(1+\epsilon)j+i+(1+\epsilon)j = 2i+2(1+\epsilon)j$, whereas cost of $P_4 = i+1+(1+\epsilon)j+i+(1+\epsilon)(j+1) = 2i+2(1+\epsilon)j+2+\epsilon$. Thus, cost of P_3 is at most that of P_4. So, it follows that no shortest path can contain a bend of type P_2 or P_4. Now, consider

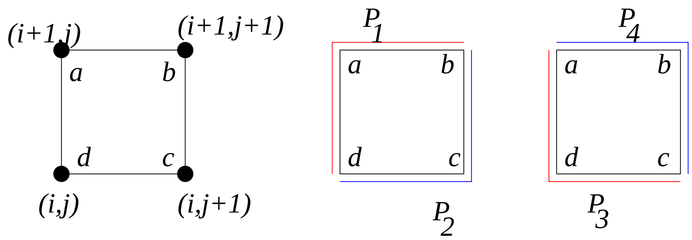

Figure 5: The local path interchange operations

Figure 6. We claim that in both the cases, path P_1 is shorter than path P_2. Let us consider the left figure first. Cost of path $P_1 = (i+1)r+(1+\epsilon)(j+\cdots+(j+r)) = ir+(1+\epsilon)(j+\cdots+(j+r))+r$, while cost of path $P_2 = i+(1+\epsilon)j+ir+(1+\epsilon)(j+\cdots+(j+r))+i+(1+\epsilon)(j+r) = ir+(1+\epsilon)(j+\cdots+(j+r))+r+i+2(1+\epsilon)j+2i+\epsilon r$. So, cost of P_1 is at most that of P_2.

Let us now look at the case on the right of Figure 6. Cost of $P_1 = (1+\epsilon)(j+1)r+(i+\cdots+(i+r)) = (1+\epsilon)jr+(i+\cdots+(i+r))+(1+\epsilon)r$. Cost of $P_2 = i+r+(1+\epsilon)j+(1+\epsilon)jr+(i+\cdots+(i+r))+i+(1+\epsilon)j = (1+\epsilon)jr+(i+\cdots+(i+r))+r+2i+2(1+\epsilon)j$. If ϵ is small enough, $\epsilon r < 1 < 2i$. So, we have that cost of P_1 is at most that of P_2. This fact shows that whenever a shortest path contains two

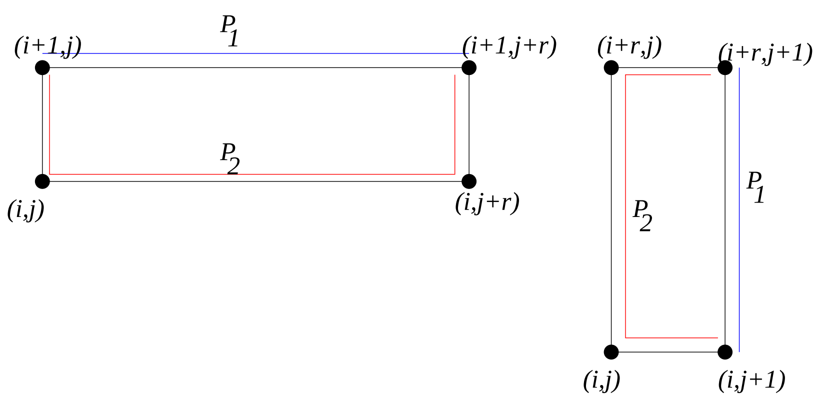

Figure 6: The straight path is the shortest one

vertices whose i or j coordinates are same, then it must contain the straight path joining them. Such a path with no bends of type P_2 or P_4 of Figure 5 must be a path with at most one bend. This proves the theorem. ■

Proof of Lemma 4.7: We say that a connected path P in T is *straight* if it does not have any bends and is of maximal length (i.e., adding any other edge of T to P will result in a bend). Let

$P_1, \dots, P_k$ be the set of all straight paths in T. We denote the vertex set of P_i also by P_i. It is easy to see that for all i, $|P_i| \le t$. Furthermore, for any $i \ne j$, $|P_i \cap P_j| \le 1$.

Construct a new graph $T' = (V', E')$ as follows, where V' contains one vertex p_i for each path P_i. E' contains an edge joining p_i and p_j if and only if $P_i \cap P_j \ne \emptyset$. It is not too difficult to show that T' is a tree. Furthermore, the following claim follows directly from the property of weights on edges.

Claim C.1 *Let $u, v \in T$, $u \in P_i$, $v \in P_j$. T preserves the shortest path between u and v only if $i = j$ or (p_i, p_j) is an edge in T'.*

Let $t_i = |P_i|$, and define the cost of the tree T' to be

$$f(T') = \sum_{p_i \in V'} t_i^2 + \sum_{(p_i, p_j) \in E'} (t_i - 1)(t_j - 1) \tag{C.1}$$

It follows from Claim C.1 that $|\mathcal{S}_T| \le f(T')$, and so it suffices to obtain an upper bound on $f(T')$.

For the rest of the proof, we do not look at the semantics of the sets again, but instead argue about arbitrary set systems on t^2 vertices, where each set P_i is of size $t_i \le t$, any two sets intersect in at most one element, and their intersection graph is a tree. For any such intersection tree T', we assign weight t_i^2 to each node and $(t_i - 1)(t_j - 1)$ to each edge (p_i, p_j) in T'. Now $f(T')$ be the total weight of vertices and edges in T'. We now show the following claim, which proves the theorem.

Claim C.2 *For any such intersection tree T', $f(T')$ is $O(t^3)$.*

Proof of Claim C.2: Let us first record the following lemma.

Lemma C.3 *Let p_i be a leaf in T' and p_j be the parent of p_i in T'. Then, either $t_i \ge t/2$ or $t_j \ge t/2$. If p_j is a degree two node and p_i is its unique child, then $t_i \ge t/2$ or $t_j \ge t/2$.*

Proof of Lemma C.3: Suppose $t_i, t_j < t/2$. Delete P_i and replace P_j by $P_i \cup P_j$; it is easy to see that the tree corresponding to this set system is the tree T' with p_i deleted (because P_i was disjoint from all other sets except P_j). The increase in weight of the tree is greater than

$$\begin{aligned} (t_i + t_j - 1)^2 - t_i^2 - t_j^2 - (t_i - 1)(t_j - 1) &= 2t_i t_j - 2t_i - 2t_j - t_i t_j + t_i + t_j \\ &= (t_i - 1)(t_j - 1) - 1 \ \ge \ 0. \end{aligned}$$

The argument about degree 2 nodes is similar, and is omitted. ■

We say that a leaf p_i in T' is *bad* if $t_i < t/2$. Delete all bad leaves from T' to get a tree T''. Then, the lemma above implies that all leaves p_i in T'' have the property $t_i \ge t/2$. We now claim that the tree T'' without the bad nodes has $O(t)$ nodes.

Indeed, let I be the index set of those p_i such that $t_i \ge t/2$. We claim that $|I| = O(t)$. To see this, note that no three of the sets P_i intersect and at most $|I|$ of the pairs of P_i have any pairwise intersection, since their intersection graph is a forest. Hence the principle of inclusion and exclusion implies that

$$t^2 \ge |\cup_{i \in I} P_i| \ge \sum_{i \in I} t/2 - |I| = (t/2 - 1)|I|.$$

Hence there are at $O(t)$ leaves in T'', which implies in turn that there are $O(t)$ nodes of degree 3 or more. Any degree node 2 which has less than $t/2$ elements can be charged uniquely to its child, which has more than $t/2$ elements by Lemma C.3.

Also, the contribution of cost of edges in T'' to $f(T')$ is at most $O(t^3)$, since each edge can contribute at most t^2. The contribution of edges joining a bad leaf to its parent in T' is at most t^3, since $\sum_i t_i$ (where the sum is over bad leaves) is at most t^2, the bad leaves being all disjoint.

Finally, we have to add up vertex contributions. T'' has $O(t)$ nodes, each having at most t elements. So the vertex weight contribution of these vertices is at most $O(t^3)$. Finally, the bad leaves are all disjoint, so their weights can be bounded by the following fact:

Fact C.4 *Suppose x_i are positive integers such that $\sum_i x_i \leq t^2$ and $x_i \leq t$. Then $\sum_i x_i^2 \leq t^3$.*

Summing all these terms up shows that $f(T') = O(t^3)$, proving the theorem. ■ ■