Path layout on tree networks: Bounds in different label switching models

Anat Bremler-Barr [*] Leah Epstein[†]

Abstract

Path Layout is a fundamental graph problem in label switching protocols. This problem is raised in various protocols such as the traditional ATM protocol and MPLS which is a new label switching protocol standardized recently by the IETF. Path layout is essentially the problem of reducing the size of the label-table in a router. The size is equivalent to the number of different paths that pass through the router, or start from it. A reduction in the size can be achieved by choosing a relatively small number of paths, from which a larger set is composed using concatenation.

In this paper we deal with three variations of the Path Layout Problem according to the special characteristics of paths in three label switching protocols, MPLS, ATM and TRAINET. We focus on tree networks, and show an algorithm which finds label-tables of small size, while permitting concatenation of at most k paths. We prove that this algorithm gives worst case tight bounds (up to constant factor) for all three models. The bounds are given as a function of the size of the tree, and the maximum degree.

1 Introduction

Improving label switching technologies is a step towards better QoS (Quality of Service) in high speed and IP networks. Label switching is done by using a quite simple idea of label based switching decisions, a packet is switched in the network according to a label it carries. The concept of Label Switching is not new. Its origins are in virtual circuit networks such as ATM networks. However, only recently the combination of switching technology and IP routing became a popular avenue of research and development. The large number of different label switching protocols [7] (tag-switching [17], CSR [14], IP-Switching [16], and threaded indices [6]) motivated the IETF to develop MPLS (Multi Protocol Label Switching), a standard approach to label switching [4]. MPLS was designed to combine the best of several approaches.

MPLS was developed to overcome some drawbacks of the standard IP routing. IP routing is basically a simple method of hop by hop routing. When a packet reaches a router, the routing decision (on the next router in the path) is based only on the destination address of the packet. At any given point in time all packets with the same destination address would be routed in exactly the same way, regardless of any other parameter of the packet. At each router along the packet path, the destination address of the packet is examined, and the longest prefix that matches the address, out of all the prefixes (usually several tens of thousands) in a forwarding table is found. The packet is then routed according to the information associated with the Best Matching Prefix (BMP) in the forwarding table.

The high cost of computing the BMP at each router combined with the inability to distinguish between different flows with the same BMP motivated the development of MPLS. All the packets of the same flow are tagged with a unique label upon entering the network layer. At each switch (router) on the path the label is used to switch (rather than route) the packet to its next hop. The label is used as an index in the switching table. That is, label switching supports the concept of "route once and switch

[*]School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. bremler@idc.ac.il

[†]School of Computer Science, The Interdisciplinary Center, Herzliya, Israel. lea@idc.ac.il. Research supported in part by Israel Science Foundation (grant no. 250/01).

many". In other words, for all the packets in a flow, routing is performed once when setting up the label path, and the label is then used for all the packets at all the intermediate switches. Essentially MPLS establishes a circuit between the source and destination which in the terminology of MPLS is called Label Switch Path (LSP).

The basic advantages of MPLS are as follows. Elimination of the time consuming BMP enables higher rates of packet processing. The second advantage is support of QoS forwarding by assigning different labels to flows that require different services even if destined to the same destination (we do not use such options in this paper). Finally, MPLS enables explicit routing. The source router in the path establishes the path by some setup mechanism (such as RSVP). Henceforth the forwarding is performed by switching on the label which was associated with the explicit route. The explicit routing feature of MPLS enables the support of good traffic engineering (design of paths in the network), load-balancing, and support for resource reservations.

In order to support hierarchical routing (between networks and inside them), MPLS supports multi label paths by placing a stack of labels on each packet of a corresponding flow. In addition, in MPLS each router may use operations on the stack of labels, it can be instructed to push or pop one or more labels. This mechanism is also useful for path concatenation [5, 1, 2]. A path is described by concatenation of LSPs and each LSP described by a label in the stack label. One of the applications of path concatenation is using a small set of LSPs to compose a larger set. The main advantages of this application is reducing the number of labels necessary in the network thus enabling more label switched routes with the same number of labels. Labels in MPLS are a scarce resource. They are the key elements in the scalability of MPLS, and the more labels we need the larger the switching tables are. Since the switching has to operate at high speeds, it is built out of expensive memories and its size is critical.

This paper deals with the following fundamental graph problem. How to choose a small set of routes in order to compose a larger set of routes by using concatenation. We mainly focus on tree networks, which are relevant to many server client applications, such as multicast, and can also be used as a building block for solutions in general graphs [9].

This question is not new. This is the basically **Virtual Path Layout** in ATM, that was considered in the literature [18, 19, 8, 3, 12, 9, 10] in the blossoming year's of ATM protocol. ATM [13] is also protocol based on circuits, where a circuit is a path defined between a source and a destination. In ATM, a VC or a Virtual Channel connects between users and may concatenate many VPs (virtual paths). The main rule of VP is an internal network usage, of reducing the cost of management the establishment of VCs. The concatenation of paths in ATM is done by using the VCI/VPI labels, where VCI is the index of a VC and VPI is the index of a VP. Basically the VCI is the glue that concatenates several VPIs.

Our first contribution in this paper, is giving tight (up to a constant multiplicative factor) upper and lower bounds for the worst case maximum load on any router in a tree network. This result also shades some light on the corresponding open issue in ATM networks.

The second contribution in this paper, is to model the problem of path layout in three different label switching models, ATM, MPLS and a new suggested extended version of MPLS, Trainet. We examine the different power of the models by demonstrating them on the problem of path layout on tree networks.

One of the key differences between MPLS and ATM is caused by the path merging capability in MPLS. MPLS permits tagging all the packets with the same remaining path with the same label, even if there initial path was different [4]. Hence when a label switch path is built in the network, one may use all its, with no extra charge of building and tagging these paths with different labels.

The third model, the Trainet Model, is a model suggested by [1]. The main idea, of this extended MPLS model, is to increase the utilization of each label by using it to define each sub-path of a labeled switched path (LSP) as a legal route. It is achieved by replacing the label with a <label,counter> pair. At each switch the counter shows how many more hops the corresponding packet should take on this LSP. This is much like taking a subway, where the packet has to go off the train after a specified number of stations. This model resembles the idea of Manhattan networks [15].

The outline of the rest of the paper is as follows. The different models are formalized in Section 2. We discuss related work in Section 3. The main result of the paper is presented in Section 4. The algorithms for different cases are given in Section 5. The lower bounds for different cases are given in Section 6. Section 7 explores some results for the different models, which are derived from the main result. We finish the paper with some conclusions.

2 Model

All label switching models establish switch-paths. A packet moving in a network, may move only along a switch-path. Each packet is assigned a label, corresponding to the switch path it is supposed to use [1]. A packet traverses its path using a switching technique. We begin with defining switch-paths. Virtual Paths (in ATM), Label Switch Paths (in MPLS) and Trainet Switch Paths (in Trainet) are all instances of a switch-paths.

We model the network, as a bidirectional graph $G = (V, E)$, where the nodes are routers and the edges are physical connections between the routers. In some cases (where we are interested in paths from a single source) we consider directed graphs.

Definition 1 *A switch-path SP in graph G is a simple directed path.*

The label switching models differ in the number and the type of different routes (sub-paths of switch-paths) that a packet may traverse using a specific switch-path of the specific model. Metaphorically, the switch-path resembles a directed express train-line, which the packets can use for fast switching. The routes that packets may use are all sub-paths of the train line. The ATM switch-paths, called also Virtual Paths, define only one route each. In order to use the path, a packet must begin its trip in the first station, the source of the switch-path, and may leave only in the last station, the destination. In MPLS, the packet may enter the train-line at any station, but must stay till the end of the switch-path. A label switch path in MPLS defines a subset of routes, each of which corresponds to a suffix of the label switch-path. In the Trainet model, a packet may enter the train-line at every station, and get off at any later station. A Trainet switch-path defines a set of routes, each of them is some sub-path of the switch-path i.e., each path in the Trainet model introduces all its sub-paths as routes. Next, we give definitions, which formally define the routes that are induced by each type of switch-path.

Definition 2 *Let V be a Virtual Switch Path. We define the set of routes that are induced by VSP in the ATM model, simply as $\{V\}$.*

Definition 3 *Let L be a Label Switch Path. We define the set of routes that are induced by L in the MPLS model, as the set of routes $\{r | r$ is suffix of $L\}$.*

Definition 4 *Let T be a Trainet Switch Path. We define the set of routes that are induced by T in the Trainet model, as the set of routes $\{r | r$ is (a non-empty) sub-path of $T\}$.*

In the next definition, we define the **path layout** in a Model M (where M can be ATM, MPLS or Trainet).

Definition 5 *Let $G = (V, E)$ be a directed graph. Let P be a collection of routes in G. A collection of switch-paths P' in G is* **a k-hop path layout** *of P in the model M, iff for every $r \in P$, it can be created by concatenating of at most k valid routes induced by P' in the model M.*

[1]We ignore the fact that the actual label value of a packet is swapped while traversing the switch path, in order to avoid global synchronization of the label value. E.g. a packet arriving at a router with a label l-in is forwarded according to the outgoing port (line card) found at the switching table at entry l-in. While being switched, the l-in label is swapped with a l-out label found in that entry of the switching table. Note that this label swapping does not change the number of label values required for each router, and hence in this paper we ignore the above mechanism.

Our paper focuses on finding a path layout that minimizes the size of the largest label-table (or the switching table) of any router. A label-table of a router is the table that contains the database of the labels. The size of this table for a given router vertex is also called **the vertex load**. Next we give a formal definition of vertex load in the different Models. Similarly to the load of a vertex, one can define the load on an edge. We prefer to discuss the vertex loads due to their direct connection to the sizes of the label-tables. The vertex load of node v is denoted $L(v)$.

Definition 6 *Let VSP be the set of all the virtual paths in an ATM network. Let $VSP(v)$ be the subset of paths $p \in VP$ such that $v \in p$. $L(v) = |VSP(v)|$.*

Definition 7 *Let LSP be the set of all label switch paths p in a MPLS network. Let LSP' be the set of all routes induced by LSP. Let $LSP'(v)$ be the set of all routes p in LSP', such that v is the first vertex in p. $L(v) = |LSP'(v)|$. Note that this definition captures the merging of labels capabilities of the MPLS model.*

Definition 8 *Let TSP be the set of Trainet switch paths in a Trainet Network. Let TSP' be the set of all routes induced by TSP. Let $TSP'(v)$ be the subset of all routes in TSP' such that v is the first vertex in p. Let $TSP''(v)$ be a minimal subset of $TSP'(v)$ such that for each $p' \in TSP'(v)$ there exists a route $p'' \in TSP''(v)$ such that p' is a prefix of p''. $L(v) = |TSP''(v)|$. Note again, that this captures the merging of labels capabilities of the Trainet model.*

We say that a path payout has vertex load m if the maximum load on any vertex in the graph is m. Our paper deals with finding a k-hop path layout, that requires minimum vertex load, and hence minimizes the memory used for the incoming label-table at the router. The paper focuses on the case of a tree network where the basic set of paths P is naturally a set of shortest paths. In the following definitions, we define three problems of path layout in a tree, one-to-all path layout, all-to-one path layout, and all-to-all path layout.

Definition 9 *Let r be a designated vertex in graph G. A* **one-to-all path layout** *is a path layout where r has shortest path routes to all other nodes in G.*

Definition 10 *Let r be a designated vertex in graph G. A* **all-to-one path layout** *is a path layout where all the vertices in G have shortest path routes to r.*

Definition 11 *A* **all-to-all path layout** *is a path layout in G, where every node has a shortest path route to every other node in the graph.*

In order to give some insight on the different models, and as a building block to the general problems, we give in Appendix A.1 as an example the path layout for $k = 1$ (one-hop path layout). Throughout the paper, the symbol n denotes the number of nodes in the network. The **size** of a tree is defined to be the number of nodes in it.

3 Related work

While there is extended literature about the virtual path layout in general ATM network [18, 19, 8, 3, 12, 9, 10, 11], there is no work, to the best of our knowledge, that deals with the new model of label switch path layout.

The paper of Gerstel, Cidon and Zaks deals with the problem of virtual path layout in a tree network [9]. In that paper, an optimal polynomial algorithm for finding a virtual path layout (in ATM model) with minimum edge load was introduced. However, to the best of our knowledge, there is no tight bound known for vertex loads in tree networks. The paper [9] does not analyze the resulting optimal edge load found by the algorithm. The result of [9] is important and raises many further questions.

We believe that there is practical need in knowing the maximum resulting load. Moreover the paper discusses the load on edges and not on the vertices. We believe that both measures are important for practical reasons. Our paper, fills this gap, and gives a lower bound for the minimum vertex load that any algorithm can find. We feel that our lower bound, is an important step in reaching a real understanding and estimating the gain by using path concatenation.

Our paper also introduces polynomial algorithms, which give a tight upper bound. The paper [10] gives an upper bound of $O(\sqrt{n})$ of edge load for the 2-hop path layout, which leads to vertex load of $O(d\sqrt{n})$. In this paper we give algorithms for the general k-hop path layout problem. Specifically, for the special case of the 2-hop path layout, our algorithm gives a vertex load of only $O(\sqrt{dn})$.

To the best of our knowledge, our work is also the first paper to model the virtual path problem in the MPLS and Trainet models.

4 Main results

In the rest of the paper, we prove the following theorem, and derive some results on the different models from it. We consider a network which is a rooted directed tree. We define d to be the maximum out-degree of any node in the directed tree. It is also possible to see the tree as undirected. This does not change the result since the maximum degree in that case is at least d and at most $d+1$ and a change in d by a multiplicative factor of at most two does not affect the results. I.e., the following theorem gives tight bounds on the worst case vertex load for the one-to-all k-hop path layout in tree networks. The theorem holds in all three models.

Theorem 1 *Given a rooted directed tree with n nodes and maximum degree d. For an integer $k > 1$, the following function f_k describes the maximum size of the switching table of any node (i.e, the vertex load), for a fixed value of k (maximum number of concatenated paths).*

If k is even then

$$f_k(n,d) = \begin{cases} \Theta(n^{\frac{1}{k}} d^{\frac{k-1}{k}}), & \text{for } d \in [1, n^{\frac{1}{k+1}}]; \\ \Theta((nd)^{\frac{2}{k+2}}), & \text{for } d \in [n^{\frac{1}{k+1}}, n^{\frac{2}{k}}]; \\ \Theta(d), & \text{for } d \in [n^{\frac{2}{k}}, n]. \end{cases}$$

If k is odd then

$$f_k(n,d) = \begin{cases} \Theta(n^{\frac{1}{k}} d^{\frac{k-1}{k}}), & \text{for } d \in [1, n^{\frac{1}{k+1}}]; \\ \Theta(n^{\frac{2}{k+1}}), & \text{for } d \in [n^{\frac{1}{k+1}}, n^{\frac{2}{k+1}}]; \\ \Theta(d), & \text{for } d \in [n^{\frac{2}{k+1}}, n]. \end{cases}$$

There is one exception which is the case $d = 1$ in the Trainet model, for which $f_k(n,d) = 1$.

To prove the theorem, we need to show algorithms for all cases, and to give negative examples to all cases as well. We do that in the next sections.

5 Algorithms

We start with designing the algorithms, and construct the lower bounds in the next section. We define a procedure DECOMPOSE. This procedure is used by the algorithms for all the different cases. The procedure receives a rooted directed tree T' of size m, and an integer $\ell \leq m$ and returns a node v. v is the root of a minimal sub-tree of size at least ℓ (we require the sub-tree to contain all the descendants of v in T'). Note that since the sub-tree is minimal, the size of the sub-trees of the children of v are smaller than ℓ. The operation of DECOMPOSE is iterative and simple; It moves a pointer along a descending path in the tree. The pointer starts at the root of T, and goes along a directed edge to a child whose sub-tree is of size at least ℓ, as long as such a child (whose sub-tree is large enough) still exists. We later show how to combine several iterations of decompose into a single run of DFS.

All algorithms have a similar general structure. The original tree T is decomposed recursively into smaller and smaller trees, until the trees are small enough. In each decomposition step, some switch-paths are defined.

Case 1: An algorithm for even or odd k and $d \in [1, n^{\frac{1}{k+1}}]$.
Let $\ell_0, \ldots, \ell_{k-1}$ be a monotonically decreasing sequence such that $\ell_0 = n$. The values ℓ_i serve as thresholds for sizes of trees (numbers of nodes). We define sets of roots $R_0, \ldots R_{k-1}$ in the following way. $R_0 = \{r\}$, where r is the root of T. R_i contains nodes, which are defined to be roots of sub-trees (those are sub-trees which contain a subset of their descendants). such sub-trees have sizes of at most ℓ_i. Note that the sets are not necessarily disjoint. Moreover, each vertex $r_i \in R_i$, has an associated vertex in R_{i-1}, which is expressed by a function $f_i : R_i \rightarrow R_{i-1}$. We explain below how to create the sets and the functions.

The algorithm uses the term "trees of type i". Those are trees whose root is in R_i. Clearly the size of a tree of type i ($0 \leq i \leq k-1$) is of size at most ℓ_i. The original tree T is a type 0 tree. A root of a tree of type i, i.e. a node of R_i, is called a root of type i. The root of T is also called *the main root.*

Decomposition into smaller trees: The algorithm uses the set of thresholds to recursively define sets of trees of type i for each i. Starting with T (which is the only tree of type 0), trees of type $i+1$ are simply defined by decomposing trees of type i.

The structure of switch-paths: Switch-paths are defined in a way that for each root r of type $i+1$, either there exists a root of type i ($f_{i+1}(r)$) with a switch-path going to r, or r itself is a type i root. The exact way to do this is defined together with the defining the decomposition algorithm. The

Finally each root of type $k-1$ is defined to have switch paths to all the nodes in its sub-tree (of type $k-1$). The value of ℓ_{k-1} has to be relatively small so that roots of type $k-1$ do not have a large vertex load.

Using the above set of paths we get that the set of all concatenations of at most $k-1$ switch-paths from the main root, allow us to get to every root of type $k-1$. Using the switch paths from roots of type $k-1$ we get that it is possible to use a concatenation of at most k switch-paths to get from the main root to any node.

An algorithm for decomposing a tree of type i into trees of type $i+1$: Given a tree T_i of type i, fix $\ell_{i+1} = n^{\frac{k-i-1}{k}} d^{\frac{i+1}{k}}$ and do the following until the tree T_i is left with at most ℓ_{i+1} nodes: (If already $|T_i| \leq \ell_{i+1}$, then the tree is not decomposed further). Run DECOMPOSE with the parameter ℓ_{i+1} on the tree, and get a sub-tree rooted at x. Add all paths from the root of T_i to x and its direct children to the set of switch-paths, and remove this sub-tree from T_i. Let x be the root of the removed sub-tree. Each direct child y of x is defined to be a root of type $i+1$ (the tree of type $i+1$ contains all the descendants of y that were not removed previously from the tree). Since the sub-tree of x is minimal, the size of the sub-tree of each child of x is of size at most ℓ_{i+1}. The remainder of T_i (after all removals) is also defined to be a tree of type $i+1$. Recall that by definition, the remainder is of size at most ℓ_{i+1}.

Analysis of the vertex load: The number of times that DECOMPOSE was run (for a given i) is at most $\frac{\ell_i}{\ell_{i+1}}$. Consider the labels given only while decomposing trees of type i into trees of type $i+1$. For each sub-tree of type $i+1$, at most $d+1$ new switch-paths are defined, hence the root of T_i is assigned at most $(d+1)\frac{\ell_i}{\ell_{i+1}}$ new labels. Using $\ell_i = n^{\frac{k-i}{k}} d^{\frac{i}{k}}$, we get $\frac{\ell_i}{\ell_{i+1}} = n^{\frac{1}{k}} d^{-\frac{1}{k}}$ and the number of new labels assigned to each root of type $i < k-1$ is at most $(d+1)n^{\frac{1}{k}} d^{-\frac{1}{k}} = \Theta(n^{\frac{1}{k}} d^{\frac{k-1}{k}})$ labels. For $i = k-1$, the size of each tree of type $k-1$ is at most $\ell_{k-1} = n^{\frac{1}{k}} d^{\frac{k-1}{k}}$, hence each root of type $k-1$ also has $O(n^{\frac{1}{k}} d^{\frac{k-1}{k}})$ new labels defined for switch-paths to its descendants. To calculate the total number of labels that any node has received, note that each node in the tree T belongs to at most one tree of each type. While considering trees of type i, all switch-paths were defined inside the trees. In the worst case, a node in a tree of type i belongs to all switch-paths starting at its root. Since there are

k types of trees, and k is constant, each node has at most $\Theta(n^{\frac{1}{k}} d^{\frac{k-1}{k}})$ labels of different paths.

Case 2: Algorithm for odd k and $d \geq n^{\frac{1}{k+1}}$.
The algorithm is similar to the previous algorithm. We specify the differences between this case and case 1. Instead of k types of trees, there are only $\frac{k+1}{2}$ types. The number of thresholds is naturally also $\frac{k+1}{2}$, and again $\ell_0 = n$. Clearly, the number of recursive decomposition phases is also smaller.

The structure of switch-paths: For each root of type $i+1$ ($i > 0$) there exists a concatenation of at most two switch-paths to it from some root of type i (the root of type $i+1$ may be already a root of type i, this can be seen as a special case). In the trees of type $\frac{k-1}{2}$, there is a switch-path from each root to all the nodes in the sub-tree.

The maximum number of switch-paths that need to be concatenated to be able to get from the root to some node is two times the number of phases of decomposition, plus 1 (for the switch-paths from every root of type $\frac{k+1}{2}$ to its descendants). This number is exactly k.

An algorithm for decomposing a tree of type i into trees of type $i+1$: We set $\ell_{i+1} = n^{\frac{k-2i-1}{k+1}}$. To decompose a tree of type i, T_i, into trees of type $i+1$, we again run DECOMPOSE with the parameter ℓ_{i+1}, removing each tree until $|T_i| \leq \ell_{i+1}$. Let x be a root of a removed sub-tree. A switch-path is fixed from the root of T_i to x, and switch-paths are also fixed from x to all its direct children. The sub-tree of each direct child is defined to be a tree of type $i+1$. The remainder of T_i after all removals is also defined to be a tree of type T_{i+1}.

Analysis of the vertex load: To calculate the number of labels given to a node while considering trees of type i, note that there are two cases. Nodes got labels either on a switch-path from a root of type i, or on a switch-path of length 1 (to roots of type $i+1$ from their direct parents). In the first case, a root of type i is assigned at most $\frac{\ell_i}{\ell_{i+1}}$ labels for $i < \frac{k-1}{2}$ and at most ℓ_i for $i = \frac{k-1}{2}$. This equals to $n^{\frac{2}{k+1}}$. In the second case, a node is defined to have switch-paths going to its children, which gives at most d labels for a node. Hence the maximum number of labels given to one node is $O(\max(d, n^{\frac{2}{k+1}}))$. Using similar reasoning as in the first case, the total number of labels given to one node in total is also $O(\max(d, n^{\frac{2}{k+1}}))$.

Case 3: Algorithm for even k, $d \in [n^{\frac{1}{k+1}}, n^{\frac{2}{k}}]$.
We use a combination of the first and the second algorithms. The number of types of trees is $\frac{k}{2}+1$. Define $\ell_0 = n$ and $\ell_i = (nd)^{\frac{k-2i+2}{k+2}}$ for $i > 0$. This gives $\frac{\ell_i}{\ell_{i+1}} = (nd)^{\frac{2}{k+2}}$ for $0 < i < \frac{k}{2}$ and $\frac{\ell_0}{\ell_1} = n^{\frac{2}{k+2}} d^{-\frac{k}{k+2}}$. Note that ℓ_1 can be defined since $d^k \leq n^2$. The construction of switch-paths from the main root, to trees of type 1 is done as in the first case algorithm, i.e. the switch-paths are from the main root to all roots of decomposed trees and their direct children (who become roots of type 1). The definitions of switch-paths while decomposing a tree of type $i > 0$ into trees of type $i+1$ is done as in the second algorithm, i.e. switch-paths from the type i root to the roots of decomposed trees, and switch-paths from them to their direct children (who will be the type $i+1$ roots). The switch-path inside the type $\frac{k}{2}$ trees are defined as in both algorithms, from the root, to all the nodes. The size of such sub-tree is at most $\ell_{\frac{k}{2}} = (nd)^{\frac{2}{k+2}}$.

The main root was assigned $O(d\frac{\ell_0}{\ell_1}) = O(dn^{\frac{2}{k+2}} d^{-\frac{k}{k+2}}) = O((nd)^{\frac{2}{k+2}})$ labels which bounds the number of labels each node in the type 0 tree was assigned. In all other cases a root of type $i > 0$ received at most $\Theta((nd)^{\frac{2}{k+2}})$ labels and some nodes received extra $O(d)$ labels. Since $d \leq (nd)^{\frac{2}{k+2}}$, each node in a tree of type i was assigned $O((nd)^{\frac{2}{k+2}})$ labels. Since k is constant, summing over all types of trees still gives $\Theta((nd)^{\frac{2}{k+2}})$.

Case 4: Algorithm for even k, $d \geq n^{\frac{2}{k}}$.
Let $a = \lceil \frac{\log n}{\log d} \rceil - 1$. We run the second algorithm with $a+1$ types of trees $\ell_i = \frac{n}{d^i}$, for $0 \leq i \leq a$.

Since $n \leq d^{\frac{k}{2}}$, $a \leq \frac{k}{2} - 1$. The size of trees of type a is at most d. The number of concatenated paths is at most $k-1 < k$. Since $\frac{\ell_i}{\ell_{i+1}} = d$ and $\ell_a \leq d$, each node in a tree of type i is assigned $O(d)$ new labels, hence in total, for constant k, each node has $O(d)$ labels.

Running time: We assume that the algorithm needs to gather information, but does not need to construct the paths. The decomposition of a tree of type i into trees of type $i+1$ can be done e.g. while running a Depth First Search on the tree. Each node may have a counter of the size of its sub-tree. As soon as the search returns to a node with enough descendants, switch-paths from the root are defined, and the counter is set to zero. We run this once for each tree (may be done recursively, running a new DFS when a node is declared to be a root of some type), The running time is $O(n)$ for each type of trees (each node or edge belongs to at most one tree of each type). The total running time is $O(kn)$. We can conclude that the running time is linear in n.

The upper bounds above assume that k is a constant, which is the case in reality. However, even if k is non-constant, it is possible to get the same upper bound in the Trainet model. (see Appendix A.2).

6 Lower bounds

The lower bounds presented in this section are valid for all three models. Whenever, we use routes, in the proof, the routes are induced by the switching path according to any of the models. To prove ***lower bounds*** we introduce several trees in which at least one node has a large switching table.

If $d = 1$, then the tree is a chain. We need to show that there exists a node with $\Omega(n^{\frac{1}{k}})$ labels in the ATM and in the MPLS models. Assume to the contrary that all nodes have a table of size at most $\frac{1}{3}n^{\frac{1}{k}}$. Starting at the root of T, using a concatenation of exactly i routes, the root has access to at most $(\frac{1}{3}n^{\frac{1}{k}})^i$ different nodes. Using a concatenation of at most k routes we get access to at most $1 + (\frac{1}{3}n^{\frac{1}{k}}) + (\frac{1}{3}n^{\frac{1}{k}})^2 + \ldots (\frac{1}{3}n^{\frac{1}{k}})^k \leq 1 + \frac{1}{3}n + (\frac{1}{3})^2 n + \ldots \leq 1 + \frac{n}{2} < n$ (for $n \geq 3$). Contradiction.

For other cases, we build trees of sizes at most n that give negative examples.

Case a: Lower bound for $d \in [2, n^{\frac{1}{k+1}}]$.
We define the following tree types. Let $n' = n/2$. Let T_1 be a maximum complete binary tree with at most $(\frac{n'}{d})^{\frac{1}{k}}$ leaves and T_2 a maximum complete binary tree with at most $(\frac{n'}{d^{k+1}})^{\frac{1}{k}}$ leaves (if $\frac{n'}{d^{k+1}} < 1$, or $(\frac{n'}{d})^{\frac{1}{k}} < 1$ use a tree which consists of a single node instead of a binary tree. Note that $d^k < n$).

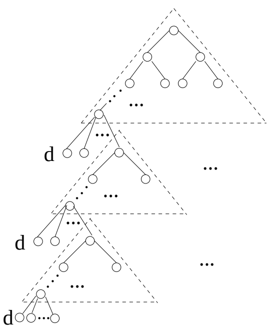

Figure 1: Illustration of the the lower bound tree of Case A

Let T_1' (resp. T_2') be a T_1 (resp. T_2) tree such that each leaf has d children. We define a concatenation of two trees S_1 and S_2 as follows : S_1S_2 is rooted at the root of S_1 and each leaf of S_1 has a S_2 as a sub-tree (the leaf of the S_1 is a root of an S_2). Let $T = T_1'(T_2')^{k-1}$ (See Figure 1). Note that T has at most n' leaves, and at most $2n' = n$ nodes. We need to show that some node has $\Omega(n^{\frac{1}{k}} d^{\frac{k-1}{k}})$ labels.

For a set of vertices Q in a tree T, let $D(Q)$ denote the set of all descendants of nodes in Q (including Q i.e. $Q \subseteq D(Q)$). For a tree T with a given switching table, where the root can reach any node concatenating k routes or less, let $T(i)$ ($0 \leq i \leq k$) denote the set of nodes that are reachable from the root by at concatenating at most i routes, or some descendant of the node is reachable. It is easy to see that $T(0)$ is the root and $T(k)$ is the set of all nodes.

Assume that at most $\frac{1}{4}(n'^{\frac{1}{k}} d^{\frac{k-1}{k}})$ labels for the root and at most $\frac{1}{8}(n'^{\frac{1}{k}} d^{\frac{k-1}{k}})$ labels for any other node are enough. We will reach a contradiction which would mean that some node needs $\Omega(n^{\frac{1}{k}} d^{\frac{k-1}{k}})$ labels. Let I_0 be the set of leaves of the tree T'_1, and let I_i be the set of leaves of $T'_1(T'_2)^i$ for $i = 1,..,k-1$.

If the root has at most $\frac{1}{4}(n'^{\frac{1}{k}} d^{\frac{k-1}{k}})$ labels and each other node has at most $\frac{1}{8}(n'^{\frac{1}{k}} d^{\frac{k-1}{k}})$ labels, we can prove the following claim.

Claim 2 *For every $0 \leq i \leq k-1$, the set I_i contains a subset I'_i which satisfies the following properties: 1. $|I'_i| \geq \frac{d}{2}$. 2. All vertices in I'_i have the same parent node. 3. $T(i+1) \cap D(I'_i) = \emptyset$ (i.e. it is impossible to get from the root of T to any of the nodes in the subset or any nodes in their sub-trees by concatenating at most $i+1$ routes).*

Proof: By induction. For the basis we calculate the size of the set I_0. Since T_1 is a maximum binary tree with at most $(\frac{n'}{d})^{\frac{1}{k}}$ leaves, it has more than $\frac{1}{2}(\frac{n'}{d})^{\frac{1}{k}}$ leaves and hence $|I_0| > \frac{d}{2}(\frac{n'}{d})^{\frac{1}{k}}$. If the root of T has at most $\frac{1}{4}n'^{\frac{1}{k}} d^{\frac{k-1}{k}}$ labels, then at most half of the nodes in I_0 have a route from the root passing through them (possibly stopping there). Hence there is at least one leaf of T_1 that at least half its (d) children do not have a route from the root to them or to some descendant of them. Let I'_0 be such subset, it satisfies all three properties.

For the inductive step (for $i > 0$) consider I'_{i-1} and let x be their parent. Let $\tilde{I}_i = I_i \cap D(I'_{i-1})$ (descendants of I_{i-1} which are leaves of $T'_1(T'_2)^i$). Assume to the contrary that there can be no I'_i that satisfies all three properties. Hence each node in the leaves of $T'_1(T'_2)^{i-1}T_2$ has at least $\frac{d}{2}$ children who are on a path created by concatenating $i+1$ routes starting from the root. In particular at least half of $\tilde{I}_i$ have such a path. In order to have a path to a node y in $\tilde{I}_i$ which is a concatenation of $i+1$ routes, there must be a route beginning at x or passing through x to y. The size of each T'_2 is at least $\frac{d}{2}(\frac{n'}{d^{k+1}})^{\frac{1}{k}}$. The set $\tilde{I}_i$ contains at least $\frac{d}{2}$ trees of type T_2 and hence $(\frac{d}{2})^2(\frac{n'}{d^{k+1}})^{\frac{1}{k}}$ nodes, and at least half of them have a route from x or an ancestor of x. This results in at least $\frac{1}{8}((n'd^{k-1})^{\frac{1}{k}})$ labels for x, which is a contradiction. ∎

The claim shows that there exists a leaf node of T which does not have a concatenation of k routes from the root which leads to this leaf. This is a contradiction. (Actually we showed that there are at least $\frac{d}{2}$ such leaves).

Case b: Lower bound for even k and $d \in [n^{\frac{1}{k+1}}, n^{\frac{2}{k}}]$
We build the following tree: Let $n' = n/2$. Let T_3 be a maximum complete binary tree with at most $(\frac{n}{d^{\frac{k}{2}}})^{\frac{2}{k+2}}$ leaves (this is well defined since $d \leq n^{\frac{2}{k}}$). Let T'_3 be a defined analogously to T'_i in previous proof. Let $T = (T'_3)^{\frac{k}{2}}T_3$. Note that T has at most n' leaves, and at most $2n' = n$ nodes. Define J_{2i-1} to be the leaves of $(T'_3)^i$ ($1 \leq i \leq \frac{k}{2}$), J_k be the leaves of T and let J_{2i} be the parents of J_{2i+1} ($0 \leq i \leq \frac{k}{2}-1$).

Assume to the contrary that at most $\frac{1}{4}(n'd)^{\frac{2}{k+2}}$ labels for the root and at most $\frac{1}{8}(n'd)^{\frac{2}{k+2}}$ labels for any other node are enough.

If the root has at most $\frac{1}{4}(n'd)^{\frac{2}{k+2}}$ labels and each other node has at most $\frac{1}{8}(n'd)^{\frac{2}{k+2}}$ labels, we can prove the following claim, which is similar to the claim for case a. The proof is given in the Appendix A.3.

Claim 3 *For all odd i, there exists a set $J_i' \subset J_i$ which satisfies the following properties: 1. $|J_i'| \geq \frac{d}{2}$. 2. All vertices in J_i' have the same parent node. 3. $T(i) \cap D(J_i') = \emptyset$ (i.e. it is impossible to get from the root of T to any of the nodes in the subset J_i' or any nodes in their sub-trees by concatenating at most i routes).*

For even $i > 0$, there exists a set $J_i' \subset J_i$ which satisfies the following properties 1. All vertices in J_i' have the same ancestor node z in J_{i-1}, i.e. they belong to a single T_3 tree. 2. $|J_i'| \geq \frac{1}{2}|D(\{z\}) \cap J_i|$ (i.e. J_i' contains at least half of the leaves of some T_3 tree). 3. $T(i) \cap D(J_i') = \emptyset$ (i.e. it is impossible to get from the root of T to any of the nodes in the subset J_i' or any nodes in their sub-trees by concatenating at most i routes).

Using the claim for $i = k$ we get that there exists a leaf of T which is not reachable from the root by concatenating k routes. Contradiction.

Case c: Lower bound for odd k, $d \in [n^{\frac{1}{k+1}}, n^{\frac{2}{k+1}}]$

Let T_4 be a maximum complete binary tree with at most $\frac{n^{\frac{2}{k+1}}}{d}$ leaves (note that this value is larger or equal 1). Let T_4' be defined analogously to T_1'. Let $T = (T_4')^{\frac{k+1}{2}}$. The proof for this tree is very similar to the previous proof.

Case d: Lower bounds for other cases.

For the two other cases we need to prove that at least d labels are necessary for some node. It is clear that a node of degree d has at least d labels, either because of paths starting at it, or paths passing through it. Hence the lower bound for even k and $d \geq n^{\frac{2}{k}}$ and for odd k and $d \geq n^{\frac{2}{k+1}}$ are immediate from any tree where some node has degree d.

7 Results in the different models

In this section we show several corollaries which adapt the basic algorithm to give an all-to-all path layout in general tree network fors the Trainet Model, and for directed tree networks in the MPLS model.

The special properties of the Trainet model allow us construct an all-to-all path layout with no extra cost. The proof of the following claim is in the Appendix A.4.

Claim 4 *Given a bidirectional tree. In the Trainet model, it is possible to build a k-hop all-to-all path layout with the same vertex load as a k-hop one-to-all path layout in the same model.*

In the MPLS model, it is also possible to extend the reachability with no extra cost. This can be done in the directed rooted tree.

Claim 5 *Given a directed tree, in the MPLS model it is possible to build a k-hop all-to-all path layout using label-tables of the same size as are used for a k-hop one-to-all path layout in the same model.*

Proof: In the MPLS model suffixes of switch paths may be used as routes. A suffix of a concatenation of at most k routes is a concatenation of at most k routes as well, and may be used as a k-hop path in the MPLS model.

8 Conclusions

We have given tight bounds on the one-to-all path layout in trees. This leaves several open questions. Is it possible to give tight bounds for general graphs? Is it possible to give tight bounds for the all-to-all path layout? Note that for the Trainet model this paper gives a tight bound for the all-to-all path layout, but the same question for the MPLS and ATM models is still open.

Acknowledgments

We gratefully acknowledge the fruitful discussions with Haim Kaplan and Yehuda Afek from Tel-Aviv University. We also thank Rob van Stee from CWI for reading the paper and commenting on it.

References

[1] Y. Afek and A. Bremler-Barr. Trainet: A new label switching scheme. In *Proc. of The Conference on Computer Communications (INFOCOM 2000)*, pages 874–883, 2000.

[2] D. O. Awduche and Y. Rekhter. Multi-protocol lambda switching: Combining MPLS traffic engineering control with optical crossconnects. *IEEE Comm. Mag.*, 39(3):111–116, 2001.

[3] J.-C. Bermond, N. Marlin, D. Peleg, and S. Perennes. Directed virtual path layouts in atm networks. *Theor. Comput. Sci.*, 291(1):3–28, 2003.

[4] R. Callon, P. Doolan, N.Feldman, A. Fredette, and G. Swallow. A framework for multiprotocol label switching, 1997. manuscript.

[5] M. Carson. NIST switch and MPLS-enabled routing algorithms. In *MPLS Forum 2000*, 2000.

[6] G. Chandranmenon and G. Varghese. Trading packet headers for packet processing. *IEEE Transactions on Networking*, 4(2):141–152, 1996.

[7] B. Davie, P. Doolan, and Y. Rekhter. *Switching in IP Networks*. Morgan Kaufmann Publishers Inc., 1998.

[8] T. Eilam, M. Flammini, and S. Zaks. A complete characterization of the path layout construction problem for ATM networks with given hop count and load. *Parallel Processing Letters*, 8(2):207–220, 1998.

[9] O. Gerstel, I. Cidon, and S. Zaks. The layout of virtual paths in ATM networks. *Transactions on Networking*, 4(6):873–884, 1996.

[10] O. Gerstel, I. Cidon, and S. Zaks. Optimal virtual path layout in ATM networks with shared routing table switches. *The Chicago Journal of Theoretical Computer Science*, 3, December 1996.

[11] O. Gerstel, A. Wool, and S. Zaks. Optimal layouts on a chain ATM network. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 83, 1998.

[12] O. Gerstel and S. Zaks. The virtual path layout problem in ATM ring and mesh networks. In *Proc. of the First Conf. on Structure, Information and Communication Complexity (SIROCCO 1994)*, 1994.

[13] R. Handler and M.N. Huber. *Integrated Broadband Networks:an introduction to ATM-based networks*. Addison-Wesley, 1991.

[14] Y. Katsube, K. Nagami, and H. Esaki. Toshiba's router architecture extensions for atm: Overview. rfc 2098. Technical report, 1997.

[15] N. F. Maxemchuk. Routing in the manhattan street network. *Transactions on Communications*, 5:503–512, 1987.

[16] P. Newman, G. Minshall, and L. Huston. Ip switching and gigabit routers. *IEEE Communications Magazine*, 35(1):64–69, 1997.

[17] Y. Rekhter, B. Davie, D. Katz, E. Rosen, G. Swallow, and D. Farinacci. Tag switching architecture overview, 1996. manuscript.

[18] S. Ahn, R. P. Tsang, S.-R. Tong and David H.-C. Du. Virtual path layout design on ATM networks. In *Proc. of The Conference on Computer Communications (INFOCOM 1994)*, pages 192–200, 1994.

[19] A. Vakhutinsky and M. Ball. Fault-tolerant virtual path layout in atm networks. In *Proceedings of the 15th International Teletraffic Conference (ITC 15)*, pages 1031–1041, 1997.

A Appendix

A.1 One-hop path layout

Claim 6 *In the ATM model, A* 1*-hop all-to-one path layout in a tree network (or any other network) requires a vertex load of at most* $n-1$ *labels, and is equal to this number in some networks.*

Proof: In the ATM model each route (which is a pair of a source and a destination) requires a virtual-path and hence a label is needed. Defining all shortest paths to r as virtual paths, results in a maximum vertex load of $n-1$ (the vertex load is at most $n-1$ since there are $n-1$ paths). In a network which is a path on n vertices, mark the last vertex in the path as the designated vertex r. The vertex just before the r on the path has vertex load $n-1$. ▮

Claim 7 *In the MPLS model, and in the Trainet model, a* 1*-hop all-to-one path layout in a tree network requires a vertex load of exactly* 1 *label.*

Proof: Consider the induced tree that is rooted towards r. Consider the switch-paths from all leaves to r. All those paths starting at a certain node v can be labeled by the same label $L(v)$ due to the merging capability of labels. In other words, every node v in the graph requires a single label to describe its path to the root r. ▮

Next we consider the general all-to-all path layout.

Claim 8 *In the ATM model, a* 1*-hop all-to-all path layout in any tree network requires a vertex load of* $\Theta(n^2)$ *labels.*

Proof: Every pair of a source and a destination requires a virtual-path and correspondingly a label. Hence $O(n^2)$ is enough. To show a vertex which achieves such load, we recall a well known fact about trees: In every tree there exists an articulation node z whose removal allows to partition the nodes into two forests, each of which contains $\Omega(n)$ nodes. Since a virtual path is required for each pair of nodes that are not in the same forest, and each such path passes through z, z has $\Omega(n^2)$ labels. ▮

Claim 9 *In the MPLS model, a* 1*-hop all-to-all path layout in a tree network requires vertex load of exactly* $n-1$ *labels.*

Proof: It is clear that every node needs at least $n-1$ labels for the different $n-1$ destinations. For every vertex, u, in the tree, consider the rooted tree directed towards u. We have n such trees. By Claim 7 the all-to-one path layout to the root u requires one label at each node. Hence every node needs to store exactly $n-1$ labels. ▮

Claim 10 *In Trainet Model, let F denote the set of leaves in a tree network. A 1-hop all-to-all path layout in this tree network requires vertex load of $|F|$ labels.*

Proof: Let r be a non-leaf vertex. The vertex r needs routes to all leaves, those are all routes outgoing of r in different directions, and hence at least $|F|$ labels are required. To show that $|F|$ labels are enough, for every leaf f, we build a path layout all-to-one, where f is the designated vertex. By Claim 7 this path layout requires one label, and we have f such trees. Hence in total we get a vertex load of $|F|$. It is left is to show that any shortest path route between two vertices can be described by this path layout. Consider a pair u and v. Root the tree from u outwards. If v if a leaf then let $v' = v$. Otherwise there exists a leaf that is a descendant of v in the rooted tree. Denote this leaf by v'. The route of u to v is a sub-path of the Trainet switch-path between u to v'. Hence all pair paths are induced from the above path layout in Trainet Model, with no need in extra labels. ▮

A.2 An Improved upper bound for the Trainet model

We show how to get the upper bound in the Trainet model even if k is not a constant. This can be done by giving one node labels during at most two phases instead of all $\Theta(k)$. We would like to make sure that once a node gets a label (while defining switch-paths) for phase i, in the next phase it is a root of a tree of type $i+1$, and in the phase after that (if exists), it is not included in a tree of type $i+2$.

It requires the following two changes: 1. While decomposing a tree of type i, T_i, into trees of type $i+1$, the basic algorithm leaves a remainder of T_i which becomes a tree of type $i+1$. Instead, we define a switch-path from the root of T_i to all its children in the remainder, and each of them becomes a tree of type $i+1$. This adds at most d labels to the root, and at most one to any other node. In all cases, $F_k(n,d) = \Omega(d)$ and adding those paths is possible.

2. Consider a node v which received some label in a tree of type i, such that i is not the last type, and there exist also trees of type $i+1$. Let q be the number of switch-paths that need to be concatenated to get to the roots of type $i+1$. Since v got a label, there exists a prefix of the concatenation of paths which contains q switch-paths that connects the main root with v. A prefix may be used as a route in this model and hence, v can be defined to be a type $i+1$ root. We mark all nodes that get a label, and define each of them to be a root of a tree of type $i+1$. This may result in smaller trees of type $i+1$. In terms of running time, the marking can be done during the DFS, and determining the trees of type $i+1$ can be done in another search.

A.3 Proof of Claim 3

Claim *For all odd i, there exists a set $J'_i \subset J_i$ which satisfies the following properties:*

1. $|J'_i| \geq \frac{d}{2}$.
2. *All vertices in J'_i have the same parent node.*
3. $T(i) \cap D(J'_i) = \emptyset$*, i.e. it is impossible to get from the root of T to any of the nodes in the subset J'_i or any nodes in their sub-trees by concatenating at most i routes.*

For even $i > 0$, there exists a set $J'_i \subset J_i$ which satisfies the following properties

1. *All vertices in J'_i have the same ancestor node z in J_{i-1}, i.e. they belong to a single T_3 tree.*
2. $|J'_i| \geq \frac{1}{2}|D(\{z\}) \cap J_i|$*, i.e. J'_i contains at least half of the leaves of some T_3 tree.*
3. $T(i) \cap D(J'_i) = \emptyset$*, i.e. it is impossible to get from the root of T to any of the nodes in the subset J'_i or any nodes in their sub-trees by concatenating at most i routes.*

Proof: By induction. For the basis $i = 1$ we calculate the size of J_1. The size of J_0 is more than $\frac{1}{2}(\frac{n'}{d^{\frac{k}{2}}})^{\frac{2}{k+2}}$, hence $|J_1| > \frac{d}{2}(\frac{n'}{d^{\frac{k}{2}}})^{\frac{2}{k+2}} = \frac{(n'd)^{\frac{2}{k+2}}}{2}$. Since the root has at most $\frac{(n'd)^{\frac{2}{k+2}}}{4}$ labels, at least half of J_1 do not have a route to them or to their sub-tree. By the pigeon-hole principle, there is a node in J_0 who has at least $\frac{d}{2}$ such children.

For odd i, we use the claim given for the even number $i-1$ in the inductive step; Consider J'_{i-1}, the set of leaves in J_{i-1} that each of them is not reachable by a concatenation of $i-1$ routes starting from the root. The root of their common T_3 sub-tree has at most $\frac{(n'd)^{\frac{2}{k+2}}}{8}$ labels. Since $|J_i \cap D(J'_{i-1})| > \frac{d}{4}(\frac{n'}{d^{\frac{k}{2}}})^{\frac{2}{k+2}}$, at least half of the children of J'_{i-1} are not reachable from the root by concatenating i routes (since their direct parents are not reachable by concatenating $i-1$ routes). Hence there exists a set of at least $\frac{d}{2}$ such nodes in J_i that share a parent node.

For even i, we use the claim for the odd number $i-1$ for the inductive step. J'_{i-1} is the set of $\frac{d}{2}$ nodes in J_{i-1} that share a parent and none of them is reachable by concatenating $i-1$ routes starting from the root. Let x be their parent and $\tilde{J}_i$ their successors in J_i. The node x has at most $\frac{(n'd)^{\frac{2}{k+1}}}{8}$ labels and $|\tilde{J}_i| > \frac{d}{4}(\frac{n'}{d^{\frac{k}{2}}})^{\frac{2}{k+2}}$. Since the nodes in J'_{i-1} do not have a path from the root which is a concatenation of at most $i-1$ routes, in order to have a path from the root which is a concatenation of i routes, each node in $\tilde{J}_i$ needs to have a route from x. At least half of $\tilde{J}_i$ do not have a path from the root which is a concatenation of i routes. By the pigeon-hole principle, there exists a T_3 tree rooted at some node in J'_{i-1} in which at least half of the leaves do not have a path. ∎

A.4 Proof of Claim 4

Claim *Given a bidirectional tree. In the Trainet model, it is possible to build a k-hop all-to-all path layout with the same vertex load as k-hop one-to-all path layout in the same model.*

Proof: Given a tree, we fix one node x as root, and use the basic algorithm to build an one-to-all path layout. Consider all nodes Y for which a switch path was defined from x to them. We construct an all-to-one 1-hop path layout to all vertices of Y. This increases the maximum vertex load by at most Y, see Claim 7. Therefore we do not change the order of size of the label-table since each node gets at most Y new labels, and x had Y labels in the first-place. Given two nodes v and u we show how to concatenate at most k routes in order to get from v to u. Let $y \in Y$ by a node such that there exists a concatenation of at most $k-1$ switch paths leading from y to u. Such a path exists due to the following: The output of the algorithm gives a set of switch paths, such that a concatenation of at most k of them leads from x to any node. Y is the set of nodes that are one hop away from x. This means that all nodes are at most $k-1$ hops away from some node in Y. If the concatenation of the one-hop path from v to y and the $k-1$-hop path from y to u is simple (no cycles) we are done. Otherwise, let y' be the lowest common ancestor of u and v. Clearly y is an ancestor of y'. We use the prefix (v to y') of the path from v to y, and the suffix (from y' to u) of the (at most) $k-1$-hop path from y to u. This is possible in the Trainet model and gives the desired result. ∎