Labeling Schemes with Queries

Amos Korman [*]   Shay Kutten [†]

Abstract

Recently, quite a few papers studied methods for representing network properties by assigning *informative labels* to the vertices of a network. Consulting the labels given to any two vertices u and v for some function f (e.g. "distance(u, v)") one can compute the function (e.g. the graph distance between u and v). Some very involved lower bounds for the sizes of the labels were proven.

In this paper, we demonstrate that such lower bounds are very sensitive to the number of vertices consulted. That is, we show several almost trivial constructions of such labeling schemes that beat the lower bounds by large margins. The catch is that one needs to consult the labels of *three* vertices instead of two. We term our generalized model *labeling schemes with queries*.

Additional contributions are several extensions. In particular, we show that it is easy to extend our schemes for tree to work also in the the dynamic scenario. We also demonstrate that the study of the queries model can help in designing a scheme for the traditional model too. Finally, we demonstrate extensions to the non-distributed environment. In particular, we show that one can preprocess a general weighted graph using almost linear space so that flow queries can be answered in almost constant time.

Keywords: Labeling schemes, routing schemes, distance queries.
Submitted as a regular presentation.

[*]Contact person. Information Systems Group, Faculty of IE&M, The Technion, Haifa, 32000 Israel. E-mail:`pandit@tx.technion.ac.il`. Supported in part at the Technion by an Aly Kaufman fellowship.

[†]Information Systems Group, Faculty of IE&M, The Technion, Haifa, 32000 Israel. E-mail: `kutten@ie.technion.ac.il`. Fax: +972 (4) 829 5688. Phone: +972 (4) 829 4505. Supported in part by a grant from the Israeli Ministry for Science and Technology.

1 Introduction

Background: Network representations play a major role in many domains of computer science, ranging from data structures, graph algorithms, and combinatorial optimization to databases, distributed computing, and communication networks. In most traditional network representations, the names or identifiers given to the vertices betray no useful information, and they serve only as pointers to entries in the data structure, which forms a *global* representation of the network. Recently, quite a few papers studied methods for representing network properties by assigning *informative labels* to the vertices of the network (see e.g., [43, 9, 32, 39, 51, 5, 24]).

Let f be a function on pairs of vertices (e.g., distance). Informally, the goal of an f-labeling scheme is to label the vertices of a graph G in such a way that for every two vertices $u, v \in G$, the value $f(u, v)$ (e.g., the distance between u and v) can be inferred by merely inspecting the labels of u and v. Of course, this can be done trivially using labels that are large enough (e.g., every label includes the description of the whole graph). Therefore, the main focus of the research concerning labeling schemes is to minimize the amount of information (the sizes of the labels) required. Informally, an f-labeling scheme can be viewed as a way of distributing the graph structure information concerning f to the vertices of the graph, using small chunks of information per vertex.

Rather involved proofs were introduced to lower bound the sizes of such labeling schemes. The main contribution of this paper is the demonstration that these lower bounds are very sensitive to the model uses. Intuitively, if the labels of three vertices can be consulted (rather than two, such as u and v above), that it is very easy to reduce the sizes significantly, much below the previous lower bounds. Moreover, our query labeling schemes can be obtained using very simple methods, sometimes trivial.

Elaborating somewhat more (formal definitions appear in Section 2) this paper introduces the notion of *f-labeling schemes with queries* which generalizes the notion of f-labeling schemes. The idea is to distribute the global information (relevant to f) to the vertices, in such a way that $f(u, v)$ can be inferred by inspecting not only the labels of u and v but possibly the labels of additional vertices. We note that all the constructions given in this paper calculate $f(u, v)$ by inspecting the labels of three vertices (u and v above, and some w). That is, given the labels of u and v, we first find a vertex w and then consult its label to derive $f(u, v)$. However, in the concluding section we discuss generalizations to inspecting additional labels.

We also show several additional extensions. One extension is meant to demonstrate an advantage of the simplicity of the design of labeling schemes with queries. It helped us to simplify the design of a labeling scheme for the traditional model (with no queries). We did that in two steps: first we designed the simple scheme with queries, and then "simulated" this scheme in the old model (with no queries). (The "simulation" had some associated cost, which made the resulting scheme work for an approximation function f, rather than for the exact function we would have liked).

A second extension of the result is to dynamic scenarios. We note that most previous research concerning distributed network representations considered the *static* scenario, in which the topology of the underlying network is fixed. This is probably due to the fact that designing for the dynamic scenario is more complex. Some recent papers did tackle task of labeling dynamic networks in a distributed fashion. Such labeling methods should, of course, be dynamic too. Indeed, the designs in these recent paper tend to be harder and more complex. We show that the effect of introducing a query to the dynamic case is similar to the effect on the static case. That is- the sizes can be reduced considerably, and the construction of the schemes is rather easy (though somewhat more complex than in the static case). To do that, we modify the model translation methods of [38] and [42], and then use them to extend our static labeling schemes with queries on trees to the dynamic scenario. We then show that the sizes of the resulted schemes are smaller than those of the schemes for the older model. The reduction in the label size is similar to the reduction in the static case.

In a final extension, we show that our methods are also useful in the non-distributed environment.

Related work: In this subsection we mostly survey results concerning labeling schemes (with no queries). However, let us first mention an area of research (namely, overlay and Peer to Peer networks) that may serve as a practical motivation for our work, and for some other studies concerning labeling schemes. We stress, though, that the main motivation for this paper is theoretical.

When the third vertex w (mentioned above) is near by to u, it may be quite cheap for u to access the main memory at w, sometimes even cheaper than consulting the disk at u itself. See, for example [47, 37]. Indeed, some of our schemes below are based on such a "near by" w. Even when w is remote, accessing it may be cheap in some overlay networks. The main overhead there is finding w (which can be done in our constructions by u using v's label) and creating the connection to it. Such models are presented explicitly, e.g. in [34, 16, 2, 54], where such remote accesses are used to construct and to use overlay data structures. Famous overlay data structures that can fit such models appear for example in [56, 63, 48, 26, 49]. In some of these overlay networks, a vertex w is addressed by its contents. This may motivate common labeling schemes that assume content addressability.

Implicit labeling schemes were first introduced in [11, 12, 43]. Labeling schemes supporting the adjacency and ancestry functions on trees were investigated in [43, 9, 44, 7, 8].

Distance labeling schemes were studied in [50, 40, 32, 31, 29, 44, 59, 18, 4]. In particular, [50] showed that the family of n vertex weighted trees with integer edge capacity of at most W enjoys a scheme using $O(\log^2 n + \log n \log W)$-bit labels. This bound was proven in [32] to be asymptotically optimal.

Labeling schemes for routing on trees were investigated in a number of papers [55, 62, 15, 21, 23] until finally optimized in [24, 25, 61]. For the *designer port* model, in which the designer of the scheme can freely enumerate the port numbers of the nodes, [24] shows how to construct a routing scheme using labels of $O(\log n)$ bits on n-node trees. In the *adversary port* model, in which the port numbers are fixed by an adversary, they show how to construct a routing scheme using labels of $O(\log^2 n/\log\log n)$ bits on n-node trees. In [25] they show that both label sizes are asymptotically optimal. Independently, a routing scheme for trees using $(1 + o(1)) \log n$-bit labels was introduced in [61] for the designer port model.

Two variants of labeling schemes supporting the nearest common ancestor (NCA) function in trees appear in the literature. In an id-NCA labeling schemes, the vertices of the input graph are assumed to have disjoint identifiers (using $O(\log n)$ bits) given by an adversary. The goal of an id-NCA labeling scheme is to label the vertices such that given the labels of any two vertices u and v, one can find the identifier of the NCA of u and v. Static labeling schemes on trees supporting the separation level and id-NCA functions were given in [51] using $\Theta(\log^2 n)$-bit labels. The second variant considered is the label-NCA labeling scheme, whose goal is to label the vertices such that given the labels of any two vertices u and v, one can find the label (and not the pre-given identifier) of the NCA of u and v. In [5] they present a label-NCA labeling scheme on trees enjoying $\Theta(\log n)$-bit labels.

In [39] they give a labeling scheme supporting the flow function on n-node general graphs using $\Theta(\log^2 n + \log n \log W)$-bit labels, where W is the maximum capacity of an edge. They also show a labeling scheme supporting the k-vertex-connectivity function on general graphs using $O(2^k \log n)$-bit labels. See [30] for a survey on (static) labeling schemes.

Most of the research concerning labeling schemes in the dynamic settings considered the following two dynamic models on tree topologies. In the *leaf-dynamic* tree model, the topological event that may occur is that a leaf is either added to or removed from the tree. In the *leaf-increasing* tree model, the only topological event that may occur is that a leaf joins the tree.

The study of dynamic distributed labeling schemes was initiated in [42, 41]. In [42], a dynamic labeling scheme is presented for distances in the leaf-dynamic tree model with $O(\log^2 n)$ label size and $O(\log^2 n)$ amortized message complexity, where n is the current tree size. β-approximate distance labeling schemes (in which, given two labels, one can infer a β-approximation to the distance between the corresponding nodes) are presented [41]. Their schemes apply for dynamic models in which the tree topology is fixed but the edge weights may change.

Two general translation methods for extending static labeling schemes on trees to the dynamic setting are considered in the literature. Both approaches fit a number of natural functions on trees, such as ancestry, routing, label-NCA, id-NCA etc. Given a static labeling scheme on trees, in the leaf-increasing tree model, the resulting dynamic scheme in [42] incurs overheads (over the static scheme) of $O(\log n)$ in both the label size and the communication complexity. Moreover, if an upper bound n_f on the final number of vertices in the tree is known in advance, the resulting dynamic scheme in [42] incurs overheads (over the static scheme) of $O(\log^2 n_f / \log\log n_f)$ in the label size and only $O(\log n/\log\log n)$ in the communication complexity. In the leaf-dynamic tree model there is an extra additive factor of $O(\log^2 n)$ to the amortized message complexity of the resulted schemes.

In [38], it is shown how to construct for many functions $k(x)$, a dynamic labeling scheme in the leaf-increasing tree model extending a given static scheme, such that the resulting scheme incurs overheads (over the static scheme) of $O(\log_{k(n)} n)$ in the label size and $O(k(n)\log_{k(n)} n)$ in the communication complexity. As in [42], in the leaf-dynamic tree model there is an extra additive factor of $O(\log^2 n)$ to the amortized message complexity of the resulted schemes. In particular, by setting $k(n) = n^{\epsilon}$, dynamic labeling schemes are obtained with the same asymptotic label size as the corresponding static schemes and sublinear amortized message, namely, $O(n^{\epsilon})$.

1.1 Our contribution

We introduce the notion of f-labeling schemes with queries that is a natural generalization of the notion of f-labeling schemes. Using this notion we demonstrate that by increasing slightly the number of vertices whose labels are inspected, the size of the labels decreases considerably. Specifically, we inspect the labels of 3 vertices instead of 2, that is, we use a single *query*. In particular, we show that there exist simple labeling schemes with one query supporting the distance function on n-node trees as well as the flow function on n-node general graphs with label size $O(\log n + \log W)$, where W is the maximum (integral) capacity of an edge. (We note that the lower bound for labeling schemes without queries for each of these problems is $\Omega(\log^2 n + \log n \log W)$ [32, 39].) We also show that there exists a labeling scheme with one query supporting the id-NCA function on n-node trees with label size $O(\log n)$. (The lower bound for schemes without queries is $\Omega(\log^2 n)$ [51].) In addition, we show a routing labeling scheme with one query in the fixed-port model using $O(\log n)$-bit labels, while the lower bound (see [25]) for the case of no queries is $\Omega(\frac{\log^2 n}{\log\log n})$. We note that all the schemes we introduce have asymptotically optimal label size for schemes with one query. The matching lower bound proofs are straightforward in most of the cases, and we present the proofs in the remaining cases. Moreover, most of the results are obtained by simple constructions, which strengthens the motivation for this model.

We then show several extensions that are somewhat more involved. In particular, we show that our labeling schemes with queries on trees can be extended to the dynamic scenario using model translation methods based on those of [42, 38]. In order to save in the message complexity, we needed to make some adaptations to those methods, as well as to one of the static routing schemes of [24]. Second, we show that the study of the queries model can help with the traditional model too. That is, using ideas from our routing labeling scheme with one query, we show how to construct a 3-approximation routing scheme *without queries* for unweighted trees in the fixed-port model with $\Theta(\log n)$-bit labels.

Finally, we turn to a non-distributed environment and demonstrate similar constructions. That is, first, we show a simple method to transform previous results on NCA queries on static and dynamic trees in order to support also distance queries. Then, we show that one can preprocess a general weighted graph using almost linear space so that flow queries can be answered in almost constant time.

2 Preliminaries

Let T be a tree and let v be a vertex in T. Let $deg(v)$ denote the degree of v. For a non-root vertex $v \in T$, let $p(v)$ denote the parent of v in T. In the case where the tree T is weighted (respectively, unweighted), the *depth* of a vertex is defined as its weighted (resp., unweighted) distance to the root. The *nearest common ancestor* of u and w, $NCA(u, w)$, is the common ancestor of both u and w of maximum depth. Let $\mathcal{T}(n)$ denote the family of all n-node unweighted trees. Let $\mathcal{T}(n, W)$ (respectively, $\mathcal{G}(n, W)$) denote the family of all n-node weighted trees (resp., connected graphs) with (integral) edge weights bounded from above by W.

Incoming and outgoing links from every node are identified by so called *port-numbers*. We distinguish between the following two variants of port models regarding routing schemes. In the *designer port* model the designer of the scheme can freely assign the port numbers of each vertex (as long as these port numbers are unique), and in the *fixed-port* model the port numbers at each vertex are assigned by an adversary. We assume that each port number is encoded using $O(\log n)$ bits.

We consider the following functions which are applied on pairs of vertices u and v in a graph $G = \langle V, E\rangle$. The detailed definitions of these functions are given in the appendix. (1) **flow** (maximum legal flow between u and v), (2) **distance** (either weighted, or unweighted), (3) **routing** (the port in u to the next vertex towards v). If the graph is a tree T then we consider also the following functions: (4) **separation level** (depth of $NCA(u, v)$), (5) **id-NCA**, (6) **label-NCA**. In (5) above, it is assumed that identities containing $O(\log n)$ bits are assigned to the vertices by an adversary, and $id - NCA(u, v)$ is the identity of $NCA(u, v)$. In (6) above, it is assumed that each vertex can freely select its own identity (as long as all identities remain unique). In this case, the identities may also be referred to as labels.

Labeling schemes and c-query labeling schemes: Let f be a function defined on pairs of vertices. An *f-labeling scheme* $\pi = \langle \mathcal{M}, \mathcal{D}\rangle$ for a family of graphs $\mathcal{F}$ is composed of the following components:

1. A *marker* algorithm $\mathcal{M}$ that given a graph $G \in \mathcal{F}$, assigns a label $\mathcal{M}(v)$ to each vertex $v \in G$.

2. A (polynomial time) *decoder* algorithm $\mathcal{D}$ that given the labels $\mathcal{M}(u)$ and $\mathcal{M}(v)$ of two vertices u and v in some graph $G \in \mathcal{F}$, outputs $f(u, v)$.

The most common measure used to evaluate a labeling scheme $\pi = \langle \mathcal{M}, \mathcal{D}\rangle$, is the *label size*, i.e., the maximum number of bits used in a label $\mathcal{M}(v)$ over all vertices v in all graphs $G \in \mathcal{F}$.

Let c be some constant integer. Informally, in contrast to an f-labeling scheme, in a c-query f-labeling scheme, given the labels of two vertices u and v, the decoder may also consult the labels of c other vertices. More formally, a *c-query f-labeling scheme* $\varphi = \langle \mathcal{M}, Q, \mathcal{D}\rangle$ is composed of the following components:

1. A *marker* algorithm $\mathcal{M}$ that given a graph $G \in \mathcal{F}$, assigns a label $\mathcal{M}(v)$ to each vertex $v \in G$. This label is composed of two sublabels, namely, $\mathcal{M}^{index}(v)$ and $\mathcal{M}^{data}(v)$, where it is required that the index sublabels are unique, i.e., for every two vertices v and u, $\mathcal{M}^{index}(v) \neq \mathcal{M}^{index}(u)$. (In other words, the index sublabels can serve as identities.)

2. A (polynomial time) *query* algorithm Q that given the labels $\mathcal{M}(u)$ and $\mathcal{M}(v)$ of two vertices u and v in some graph $G \in \mathcal{F}$, outputs $Q(\mathcal{M}(u), \mathcal{M}(v))$ which is a set containing the indices (i.e., the first sublabels) of c vertices in G.

3. A (polynomial time) *decoder* algorithm $\mathcal{D}$ that given the labels $\mathcal{M}(u)$ and $\mathcal{M}(v)$ of two vertices u and v and the labels of the vertices in $Q(\mathcal{M}(u), \mathcal{M}(v))$, outputs $f(u, v)$.

As in the case of f-labeling schemes, we evaluate a c-query f-labeling scheme $\varphi = \langle \mathcal{M}, Q, \mathcal{D}\rangle$ by its *label size*, i.e, the maximum number of bits used in a label $\mathcal{M}(v)$ over all vertices v in all graphs $G \in \mathcal{F}$.

We note that all the schemes in this paper use $c = 1$, increasing the number of labels used by a smallest constant, while the size of the maximum label drops significantly in the order of magnitude. Let us comment also that clearly, since the index sublabels must be disjoint, any c-query f-labeling scheme on any family of n-node graphs must have label size $\Omega(\log n)$. See Section 7 for alternative definition for query labeling schemes.

2.1 Routing schemes and β-approximation routing schemes

A *routing scheme* is composed of a *marker algorithm* $\mathcal{M}$ for assigning each vertex v of a graph G with a label $\mathcal{M}(v)$, coupled with a *router* algorithm $\mathcal{R}$ whose inputs are the header of a message, $\mathcal{M}(v)$ and the label $\mathcal{M}(y)$ of a destination vertex y. If a vertex x wishes to send a message to vertex y, it first prepares and attaches a header to the message. Then the router algorithm x outputs a port of x on which the message is delivered to the next vertex. This is repeated in every vertex until the message reaches the destination vertex y. Each intermediate vertex u on the route may replace the header of the message with a new header and may perform a local computation. The requirement is that the weighted length of resulting path connecting x and y is the same as the distance between x and y in G. For a constant β, a β-approximation routing scheme is the same as a routing scheme except that the requirement is that the length of resulting route connecting x and y is a β-approximation for the distance between x and y in G.

In addition to the label size, we also measure a routing scheme (and a β-approximation routing scheme) by the *header size*, i.e., the maximum number of bits used in a header of a message.

3 Labeling schemes with one query

In this section we demonstrate that the query model allows for significantly shorter labels. In particular, we describe simple 1-query labeling schemes with labels that beat the lower bounds in the following well studied cases: for the family of n-node trees, schemes supporting the routing (in the fixed-port model), distance, separation level, and the id-NCA functions; for the family of n-node general graphs, a scheme supporting the flow function. We note that all the schemes we present use asymptotically optimal labels.

Most of the 1-query labeling schemes obtained in this section use the label-NCA labeling scheme $\pi_{NCA} = \langle \mathcal{M}_{NCA}, \mathcal{D}_{NCA} \rangle$ described in [5]. Given an n-node, the marker algorithm $\mathcal{M}_{NCA}$ assigns each vertex v a distinct label $\mathcal{M}_{NCA}(v)$ using $O(\log n)$ bits. Given the labels $\mathcal{M}_{NCA}(v)$ and $\mathcal{M}_{NCA}(u)$ of two vertices v and u in the tree, the decoder $\mathcal{D}_{NCA}$ outputs the label $\mathcal{M}_{NCA}(w)$.

3.1 Id-NCA function in trees

We first describe a 1-query labeling scheme $\varphi_{id-NCA} = \langle \mathcal{M}_{id-NCA}, Q_{id-NCA}, \mathcal{D}_{id-NCA} \rangle$ that demonstrates how easy it is to support the id-NCA function on $\mathcal{T}(n)$ using one query and $O(\log n)$-bit labels. (Recall that the lower bound on schemes without queries is $\Omega(\log^2 n)$ [51].)

Informally, the idea behind φ_{id-NCA} is to have the labels of u and v (their first sublabels) be the labels given by the label-NCA labeling scheme $\pi_{NCA}(v)$. Hence, they are enough for the query algorithm to find the π_{NCA} label of their nearest common ancestor w. Then, the decoder algorithm finds w's identity simply in the second sublabel of w.

Let us now describe the 1-query labeling scheme φ_{id-NCA} more formally. Given a tree T, recall that it is assumed that each vertex v is assigned a unique identity $id(v)$ by an adversary and that each such identity is composed of $O(\log n)$ bits. The marker algorithm $\mathcal{M}_{id-NCA}$ labels each vertex v with the label $\mathcal{M}_{id-NCA}(v) = \langle \mathcal{M}^{index}_{id-NCA}(v), \mathcal{M}^{data}_{id-NCA}(v) \rangle = \langle \mathcal{M}_{NCA}(v), id(v) \rangle$. Given the labels $\mathcal{M}_{id-NCA}(v)$ and $\mathcal{M}_{id-NCA}(u)$ of two vertices v and u in the tree, the query algorithm Q_{id-NCA} uses the decoder $\mathcal{D}_{NCA}$

applied on the corresponding first sublabels to output the sublabel $\mathcal{M}_{id-NCA}^{index}(w) = \mathcal{M}_{NCA}(w)$, where w is the NCA of v and u. Given the labels $\mathcal{M}_{id-NCA}(v)$, $\mathcal{M}_{id-NCA}(u)$ and $\mathcal{M}_{id-NCA}(w)$ where w is the NCA of v and u, the decoder $\mathcal{D}_{id-NCA}$ simply outputs the second sublabel of w, i.e., $\mathcal{M}_{id-NCA}^{data}(w) = id(w)$. The fact that φ_{id-NCA} is a correct 1-query labeling scheme for the id-NCA function on $\mathcal{T}(n)$ follows from the correctness of the label-NCA labeling scheme π_{NCA}. Since the label size of $\pi_{NCA}(v)$ is $O(\log n)$ and since the identity of each vertex v is encoded using $O(\log n)$ bits, we obtain that the label size of φ_{id-NCA} is $O(\log n)$. As mentioned before, since the index sublabels must be disjoint, any query labeling scheme on $\mathcal{T}(n)$ must have label size $\Omega(\log n)$. The following lemma follows.

Lemma 3.1 *The label size of a 1-query id-NCA labeling scheme on $\mathcal{T}(n)$ is $\Theta(\log n)$.*

3.2 Distance and separation level in trees

The above method can be applied for other functions. For example, let us now describe 1-query labeling schemes $\varphi_{sep-level}$ and φ_{dist} supporting the distance and separation level functions respectively on $\mathcal{T}(n, W)$. Both our scheme have label size $\Theta(\log n + \log W)$. Recall that any labeling scheme (without queries) supporting either the distance function or the separation level function on $\mathcal{T}(n, W)$ must have size $\Omega(\log^2 n + \log n \log W)$, [32, 51]. We first show the following claim.

Claim 3.2 *Let c be a constant. Any c-query labeling scheme supporting either the separation level function or the distance function on $\mathcal{T}(n, W)$ must have label size $\Omega(\log W + \log n)$.*

Proof: As mentioned before, any query labeling scheme on $\mathcal{T}(n)$ must have label size $\Omega(\log n)$. We show the proof for the distance function. Similar proof holds for the separation level function. First note that we may assume that $W \geq ((c + 2) \cdot n^c)^2$, otherwise, the lower bound trivially follows since $\Omega(\log n) = \Omega(\log n + \log W)$. Let $\varphi = \langle \mathcal{M}, Q, \mathcal{D} \rangle$ be any c-query labeling scheme supporting the distance function. Let P be an n-node path rooted at one of its end-nodes r. Let v be the (only) child of r. For every $1 \leq i \leq W$, let P_i be the path P such that the edge (r, v) has weight i and all other edges have weight 1. For every $1 \leq i \leq W$ and every vertex $u \in P_i$, let $L_i(u)$ denote the label given to u by the marker algorithm $\mathcal{M}$ applied on P_i. For every $1 \leq i \leq W$, let S_i be the set of c vertices given by the query algorithm applied on the labels of r and v in P_i, i.e., $S_i = Q(L_i(r), L_i(v))$. Since there are at most n^c sets of c vertices, there exists a set $X \subset \{1, 2, \cdots W\}$ such that $|X| \geq W/n^c$ and for every $i, j \in X$, $S^i = S^j$. Let S denote the set of c vertices such that for every $i \in X$, $S^i = S$. For each $i \in X$, given the labels $L_i(r)$, $L_i(v)$ and the labels of the vertices in S assigned by $\mathcal{M}$ applied on P_i, the decoder outputs i, which is the distance between r and v in P_i. Since $|X| \geq W/((c + 2) \cdot n^c)$, there must exists a vertex in $u \in \{r, v\} \cup S$ such that the set $\{L_i(u) \mid i \in X\}$ contains $\frac{W}{(c+2)\cdot n^c}$ values. Therefore, there must exist an $i \in X$ such that $L_i(u)$ contains $\log W - \log((c + 2) \cdot n^c) > (\log W)/2$ bits. The claim follows. ▮

We now show how to construct 1-query labeling schemes supporting the separation level and distance functions using similar method to the one described in Subsection 3.1. Both schemes are based on keeping the depth of a vertex in its data sublabel (instead of its identity). The correctness of the 1-query labeling scheme supporting the distance function is based on the following equation.

$$d(v, u) = depth(v) + depth(u) - 2 \cdot depth(NCA(v, u)). \tag{1}$$

The description of these schemes as well as the proof of the following lemma is deferred to the appendix.

Lemma 3.3 *The label size of a 1-query labeling scheme supporting either the separation-level or the distance function on $\mathcal{T}(n, W)$ is $\Theta(\log n + \log W)$.*

3.3 Routing in trees using one query

As mentioned in before, any 1-query routing labeling scheme on $\mathcal{T}(n)$ must have label size $\Omega(\log n)$. In this subsection, we establish a 1-query routing labeling scheme φ_{fix} in the fixed-port model using $O(\log n)$-bit labels.

In [24], they give a routing scheme $\pi_{des} = \langle \mathcal{M}_{des}, \mathcal{D}_{des} \rangle$ for the designer port model in $\mathcal{T}(n)$. Given a tree $T \in \mathcal{T}(n)$, for every vertex $v \in T$, and every neighbor u of v, let $port_{des}(v,u)$ denote the port number (assigned by the designer of the routing scheme π_{des}) leading from v to u. In particular, the port number leading from each non-root vertex v to its parent $p(v)$ is assigned the number 1, i.e., $port_{des}(v,p(v)) = 1$. Given the labels $\mathcal{M}_{des}(v)$ and $\mathcal{M}_{des}(w)$ of two vertices v and w in T, the decoder $\mathcal{D}_{des}$ outputs the port number $port_{des}(v,u)$ at v leading from v to the next vertex u on the shortest path connecting v and w.

Let T be an n-node tree. We refer to a port number assigned by the designer of the routing scheme π_{des} as a *designer port number* and to a port number assigned by the adversary as an *fixed-port number.* Let *port* be some port of a vertex in the fixed-port model. Besides having a fixed-port number assigned by the adversary, we may also consider *port* as having a designer port number, the number that would have been assigned to it had we been in the designer port model. For a port leading from vertex v to vertex u, let $port_{fix}(v,u)$ denote its fixed-port number and let $port_{des}(v,u)$ denote its designer port number.

We now describe our 1-query routing labeling scheme $\varphi_{fix} = \langle \mathcal{M}_{fix}, Q_{fix}, \mathcal{D}_{fix} \rangle$ which operates in the fixed-port model. Given a a tree $T \in \mathcal{T}(n)$ and a vertex $v \in T$, the index sublabel of v is composed of two fields, namely, $\mathcal{M}^{index}(v) = \langle \mathcal{M}^{index}_1(v), \mathcal{M}^{index}_2(v) \rangle$ and the data sublabel of v is composed of three fields, namely, $\mathcal{M}^{data}(v) = \langle \mathcal{M}^{data}_1(v), \mathcal{M}^{data}_2(v), \mathcal{M}^{data}_3(v) \rangle$. If v is not the root then the index and data sublabels of v are

$$\mathcal{M}^{index}(v) = \langle \mathcal{M}_{des}(p(v)) \, , \, port_{des}(p(v),v) \rangle \quad , \quad \mathcal{M}^{data}(v) = \langle \mathcal{M}_{des}(v) \, , \, port_{fix}(p(v),v) \, , \, port_{fix}(v,p(v)) \rangle.$$

Note that we use the designer port number as a part of the label in the fixed-port model. Moreover, the designer port number at the parent is used to label the child in the fixed-port model. Also note that the index sublabel is unique, since $\mathcal{M}_{des}(x)$ must be unique for π_{des} to be a correct routing scheme.

The index sublabel of the root r of T is $\langle 0,0 \rangle$ and the data sublabel of r is $\mathcal{M}^{data}(r) = \langle \mathcal{M}_{des}(r), 0, 0 \rangle$. Note that since the labels given by the marker algorithm $\mathcal{M}_{des}$ are unique, the index sublabels of the vertices are unique.

Given the labels $\mathcal{M}(v)$ and $\mathcal{M}(w)$ of two vertices v and w, the decoder $\mathcal{D}$ first checks whether $\mathcal{D}_{des}(\mathcal{M}^{data}_1(v), \mathcal{M}^{data}_1(w)) = 1$, i.e., whether the next vertex on the shortest path leading from v to w is v's parent. In this case, the query algorithm is ignored and the decoder $\mathcal{D}_{fix}$ simply outputs $\mathcal{M}^{data}_3(v)$ which is the (fixed) port number at v leading to its parent. Otherwise, the query algorithm Q_{fix} outputs $\langle \mathcal{M}^{data}_1(v), \mathcal{D}_{des}(\mathcal{M}^{data}_1(v), \mathcal{M}^{data}_1(w)) \rangle = \langle \mathcal{M}^{data}_1(v), \mathcal{D}_{des}(\mathcal{M}_{des}(v), \mathcal{M}_{des}(w)) \rangle$ which is precisely the index sublabel of u, the next vertex on the shortest path leading from v to w (and a child of v), i.e., $\langle \mathcal{M}^{data}_1(v), port_{des}(v,u) \rangle$. Therefore, given labels $\mathcal{M}_{fix}(v)$, $\mathcal{M}_{fix}(w)$ and label $\mathcal{M}_{fix}(u)$, the decoder $\mathcal{D}_{fix}$ outputs $\mathcal{M}^{data}_2(u)$ which is the desired port number $port_{fix}(v,u)$. Since the label size of π_{des} is $O(\log n)$ and since each port number is encoded using $O(\log n)$ bits, we obtain the following lemma.

Lemma 3.4 *In the fixed-port model, the label size of a 1-query routing scheme on $\mathcal{T}(n)$ is $\Theta(\log n)$.*

3.4 Flow in general graphs

We now consider the family $\mathcal{G}(n,W)$ of connected n-node weighted graphs with maximum edge capacities W, and present a 1-query flow labeling scheme φ_{flow} for this family using $O(\log n + \log W)$-bit labels. Recall that any labeling scheme (without queries) supporting the flow function on $\mathcal{G}(n,W)$ must have size $\Omega(\log^2 n + \log n \log W)$ [39]. The proof of the following lemma is sketched in the appendix.

Lemma 3.5 *The label size of a 1-query flow labeling scheme on $\mathcal{G}(n,W)$ is $\Theta(\log n + \log W)$.*

4 A 3-approximation routing scheme in the fixed-port model

By applying the method described in Subsection 3.3 to the traditional model, we now show how to construct a 3-approximation routing scheme (without queries) on $\mathcal{T}(n)$. Our 3-approximation routing labeling scheme π_{approx} operates in the fixed-port model and has label size and header size $O(\log n)$. Recall that any (precise) routing scheme on $\mathcal{T}(n)$ must have label size $\Omega(\log^2 n/\log\log n)$ [25]. We note that our ideas for translating routing schemes from the designer port model to the fixed-port model implicitly appear in [1], however, a 3-approximation routing scheme (without queries) on $\mathcal{T}(n)$ is not explicitly constructed there. The description of the 3-approximation routing labeling scheme π_{approx} as well as the proof of the following lemma is deferred to the Appendix.

Lemma 4.1 *π_{approx} is a correct 3-approximation routing scheme on $\mathcal{T}(n)$ operating in the fixed port model. Moreover, its label size and header size are $\Theta(\log n)$.*

5 Adapting the 1-query schemes on trees to the dynamic setting

In this section we show how to translate our 1-query labeling schemes on trees to the dynamic settings, i.e, to the leaf-increasing and leaf-dynamic tree models, [38, 42] (see also "Related work" in Section 1). In a dynamic scheme, the marker protocol updates the labels after every topological change. We show that the reduction in the label sizes obtained by introducing a single query in the dynamic scenario is similar to the reduction in the static case.

To describe the adaptation fully, we need to give many details about the methods of [42, 38, 24]. Unfortunately, this is not possible in this extended abstract. In our construction below, we just state and then use some facts about these methods. These facts can be proven easily given these methods.

The initial idea is to apply the methods introduced in [42, 38] to convert labeling schemes for static networks to work on dynamic networks too. Unfortunately, we cannot do this directly, since these methods were designed for traditional labeling schemes and not for 1-query labeling schemes.

The next idea is to perform the conversion indirectly. That is, recall (Section 3) that our 1-query labeling schemes utilize components that are schemes in the traditional model (with no queries). That is, some utilize π_{NCA}, the label-NCA labeling scheme of [5] and some utilize π_{des}, the routing scheme of [24]. Hence, one can first convert these components to the dynamic setting. Second, one can attempt to use the resulted dynamic components in a similar way that we used the static components in Section 3. This turns out to be simple in the cases of the distance, separation level and id-NCA functions, but more involved in the case of the routing function.

In the case of the distance and separation level functions, π_{NCA} is used together with the depth of each vertex in its label. Hence, in the dynamic case, we need to update the depth with every topological change. In the dynamic models above, this uses only a constant number of messages per topological change, since once a vertex is added, its depth remains the same.

However, when trying to translate our 1-query routing labeling scheme to the dynamic settings, things turn out to be more difficult. Informally, the reason is the following. In φ_{fix}, our static 1-query routing labeling scheme, each non-root vertex 'knows' the label assigned to its parent by another static routing scheme π_{des} (see Section 3). When this scheme is made dynamic, the natural translation that enables the child to continue to 'know' the label of its parent is the following:
Notification: whenever the label of a non-leaf vertex changes, it notifies the new label to all the children. When a child receives such a notification message, it updates its φ_{fix} label accordingly (see Section 3).

In order to have an efficient translation, we need to account for the messages used for the above notification. This turns out to be relatively cheap when using the translation method of [42]. The relevant facts about [42] is the following: Theorem 4.16 and the fact that the label of a vertex v changes only as a part of changes in all the vertices of some subtree T'. These changes in T' involve $\Omega(|T'|)$ messages anyhow (not including the new notification messages required from v to its children). It so happened that in the method of [42], whenever a vertex v is included in T', all its children are included in T' as well. Hence, the cost of the new notification messages can be amortized on the cost of the messages sent anyhow in T' to perform the changes. Hence, we can (1) use the transformation of [42] to transform π_{des} to be dynamic, and then (2) use the new notification messages to implement φ_{fix} using the resulted dynamic version of π_{des}. This yields a dynamic routing scheme with 1-query enjoying the complexities promised in [42] (see part 2 in Theorem 5.1 below). It is still left to ensure parts 1 and 3 of Theorem 5.1 below. For that we need to use the transformation of [38], which turns out to be less easy.

To prove parts 1 and 3 of Theorem 5.1, the relevant fact about [38] consists of Theorems 1 and 2 of [38], and of the list of the cases where messages are sent. The latter is needed because we would like to show that we can amortize the cost of the new messages we now send, on the messages sent anyhow by the method of [38]. Unlike the case of [42], here messages are sent over a subtree that may include some vertex v, but not all of its children. Recall that our modifications includes the notification: sending messages from v, whose label was changed, to all of its children. When v has children outside of T', we cannot amortize the cost of the notification messages on the cost of the messages sent anyhow by the method of [38]. It turns out, however, that the only vertex in T' such that not all of its children are necessarily in T' is the root r' of T'. Therefore, the notifications messages sent by a vertex $v \in T'$, where $v \neq r'$ do not increase the cost of the messages sent anyhow by the method of [38]. The only thing left to consider is the case where r' changes its label in the dynamic labeling scheme given by the method of [38] applied on π_{des}, the static routing scheme of [24]. To bound the number of notification messages sent by r', we make sure that the label of r' changes only when r' has at most 1 child. This is done by modifying the static routing scheme π_{des} such that when applying π_{des} on any tree, the label given by π_{des} to the root of the tree is always the same.

It therefore follows that the cost of the remaining notification messages can be amortized on the cost of the messages sent anyhow by [38]. The necessary modifications of the schemes, as well as the proof of the following theorem, appear in the appendix.

Theorem 5.1 *Consider the fixed-port model and let $k(x)$ be any function satisfying that $k(x)$, $\log_{k(x)} x$ and $\frac{k(x)}{\log k(x)}$ are nondecreasing functions and that $k(\Theta(x)) = \Theta(k(x))$.*[1] *There exist dynamic 1-query labeling schemes supporting the distance, separation level, id-NCA and routing functions on trees with the following complexities.*

1. *In the leaf-increasing tree model, with label size $O(\log_{k(n)} n \cdot \log n)$ and amortized message complexity $O(k(n) \cdot \log_{k(n)} n)$.*

2. *In the leaf-increasing tree model, if an upper bound n_f on the number vertices in the dynamically growing tree is known in advance, with label size $O(\frac{\log^3 n_f}{\log\log n_f})$ and amortized message complexity $O(\frac{\log n_f}{\log\log n_f})$.*

3. *In the leaf-dynamic tree model, with label size $O(\log_{k(n)} n \cdot \log n)$ and amortized message complexity $O\left(\sum_i k(n_i) \cdot \log_{k(n_i)} n \cdot \frac{\mathcal{MC}(\pi, n_i)}{n_i}\right) + O(\sum_i \log^2 n_i)$.*

[1]The above requirements are satisfied by most natural sublinear functions such as $\alpha x^\epsilon \log^\beta x$, $\alpha \log^\beta \log x$ etc..

6 Applications in a non-distributed environments

Distance queries in trees: Harel and Tarjan [36] describe a linear time algorithm to preprocess a tree and build a data structure allowing NCA queries to be answered in constant time on a RAM. Subsequently, simpler algorithms with better constant factors have been proposed in [53, 13, 28, 58, 14]. On a pointer machine, [36] show a lower bound of $\Omega(\log\log n)$ on the query time, which matches the upper bound of [60]. In the leaf-increasing tree model, [22, 10] show how to make updates in amortized constant time while keeping the constant worst-case query time on a RAM, or the $O(\log\log n)$ worst-case query time on a pointer machine. In [17] they show how to maintain the above mentioned results on a RAM in the leaf-dynamic tree model with worst case constant update. See [5] for a survey.

Simply by adding a pointer from each vertex to its depth and using Equation 1, we obtain the following lemma.

Lemma 6.1 *The results of [36, 53, 13, 28, 58, 14, 36, 60, 22, 10, 17] can be translated to support either distance queries or separation level queries (instead of NCA queries).*

We note that other types of dynamic models were studied in the non-distributed environment regarding NCA queries (e.g. [17, 10]). However, in these types of topological changes, our transformation to distance queries is not efficient since any such changes may effect the depth of too many vertices.

Flow queries in general graphs: Let $G \in \mathcal{G}(n, W)$ and let $u_1, u_2, \cdots, u_n$ be the set of vertices of G. Recall that in in [39], they show how to construct a weighted tree $\tilde{T}_G \in \mathcal{T}(O(n), W \cdot n)$ with n leaves $v_1, v_2, \cdots, v_n$ such that $flow_G(u_i, u_j) = sep - level_{\tilde{T}_G}(v_i, v_j)$. Using Lemma 6.1 applied on the results of [36], we can preprocess $\tilde{T}_G$ with $O(n \cdot \max\{1, \frac{\log W}{\log n}\})$ space such that separation level queries can be answered in $O(\max\{1, \frac{\log W}{\log n}\})$ time. The exact model needed to prove the following lemma formally is deferred to the full paper.

Lemma 6.2 *Any graph $G \in \mathcal{G}(n, W)$ can be preprocessed using $O(n \cdot \max\{1, \frac{\log W}{\log n}\})$ space such that flow queries can be answered in $O(\max\{1, \frac{\log W}{\log n}\})$ time.*

7 Conclusion and open problems

In this paper we demonstrate that considering the labels of three vertices, instead of two, can lead to a significant reduction in the sizes of the labels. Inspecting two labels, and inspecting three, are approaches that lie on one end of a spectrum. On the other end of the spectrum would be a representation for which the decoder inspects the labels of all the nodes before answering (n-query labeling schemes). It is not hard to show that for any graph family $\mathcal{F}$ on n node graphs, and for many functions (for example, distance or adjacency), one can construct an n-query labeling scheme on $\mathcal{F}$ using asymptotically optimal labels, i.e, $\log |\mathcal{F}|/n + \Theta(\log n)$-bit labels (though the decoder may not be polynomial). The idea behind such a scheme is to enumerate the graphs in $\mathcal{F}$ arbitrarily. Then, given some $G \in \mathcal{F}$ whose number in this enumeration is i, distribute the binary representation of i among the vertices of G. In this way, given the labels of all nodes, the decoder can reconstruct the graph and answer the desired query. Therefore, a natural question is to examine other points in this spectrum, i.e, examine c-query labeling schemes for $1 < c < n$.

There are other dimensions to the above question. For example, by our definition, given the labels of two vertices, the c vertices that are chosen by the query algorithm Q are chosen simultaneously. Alternatively, one may define a possibly stronger model in which these c vertices are chosen one by one, i.e., the next vertex is determined using the knowledge obtained from the labels of previous vertices.

References

[1] I. Abraham, C. Gavoille, and D. Malkhi. Routing with improved communication-space trade-off. *Proc. 18th Int. Symp. on Distributed Computing (DISC)*, Oct. 2004.

[2] D. Angluin, J. Aspnes, J. Chen, Y Wu, Y. Yin. Fast construction of overlay networks. *Proc. SPAA 2005*, pp. 145-154.

[3] Y. Afek, B. Awerbuch, S.A. Plotkin and M. Saks. Local management of a global resource in a communication. *J. of the ACM* **43**, (1996), 1–19.

[4] S. Alstrup, P. Bille and T. Rauhe. Labeling schemes for small distances in trees. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2003.

[5] S. Alstrup, C. Gavoille, H. Kaplan and T. Rauhe. Nearest Common Ancestors: A Survey and a new Distributed Algorithm. *Theory of Computing Systems* **37**, (2004), 441–456.

[6] S. Alstrup, J. Holm and M. Thorup. Maintaining Center and Median in Dynamic Trees. In *Proc. 7th Scandinavian Workshop on Algorithm Theory*, July. 2000.

[7] S. Abiteboul, H. Kaplan and T. Milo. Compact labeling schemes for ancestor queries. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2001.

[8] S. Alstrup and T. Rauhe. Improved Labeling Scheme for Ancestor Queries. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.

[9] S. Alstrup and T. Rauhe. Small induced-universal graphs and compact implicit graph representations. In *Proc. 43'rd annual IEEE Symp. on Foundations of Computer Science*, Nov. 2002.

[10] S. Alstrup and M. Thorup. Optimal pointer algorithms for finding nearest common ancestors in dynamic trees. *J. of Algorithms*, **35(2)**, (2000), 169–188.

[11] M.A. Breuer. Coding the vertexes of a graph. *IEEE Trans. on Information Theory*, IT-12:148–153, 1966.

[12] M.A. Breuer and J. Folkman. An unexpected result on coding the vertices of a graph. *J. of Mathematical Analysis and Applications* **20**, (1967), 583–600.

[13] M.A. Bender and M. Farach-Colton. The LCA problem revised. In *4th LATIN*, 88–94, 2000.

[14] O. Berkman and U. Vishkin. Recursive star-tree parallel data structure. *SIAM J. on Computing* **22(2)**, (1993), 221–242.

[15] L. J. Cowen. Compact Routing Schemes with Minimum Stretch. In *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 1999.

[16] I. Cidon, I. S. Gopal, and S. Kutten. New models and algorithms for future networks. *IEEE Transactions on Information Theory* 41(3): 769-780 (1995).

[17] R. Cole and R. Hariharan. Dynamic LCA Queries on Trees. *SIAM J. on Computing* **34(4)**, (2005), 894-923.

[18] E. Cohen, E. Halperin, H. Kaplan and U. Zwick. Reachability and Distance Queries via 2-hop Labels. In *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.

[19] E. Cohen, H. Kaplan and T. Milo. Labeling dynamic XML trees. In *Proc. 21st ACM Symp. on Principles of Database Systems*, June 2002.

[20] D. Eppstein, Z. Galil and G. F. Italiano. Dynamic Graph Algorithms. In *Algorithms and Theoretical Computing Handbook* , M.J. Atallah, Ed., CRC Press, 1999, Chapt. 8.

[21] T. Eilam C. Gavoille and D. Peleg. Compact Routing Schemes with Low Stretch Factor. In *Proc. 17th Annual ACM Symp. on Principles of Distributed Computing* , ACM Press, may 1996, 11–20.

[22] H. N. Gabow. Data Structure for Weighted Matching and Nearest Common Ancestor with Linking. In *Proc. 1st Annual ACM Symp. on Discrete Algorithms* , Jan. 1990, 434–443.

[23] C. Gavoille. A Survey on Interval Routing. In *Theoretical Computer Science* , **245**, (2000), 217–253.

[24] P. Fraigniaud and C. Gavoille. Routing in trees. In *Proc. 28th Int. Colloq. on Automata, Languages & Prog.*, LNCS 2076, pages 757–772, July 2001.

[25] P. Fraigniaud and C. Gavoille. A space lower bound for routing in trees. In *Proc. 19th Int. Symp. on Theoretical Aspects of Computer Science*, March 2002, 65-75.

[26] P. Fraigniaud and P. Gauron. D2B: A de Bruijn based content-addressable network. Theor. Comput. Sci. 355(1): 65-79 (2006).

[27] J. Feigenbaum and S. Kannan. Dynamic Graph Algorithms. In *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, 2000.

[28] H. N. Gabow, J. L. Bentley and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Annual ACM Symp. on Theory of Computing*, May 1984.

[29] C. Gavoille and C. Paul. Split decomposition and distance labeling: an optimal scheme for distance hereditary graphs. In *Proc. European Conf. on Combinatorics, Graph Theory and Applications*, Sept. 2001.

[30] C. Gavoille and D. Peleg. Compact and Localized Distributed Data Structures. *J. of Distributed Computing* **16**, (2003), 111–120.

[31] C. Gavoille, M. Katz, N.A. Katz, C. Paul and D. Peleg. Approximate Distance Labeling Schemes. In *9th European Symp. on Algorithms*, Aug. 2001, Aarhus, Denmark, SV-LNCS 2161, 476–488.

[32] C. Gavoille, D. Peleg, S. Pérennes and R. Raz. Distance labeling in graphs. *J. of Algorithms* **53(1)** (2004), 85–112.

[33] H. Harel. A linear time algorithm for lowest common ancestors problem. In *21st Annual IEEE Symp. on Foundation of Computer Science*, Nov. 1980.

[34] M. Harchol-Balter, F. T. Leighton, and D. Lewin. Resource Discovery in Distributed Networks. *Proc. PODC* 1999, pp. 229-237.

[35] J. Holm, K. Lichtenberg and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. of the ACM* **48(4)**, (2001), 723–760.

[36] H. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. on Computing*, **13(2)**, 1984, 338–355.

[37] H. A. Jamrozik, M. J. Feeley, G. M. Voelker, J. Evans, A. R. Karlin, H. M. Levy, and M. K. Vernon. Reducing network latency using subpages in a global memory environment. *Proc. the 7th ACM Conference on Architectural Support for Programming Languages and Operating Systems*, 1996.

[38] A. Korman. General Compact Labeling Schemes for Dynamic Trees. In *Proc. 19th International Symp. on Distributed Computing*, Sep. 2005.

[39] M. Katz, N.A. Katz, A. Korman and D. Peleg. Labeling schemes for flow and connectivity. *SIAM Journal on Computing* **34** (2004),23–40.

[40] M. Katz, N.A. Katz, and D. Peleg. Distance labeling schemes for well-separated graph classes. In *Proc. 17th Symp. on Theoretical Aspects of Computer Science*, pages 516–528, Feb. 2000.

[41] A. Korman and D. Peleg. Labeling Schemes for Weighted Dynamic Trees. In *Proc. 30th Int. Colloq. on Automata, Languages & Prog.*, Eindhoven, The Netherlands, July 2003, SV LNCS.

[42] A. Korman, D. Peleg and Y. Rodeh. Labeling schemes for dynamic tree networks. *Theory of Computing Systems* **37**, (2004), 49–75.

[43] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. In *SIAM J. on Discrete Math* **5**, (1992), 596–603.

[44] H. Kaplan and T. Milo. Short and simple labels for small distances and other functions. In *Workshop on Algorithms and Data Structures*, Aug. 2001.

[45] H. Kaplan and T. Milo. Parent and ancestor queries using a compact index. In *Proc. 20th ACM Symp. on Principles of Database Systems*, May 2001.

[46] H. Kaplan, T. Milo and R. Shabo. A Comparison of Labeling Schemes for Ancestor Queries. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2002.

[47] E. P. Markatos and G. Dramitinos. Remote Memory to Avoid Disk Thrashing: A Simulation Study. *Proc. MASCOTS* 1996: 69-73.

[48] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. *Proc. 21st annual ACM symposium on Principles of distributed computing*, 2002.

[49] Moni Naor and Udi Wieder. Novel architectures for P2P applications: the continuous-discrete approach. *Proc. SPAA* 2003, pp. 50-59.

[50] D. Peleg. Proximity-preserving labeling schemes and their applications. In *Proc. 25th Int. Workshop on Graph-Theoretic Concepts in Computer Science*, pages 30–41, June 1999.

[51] D. Peleg. Informative labeling schemes for graphs. In *Proc. 25th Symp. on Mathematical Foundations of Computer Science*, volume LNCS-1893, pages 579–588. SV, Aug. 2000.

[52] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* SIAM, 2000.

[53] P. Powell. A further improved LCA algorithm. Technical Report TR90-01, University of Minneapolis, 1990.

[54] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *Proc. ACM SIGCOMM* 2001, pp. 161-172, August 2001.

[55] N. Santoro and R. Khatib. Labeling and implicit routing in networks. *The Computer Journal* **28**, (1985), 5–8.

[56] I. Stocia, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *Proc. ACM SIGCOMM* 2001, San Diego, CA, Aug. 2001.

[57] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences* **26(1)**, (1983), 362–391.

[58] B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. on Computing*, **17**, 1988, 1253–1262.

[59] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. of the ACM* **51**, (2004), 993–1024.

[60] A. K. Tsakalides and J. van Leeuwen. An optimal pointer machine algorithm for finding nearest common ancestors. Technical Report RUU-CS-88-17, Department of CS, University of Utrecht, 1988.

[61] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. 13th ACM Symp. on Parallel Algorithms and Architecture*, pages 1–10, Hersonissos, Crete, Greece, July 2001.

[62] J. Van Leeuwen and R. B. Tan. Interval routing. *The Computer Journal* **30**, (1987), 298–307.

[63] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Tech. rep., University of California, Berkeley, 2001.

A Appendix

A.1 Definition of the functions

We consider the following functions which are applied on pairs of vertices u and v in a graph $G = \langle V, E\rangle$. (When the underlying graph is clear from the context we omit the corresponding subscript).

- Flow: Denote by G' the multigraph obtained by replacing each edge e in G with $\omega(e)$ parallel edges of capacity 1. A set of paths P in G' is *edge-disjoint* if each edge $e \in E$ appears in no more than one path $p \in P$. Let $\mathcal{P}_{u,v}$ be the collection of all sets P of edge-disjoint paths in G' between u and v. Then the maximum flow between u and v is defined as $f(u,v) = \max_{P\in\mathcal{P}_{u,v}}\{|P|\}$, where $|P|$ is the number of paths in P. See [39] for additional details regarding labeling for the flow function.
- distance: in the case where G is weighted (respectively, unweighted), $d_G(u,v)$ equals the weighted (resp. unweighted) distance between u and v in G.
- routing: function: $rout_G(u,v)$ is the port number at u leading to the next vertex on the shortest path connecting u to v.

If the graph is a tree T then we consider also the following functions:

- separation level:$Sep - level_T(u,v)$ equals the depth of $NCA(u,v)$.

We distinguish between the following two variants for the NCA function.

- id-NCA: assume that identities containing $O(\log n)$ bits are assigned to the vertices by an adversary. Then, $id - NCA_T(u,v)$ is the identity of $NCA(u,v)$.
- label-NCA: assuming each vertex can freely select its own identity (as long as all identities remain unique), $label - NCA_T(u,v)$ is the identity of $NCA(u,v)$. (In this case, the identities may also be referred to as labels.)

A.2 Query labeling schemes for the separation level and distance functions

We start with describing a the 1-query labeling scheme $\varphi_{sep-level} = \langle \mathcal{M}_{sep-level}, Q_{sep-level}, \mathcal{D}_{sep-level} \rangle$ which supports the separation level function on $\mathcal{T}(n,W)$. The idea behind $\varphi_{sep-level}$ is to label the first sublabels with the labels given by the label-NCA labeling scheme $\pi_{NCA}(v)$ and to label the second sublabel of each vertex v with its depth $depth(v)$. Then, using the labels of two vertices u and v, the query algorithm $Q_{sep-level}$ outputs the first sublabel of $w = NCA(u,v)$ using the decoder of $\pi_{NCA}(v)$. Given the labels of u, v and w, the decoder $\mathcal{D}_{sep-level}$ simply outputs the second sublabel of w, which is its depth. Since the label size of $\pi_{NCA}(v)$ is $O(\log n)$ and since the depth of each vertex v can be encoded using $O(\log(nW))$ bits, we obtain that the label size of $\varphi_{sep-level}$ is $O(\log n + \log W)$. Using Claim 3.2, we obtain the following claim.

Claim A.1 *The label size of a 1-query separation-level labeling scheme on $\mathcal{T}(n,W)$ is $\Theta(\log n + \log W)$.*

We now describe how to transform our 1-query labeling scheme $\varphi_{sep-level}$ into a 1-query distance labeling scheme $\varphi_{dist} = \langle \mathcal{M}_{dist}, Q_{dist}, \mathcal{D}_{dist} \rangle$. The marker and query algorithms of φ_{dist} are the same as the corresponding marker and query algorithms of $\varphi_{sep-level}$. Given the labels $\mathcal{M}_{dist}(v)$, $\mathcal{M}_{dist}(u)$ of two vertices v and u in some $T \in \mathcal{T}(n,W)$ as well as the label $\mathcal{M}_{dist}(w)$ of $w = NCA(v,u)$, the decoder algorithm $\mathcal{D}_{dist}$ outputs $\mathcal{M}^{data}_{dist}(v) + \mathcal{M}^{data}_{dist}(u) - 2\mathcal{M}^{data}_{dist}(w)$. The correctness of φ_{dist} follows from the following equation.

$$d(v,u) = depth(v) + depth(u) - 2 \cdot depth(NCA(v,u)). \tag{2}$$

Since the marker algorithm of φ_{dist} is the same as the marker algorithm of $\varphi_{sep-level}$, we obtain that the label size of φ_{dist} is $O(\log n + \log W)$. The following lemma follows, using Claim 3.2.

Claim A.2 *The label size of a 1-query distance labeling scheme on $\mathcal{T}(n,W)$ is $\Theta(\log n + \log W)$.*

Lemma 3.3 follows by combining Claim A.1 and Claim A.2.

A.3 Sketch of the proof of Lemma 3.5

The fact that any 1-query flow labeling scheme on $\mathcal{G}(n,W)$ must have label size $\Omega(\log n + \log W)$ follows using similar arguments as in the proof of Claim 3.2. We now show how to construct a 1-query flow labeling scheme on $\mathcal{G}(n,W)$ with label size $O(\log n + \log W)$. As shown in [39], given a graph $G \in \mathcal{G}(n,W)$ with vertices $u_1, u_2, \cdots, u_n$, one can construct a weighted tree $\tilde{T}_G \in \mathcal{T}(O(n), W \cdot n)$ with n leaves $v_1, v_2, \cdots, v_n$ such that $flow_G(u_i,u_j) = sep-level_{\tilde{T}_G}(v_i,v_j)$. Therefore, using our 1-query separation level labeling scheme $\varphi_{sep-level}$ on $\tilde{T}_G$, we obtain a 1-query flow labeling scheme φ_{flow} with size $O(\log(O(n)) + \log(nW)) = O(\log n + \log W)$. ▌

A.4 A 3-approximation routing scheme in the fixed port model

First note that in order to make sense, any routing scheme must have label size $\Omega(\log n)$ since the labels must be distinct. Moreover, the header size of any routing scheme must also be $\Omega(\log n)$ since the label of the destination vertex needs to be encoded in the header. We now construct our 3-approximation routing labeling scheme $\pi_{approx} = \langle \mathcal{M}_{approx}, \mathcal{R}_{approx} \rangle$, which operates in the fixed-port model and has label size and header size $\Theta(\log n)$.

Recall that the designer port numbers at any vertex v are numbered from 1 to $deg(v)$ (except that the designer port numbers at the root r are numbered from 2 to $deg(r)$). Moreover, the designer port number leading from any non-root vertex v to its parent is numbered 1. Let $v \in T$ be a non-leaf vertex. For every $2 \leq i \leq deg(v)$, let $child_i^{des}(v)$ be the child of v such that the designer port number at v leading to $child_i^{des}(v)$ is i, i.e., $port_{des}(v, child_i^{des}(v)) = i$. Let $c(v)$ denote the number of children of v, i.e, $c(v) = deg(v)$ if v is the root and $c(v) = deg(v) - 1$ otherwise. For every $1 \leq i \leq c(v)$, let $port_i^{fix}(v)$ be the i'th smallest fixed-port number at v among the fixed-port numbers leading from v to its children. For every $1 \leq i \leq c(v)$, let $child_i^{fix}(v)$ be the child of v such that $port_i^{fix}(v)$ is the fixed-port number at v leading to $child_i^{fix}(v)$, i.e., $port_{fix}(v, child_i^{fix}(v)) = port_i^{fix}(v)$. Let $n(v)$ denote the number i such that $v = child_i^{fix}(p(v))$. The marker algorithm $\mathcal{M}$ assigns v the label

$$\mathcal{M}_{fix}(v) = \langle \mathcal{M}_{des}(v)\ ,\ port_{fix}(p(v)\ ,\ child_{n(v)}^{des}(p(v)))\ ,\ port_{fix}(v, p(v)) \rangle.$$

The label of the root r is simply $\mathcal{M}_{fix}(r) = \langle \mathcal{M}_{des}(r), 0, 0 \rangle$. Note that the label of a vertex v is composed of three fields, namely, $\mathcal{M}_{fix}(v) = \langle \mathcal{M}_1(v), \mathcal{M}_2(v), \mathcal{M}_3(v) \rangle$. For a non-root vertex v, its second field $\mathcal{M}_2(v)$ is the fixed-port number of the port at v'parent whose designer port number is $n(v)$. (Recall that $n(v)$ was defined to be the *fixed* port leading to v.)

The routing is performed as follows. Given the label $\mathcal{M}(w)$ of the destination vertex w and the label $\mathcal{M}(v)$ of the vertex v initiating the route of the message, the router algorithm $\mathcal{R}$ at v does the following. Let $i = \mathcal{D}_{des}(\mathcal{M}_1(v), \mathcal{M}_1(w))$. If $i = 1$ then a header containing $\langle \mathcal{M}(w), 0 \rangle$ is attached to the message and the message (with the header) is forwarded to v's parent via port number $\mathcal{M}_3(v)$. If $i \neq 1$ then a header containing $\langle \mathcal{M}(w), -1 \rangle$ is attached to the message, and the message (with the header) is forwarded to v's child $child_i^{fix}(v)$ via the i'th smallest adversary port number at v among the fixed-port numbers leading to children of v. Note that since $\mathcal{M}_3(v)$ is the fixed-port number leading from v to its parent, v can identify which of its ports lead to its children. Furthermore, it is a local computation at v to find which of its (adversary) port numbers is the i'th smallest port number among the port numbers leading to its children.

If a vertex v receives a message with header $\langle \mathcal{M}(w), 0 \rangle$ and $\mathcal{M}(v) = \mathcal{M}(w)$ then $v = w$ and the message reached its destination w thus completing the route. If, $\mathcal{M}(v) \neq \mathcal{M}(w)$ then the router $\mathcal{R}$ at v does the following (similar operations to the case v is the vertex initiating the route). Let $i = \mathcal{D}_{des}(\mathcal{M}_1(v), \mathcal{M}_1(w))$. If $i = 1$ then a header is attached to the message containing $\langle \mathcal{M}(w), 0 \rangle$ and the message (with the header) is forwarded to v's parent via port number $\mathcal{M}_3(v)$. If $i \neq 1$ then a header is attached to the message containing $\langle \mathcal{M}(w), -1 \rangle$ and the message (with the header) is forwarded to v's child $child_i^{fix}(v)$ via port number $port_i^{fix}(v)$.

If a vertex v receives a message with header $\langle \mathcal{M}(w), -1 \rangle$ then the router $\mathcal{R}$ at v does the following. A header containing $\langle \mathcal{M}(w), \mathcal{M}_2(v) \rangle$ is attached to the message and the message (with the header) is forwarded to v's parent via port number $\mathcal{M}_3(v)$.

If a vertex v receives a message with header $\langle \mathcal{M}(w), x \rangle$, where $x \neq 0, -1$, then the router $\mathcal{R}$ at v does the following. A header containing $\langle \mathcal{M}(w), 0 \rangle$ is attached to the message and the message (with the header) is delivered to a child of v via port number x. Let us now show correctness.

Lemma A.3 *The routing scheme $\pi_{approx} = \langle \mathcal{M}_{approx}, \mathcal{R}_{approx} \rangle$ is a 3-approximation routing scheme.*

Proof: Let w be the destination vertex. Any intermediate node u on the route delivers the message to its parent iff its parent is the next vertex on the shortest path connecting u and w. Furthermore, if a vertex u delivers the message to its parent then the header of the message is set to $\langle \mathcal{M}(w), 0 \rangle$.

Any intermediate node u receiving the message with header $\langle \mathcal{M}(w), x\rangle$, where $x \neq 0, -1$, deliverers the message to one of its children via port number x, where x is precisely the fixed-port number of the port at v whose designer port number is $\mathcal{D}_{des}(\mathcal{M}_{des}(v), \mathcal{M}_{des}(w))$. Therefore, the message is delivered to the next vertex on the shortest path connecting u and w. Moreover, the message is delivered with the header $\langle \mathcal{M}(w), 0\rangle$.

If u is the vertex initiating the route or if u is some intermediate node receiving the message with header $\langle \mathcal{M}(w), 0\rangle$, where the next vertex on the shortest path from u to w is one of u's children, then precisely two messages are sent before the message returns to u with header $\langle \mathcal{M}(w), x\rangle$, where $x \neq 0, -1$.

We therefore get that the resulting route from the initiator v to the destination w is at most thrice as long as the shortest path connecting v and w. ▮

Note that the label size of π_{des} is $O(\log n)$ and that each port number is encoded using $O(\log n)$ bits. Therefore, using the previous lemma, we obtain Lemma 4.1

A.5 Sketch of the Proof of Theorem 5.1

The proof presented here in not completely self contained. A truly complete proof would include many details concerning three papers, namely, [24] and especially [42] and [38], and is, thus, too long to be given here.

We divide the proof into two parts. In the first part we consider the claims of Theorem 5.1 regarding the dynamic 1-query labeling schemes supporting the distance, separation level and id-NCA functions and in the second part we consider the claims of Theorem 5.1 regarding the dynamic 1-query routing labeling scheme.

A.5.1 First part of the proof

Let us first consider the claims of Theorem 5.1 regarding the dynamic 1-query labeling schemes supporting the distance, separation level and id-NCA functions. This part follows rather naturally from [42, 38].

Recall that in a query labeling scheme, the label of each vertex is composed of two sublabels, the index sublabel and the data sublabel. In our (static) 1-query labeling schemes supporting the distance, separation level and id-NCA functions, the index sublabel of each vertex v is simply the label assigned to v by π_{NCA}, the label-NCA labeling scheme of [5], and the data sublabel of v is either v's depth or v's identity. Therefore, we can dynamically maintain the index sublabel of each vertex using $\hat{\pi}_{NCA}$, the dynamic label-NCA labeling scheme resulted by applying either Theorem 4.16 of [42] or Theorems 1 and 2 of [38] on π_{NCA}. In addition, the data sublabel of each vertex v can easily be maintained as follows. In case we consider the id-NCA function, whenever a new vertex v joins the tree and an identity $id(v)$ is given to v by the adversary, this identity is stored at the data sublabel of v. If we consider, instead, either the distance or the separation level functions, the data sublabel of the root is set to be 0, and whenever a new vertex v joins the tree, it communicates with its parent $p(v)$ and sets its data sublabel to be $\hat{\mathcal{M}}_{data}(v) = 1 + \hat{\mathcal{M}}_{data}(p(v))$, where $\hat{\mathcal{M}}_{data}(p(v))$ is the data sublabel given to $p(v)$ by our dynamic 1-query scheme. Therefore, each vertex v maintains its depth (or identity) in the its data sublabel with an extra constant additive cost to the amortized message complexity of $\hat{\pi}_{NCA}$.

The query and decoder algorithms of the resulted dynamic 1-query labeling scheme relate to $\hat{\pi}_{NCA}$ similarly to the way the query and decoder algorithms of the corresponding static 1-query scheme relate to π_{NCA}. Since the labels of π_{NCA} can be assigned by a distributed algorithm using $O(n)$ messages, Theorem 4.16 of [42] and Theorems 1 and 2 of [38] imply the claims of Theorem 5.1 regarding the dynamic 1-query labeling schemes supporting the distance, separation level and id-NCA functions.

A.5.2 Second part of the proof

We now consider the claims of Theorem 5.1 regarding the dynamic 1-query routing labeling scheme. Let us first recall the scheme π_{des} of [24] for routing over trees, in which each vertex has both a routing table and a label. When routing a message from vertex x to a vertex y, vertex x is given the label $L(y)$ of the destination vertex y and uses its routing table to decide which of its incident ports leads to the next vertex w on the shortest path connecting x and y. Then x prepares a header containing the label $L(y)$ and sends the message together with the header to w. When w receives the message it repeats this process (without changing the header) until the message reaches y. Since both the routing table and the label of each vertex use $O(\log n)$ bits, one can encode the routing table into the label and easily modify π_{des} into a routing labeling scheme with label size $O(\log n)$. Using the terminology of [24], the label of each vertex v is $\langle id(v), \omega(v), \omega_1(v), cpath(v)\rangle$. In particular, the label of the root r is $\langle 1, n, \omega_1(r), \emptyset\rangle$, where $\omega_1(r)$ is the largest number of descendants of any child of r.

We now describe how to modify π_{des} to be a routing labeling scheme on n-vertices trees with label size $O(\log n)$, such that the label of the root is always $\langle 1\rangle$. We denote the modified scheme by π'_{des}. We note that our dynamic query labeling scheme (described label) will invoke π'_{des} on multiple subtrees of the given dynamic tree. In each such application, the label given to the root of the subtree (not just the root of the whole tree) is always $\langle 1\rangle$ (this label is then concatenated with an additional label, to obtain unique labeling). For every vertex v, let $L(v)$ denote the label given to v by π_{des} and for every non-root vertex v, let $\rho(v)$ be the port number leading from the root r to the next vertex on the shortest path connecting r and v. The label given to any vertex v by π'_{des} is the following:

$$L'(v) \;\leftarrow\; \begin{cases} 1 & \text{if } v = r, \\ L(v) \circ \rho(v) & \text{otherwise, where } \circ \text{ stands for concatenation.} \end{cases}$$

Let us mention that in π_{des}, the identity $id(v)$ of the a vertex v is its DFS number. Hence, $L'(r) = 1 = id(r)$. As shown in [24], given the labels $L(x)$ and $L(y)$ of two vertices x and y in the tree, the decoder $\mathcal{D}$ of π_{des} operates as follows.

$$\mathcal{D}(L(x), L(y)) \;\leftarrow\; \begin{cases} 0 & \text{if } id(y) = id(x), \\ 1 & \text{if } id(y) < id(x) \text{ or } id(y) \geq id(x) + w(x), \\ 1 + b & \text{if } id(x) < id(y) \leq id(x) + w_1(x), \\ p + b & \text{otherwise, where } p = |cpath(x) + 1|\text{-th element of } cpath(y). \end{cases}$$

Without getting into the details concerning the meanings of the parameters in the above formula, we describe the decoder $\mathcal{D}'$ of π'_{des} which satisfies $\mathcal{D}'(L'(x), L'(y)) = \mathcal{D}(L(x), L(y))$ for every two vertices x and y. Given the labels $L'(x)$ and $L'(y)$ of two vertices x and y, the decoder $\mathcal{D}'$ operates as follows.

$$\mathcal{D}'(L'(x), L'(y)) \;\leftarrow\; \begin{cases} 0 & \text{if } id(y) = id(x), \\ \mathcal{D}(L(x), L(y)) & \text{if } L'(x) \neq \langle 1\rangle \text{ and } L'(y) \neq \langle 1\rangle, \\ 1 & \text{if } id(y) = \langle 1\rangle \text{ and } id(x) > 1, \\ \rho(y) & \text{if } id(x) = \langle 1\rangle \text{ and } id(y) > 1. \end{cases}$$

The following claim shows that π'_{des} is a correct routing labeling scheme on trees.

Claim 1: For every two vertices x and y, $\mathcal{D}'(L'(x), L'(y)) = \mathcal{D}(L(x), L(y))$.

Proof of Claim 1: Clearly, if $id(x) = id(y)$ then $\mathcal{D}'(L'(x), L'(y)) = \mathcal{D}(L(x), L(y)) = 0$. Moreover, if both $x \neq r$ and $y \neq r$, then by the definition of the decoder D', $\mathcal{D}'(L'(x), L'(y)) = \mathcal{D}(L(x), L(y))$. Assume now that $id(x) = 1$ ($x = r$) and $id(y) > 1$ ($y \neq r$). In this case, by the definition of $\rho(y)$, $D'(L'(x), L'(y)) = \rho(y)$ is the port number leading from $x = r$ to the next vertex on the shortest path connecting x and y. By the correctness of π_{des}, $D(L(x), L(y))$ also equals $\rho(y)$. If $id(y) = 1$ and $id(x) > 1$, then y is the root and therefore the parent of x is next vertex on the shortest path connecting x and y. It follows that $D(L(x), L(y)) = 1$, and by the definition of the decoder $\mathcal{D}'$, $D'(L'(x), L'(y))$ also equals 1. This established Claim 1. ▮

Since the label size of π_{des} is $O(\log n)$ and since for every vertex v, the port number $\rho(v)$ can be encoded using $O(\log n)$ bits, we obtain the following claim.
Claim 2: π'_{des} is a routing labeling scheme on $\mathcal{T}(n)$ with label size $O(\log n)$. Moreover, the label given to the root of any tree by the scheme π'_{des} is $\langle 1 \rangle$.

We now describe how to extend φ_{fix}, our 1-query routing labeling scheme, to the dynamic scenario. We denote the resulted scheme by $\hat{\varphi}_{fix} = \langle \hat{\mathcal{M}}_{fix}, \hat{Q}_{fix}, \hat{\mathcal{D}}_{fix} \rangle$. Let T be a dynamic tree in the fixed-port model. Let $\hat{\pi}_{des}(v)$ be the dynamic routing labeling scheme resulted by applying either Theorem 4.16 of [42] or Theorems 1 and 2 of [38] on the modified routing labeling scheme π'_{des} described above. Note that π'_{des} is designed to operate in the designer port model, and therefore $\hat{\pi}_{des} = \langle \hat{\mathcal{M}}_{des}, \hat{\mathcal{D}}_{des} \rangle$ should also operate in the designer port model. However, our aim is a scheme for the fixed port model. Informally, we overcome this difficulty by running $\hat{\pi}_{des}$ assuming the designer port model and whenever the marker algorithm $\hat{\mathcal{M}}_{des}$ wishes to assign a port number to a port of v (as it should in the designer port model), it instead uses this value to label the corresponding child of v. Formally, the dynamic marker algorithm $\hat{\mathcal{M}}_{fix}$ operates as follows. As in any query labeling scheme, the label $\hat{\mathcal{M}}_{fix}(v)$ assigned to a vertex v by the dynamic marker algorithm $\hat{\mathcal{M}}_{fix}$ contains two sublabels, namely, the index sublabel $\hat{\mathcal{M}}^{index}(v)$ and the data sublabel $\hat{\mathcal{M}}^{data}(v)$. For simplicity, we assume that the dynamic tree contains only the root when the algorithm starts. The case that the algorithm is started when the tree already contains additional vertices is deferred to the full paper. (It uses some additional modification of the translation methods.)

The dynamic marker algorithm $\hat{\mathcal{M}}_{fix}$

The dynamic marker algorithm $\hat{\mathcal{M}}_{fix}$ first invokes $\hat{\mathcal{M}}_{des}$ as if we where in the designer port model. We do not describe here the dynamic algorithm $\hat{\mathcal{M}}_{des}$ (given either by the method of [42] or by the method of [38]). However, we note that from time to time $\hat{\mathcal{M}}_{des}$ assigns and updates labels and port numbers as a result of a recomputation (shuffle- in the terminology of [42] and reset- in the terminology of [38]) performed on some subtree. The events of assigning and updating the labels and port numbers are modified in the dynamic marker algorithm $\hat{\mathcal{M}}_{fix}$ as described in the following steps (which are applied simultaneously). In particular, the port numbers that should be assigned by the marker algorithm $\hat{\mathcal{M}}_{des}$, are not assigned to the ports (since we are dealing with the fixed-port model). Instead, these numbers are used as described in Steps 4 and 5 below.

1. Whenever a label $\hat{\mathcal{M}}_{des}(v)$ is assigned to (or updated at) a vertex v by $\hat{\mathcal{M}}_{des}$, this label is stored at the first field of v's data sublabel, i.e., $\hat{\mathcal{M}}^{data}_1(v) = \hat{\mathcal{M}}_{des}(v)$.

2. Each time a non-leaf vertex v is supposed to be assigned a new label $\hat{\mathcal{M}}_{des}(v)$ under the dynamic marker algorithm $\hat{\mathcal{M}}_{des}$, it sends a message to each of its children containing this new value $\hat{\mathcal{M}}_{des}(v)$. In turn, each child u of v sets the first field in its index sublabel to be this value, i.e., $\hat{\mathcal{M}}^{index}_1(u) = \hat{\mathcal{M}}_{des}(v) = \hat{\mathcal{M}}_{des}(p(u))$.

3. Whenever a new leaf u joins the tree, it sends a signal to its parent v which in turn sends u a message containing $\hat{\mathcal{M}}^{data}_1(v)$. When u receives this message it sets the first field in its index sublabel to be the value contained in the message, i.e., $\hat{\mathcal{M}}^{index}_1(u) = \hat{\mathcal{M}}^{data}_1(v) = \hat{\mathcal{M}}_{des}(p(u))$.

4. Whenever the marker algorithm $\hat{\mathcal{M}}_{des}$ wishes to assign a port number ρ to a port leading from a vertex v to one of its children u, it refrains from doing so and instead assigns the second field of

$\hat{\mathcal{M}}^{index}(u)$ the value ρ, i.e., $\hat{\mathcal{M}}_2^{index}(u) = port_{des}(p(u), u)$.

5. Whenever a leaf u joins the tree as a child of some vertex v and the corresponding ports are assigned a port number by the adversary, the following happens. The port number $port_{fix}(u, v)$ is stored at the third field of the data sublabel of u, i.e, $\hat{\mathcal{M}}_3^{data}(u) = port_{fix}(u, p(u))$. Moreover, a message is sent from v to u containing the new adversary port number at v, i.e., $port_{fix}(v, u)$. When u receives this message it sets the second field of its data sublabel to be the value $port_{fix}(v, u)$, i.e, $\hat{\mathcal{M}}_2^{data}(u) = port_{fix}(p(u), u)$.

It follows that at all times the index sublabel of each vertex v is

$$\hat{\mathcal{M}}^{index}(v) = \langle \hat{\mathcal{M}}_{des}(p(v)) \ , \ port_{des}(p(v), v) \rangle$$

and the data sublabel of v is

$$\hat{\mathcal{M}}^{data}(v) = \langle \hat{\mathcal{M}}_{des}(v) \ , \ port_{fix}(p(v), v) \ , \ port_{fix}(v, p(v)) \rangle .$$

The query and decoder algorithms of $\hat{\varphi}_{fix}$, the resulted dynamic 1-query labeling scheme, relate to $\hat{\pi}_{des}$ similarly to the way the query and decoder algorithms of φ_{fix} relate to π_{des}. It follows that $\hat{\varphi}_{fix}$ is a correct dynamic 1-query routing labeling scheme with asymptotically the same label size as $\hat{\pi}_{des}$. Since the labels of π'_{des} (containing $O(\log n)$ bits) can be assigned by a distributed algorithm using $O(n)$ messages, if $\hat{\mathcal{M}}_{des}$ is given by Theorem 4.16 of [42], then the label size and amortized message complexity of $\hat{\mathcal{M}}_{des}$ are as indicated (for $\hat{\mathcal{M}}_{fix}$) in the second item of Theorem 5.1. Moreover, if $\hat{\mathcal{M}}_{des}$ is given by Theorem 1 (respectively, Theorem 2) of [38], then the label size and amortized message complexity of $\hat{\mathcal{M}}_{des}$ are as indicated (for $\hat{\mathcal{M}}_{fix}$) in the first (resp., third) item of Theorem 5.1. It therefore remains to show that the number of messages used by the marker protocol $\hat{\mathcal{M}}_{fix}$ is asymptotically the same as the number of messages used by the marker protocol of $\hat{\mathcal{M}}_{des}$. Clearly, we only need to show that the number of messages resulted from modifying $\hat{\mathcal{M}}_{des}$ into $\hat{\mathcal{M}}_{fix}$ (as described in Steps 1-5 in the description of Algorithm $\hat{\mathcal{M}}_{fix}$) does not affect the asymptotic message complexity of of $\hat{\mathcal{M}}_{des}$. Step 1 in the description of Algorithm $\hat{\mathcal{M}}_{fix}$ does not incurs any messages. Steps 3 and 5 in the description of Algorithm $\hat{\mathcal{M}}_{fix}$ incur $O(1)$ amortized message complexity per topological change. Note that in the dynamic version of the fixed-port model, a port is assigned a number by the adversary only once: when a leaf u joins the tree as a child of some vertex v, the port number at u and the port number at v leading to u are assigned a port number by the adversary. Therefore, Step 4 in the description of protocol $\hat{\mathcal{M}}_{fix}$ also incurs $O(1)$ amortized message complexity per topological change. It remains to bound the number of messages incurred by Step 2. As mentioned before, in the translation scheme of [42], a non-leaf vertex v may change its label only when a shuffle operation is invoked on a subtree T' containing $T(v)$, the subtree containing v and all of its descendants (see Subsections 4.1.3 and 4.1.4 in [42]). In this shuffle operation, a distributed algorithm assigning the labels of the corresponding static scheme (in this case π_{des}) is invoked on T'. Therefore, the number of messages incurred by this shuffle operation is bounded from below by the number of vertices in T'. It follows that if $\hat{\mathcal{M}}_{des}$ is given by Theorem 4.16 of [42], then the extra number of messages incurred by Step 2 in the description of protocol $\hat{\mathcal{M}}_{fix}$ is bounded from above by the message complexity of $\hat{\pi}_{des}$. This completes the proof of Item 2 in the theorem.

We now bound the number of messages incurred by Step 2 in the case $\hat{\mathcal{M}}_{des}$ is given by the translation method of [38]. This is needed in order to prove Items 1 and 3 in the theorem. In this case (similarly to the translation method of [42]) a non-leaf vertex v may change its label only when a reset operation is invoked on a subtree T' containing v. The number of messages incurred by this reset operation is bounded from below by the number of vertices in T'. Let r' be the root of T'. In contrast to the subtree on which the shuffle operation of [42] is invoked on, the subtree T' may not necessarily contain all of r''s children. In fact T' is a subtree of the dynamic tree T which is composed of a root vertex r', a set of children $\{u_i\}_{i\in I}$ (for some I defined in [42]) of r' and all the descendant the vertices in $\{u_i\}_{i\in I}$. We now show that when applying the translation scheme of [38] on the modified routing labeling scheme π'_{des}, a non-leaf vertex v with more than one child may change its label only when a reset operation is invoked on a subtree T' containing v's parent.

Note that in $\hat{\pi}_{des}$, when a vertex v is added to the tree, v participates in a reset operation applied on a subtree containing $p(v)$. Fix some vertex v. Let $t_o(v)$ be a time in which v participated in a reset operation applied on a subtree containing $p(v)$ and let $L_0(v)$ be the label given to v in this reset operation. The first time after $t_0(v)$ that v changed its label, was either due to another reset operation which was invoked on a subtree containing $p(v)$ or when a (first) child of v was added to the tree and a reset operation was applied on a subtree rooted at v. In the latter case, let $L_1(v)$ be the label given to v in this reset operation, and let $t_1(v)$ be the time when v received this label. Since $\langle 1\rangle$ is the label given by π'_{des} to the root of any tree, it follows from the description of Scheme $FSDL_p^k$ in [38], that $L_1(v) = L_0(v) \circ \langle 1\rangle$. At any time from $t_1(v)$ until the next time v changed its label as a result of a reset operation containing $p(v)$, whenever v participated in a reset operation invoked on some subtree T', the subtree T' must be rooted at v. Moreover, the new label given to v by the reset operation on T' is $L_0(v)$ concatenated with the label given to v by the static routing labeling scheme π'_{des}, which is $\langle 1\rangle$, since v is the root of T'. Therefore, the label of v remains $L_1(v) = L_0(v) \circ \langle 1\rangle$. It follows that a non-leaf vertex v with more than one child may change its label only when a reset operation is invoked on a subtree T' containing v's parent. Therefore, T' contains all of v's children. It follows that the extra number of messages incurred by Step 2 in the description of protocol $\hat{\mathcal{M}}_{fix}$ is bounded from above by the message complexity of $\hat{\pi}_{des}$, as desired. Altogether, we obtain that the number of message used by the marker protocol $\hat{\mathcal{M}}_{fix}$ is asymptotically the same as the number of message used by the marker protocol of $\hat{\mathcal{M}}_{des}$, as desired. The theorem follows. Theorems 1 and 2 of [38] imply the remaining claim of Theorem 5.1 regarding the dynamic 1-query labeling scheme supporting the routing function. ▮