Compact Separator Decompositions in Dynamic Trees and Applications to Labeling Schemes

(Extended abstract)

Amos Korman [*]   David Peleg [†]

Abstract

In this paper we construct an efficient scheme maintaining a *separator decomposition representation* in dynamic trees. Our dynamic scheme uses asymptotically optimal labels. In order to maintain the short label, the scheme uses $O(\log^3 n)$ amortized message complexity, per topology change, where n is the current number of nodes in the tree. We note that a separator decomposition is a fundamental decomposition of trees used extensively as a component in many static graph algorithms. Therefore, our dynamic separator decomposition may be used for constructing dynamic versions to these algorithms.

Going along these lines, we then show how to use our dynamic separator decomposition to construct rather efficient routing schemes on dynamic trees, for both the designer and the adversary port models. Since passing messages from one place to another is usually the main purpose of a network, constructing efficient dynamic routing schemes is an important task in the fields of distributed computing and communication networks. Our dynamic routing schemes use $O(\log^3 n)$ amortized message complexity and the labels they maintains are optimal up to a multiplicative factor of $O(\log \log n)$.

In addition, we show how to use our dynamic separator decomposition to construct dynamic labeling schemes supporting the ancestry and NCA relations using asymptotically optimal labels and $O(\log^3 n)$ amortized message complexity. Finally, we show how to use our dynamic separator decomposition to extend a known result on dynamic distance labeling schemes.

Keywords: Distributed algorithms, dynamic networks, routing schemes, graph decompositions, informative labeling schemes.

1 Introduction

Background: A distributed representation scheme is a scheme maintaining global information using local data structures (or *labels*). Such schemes play an extensive role and sometimes crucial in the fields of distributed computing and communication networks. Their goal is to locally store useful information about the network and make it readily and conveniently accessible. As a notable example, the basic function of a communication network, namely, message delivery, is performed by its *routing scheme*, which in turn requires maintaining certain topological knowledge. Often, the performance of the network as a whole may be dominated by the quality of the routing scheme and the accuracy of the topological information. Extensive research was done on representation schemes in the *static* (fixed topology) setting (e.g., [12, 8, 2, 14]). The common measure for evaluating a static representation scheme is the *label size*, i.e., the maximum number of bits used in a label. In this paper, a representation scheme with asymptotically optimal label size is termed *compact.*

The more realistic and involved distributed *dynamic* setting, where processors may join or leave the network or new connections may be established or removed, has received much less attention. In the distributed dynamic setting, an *update protocol* is activated where the topology change occurs, and its goal is to update vertices, by transmitting messages over the links of the underlying network. Ideally, the update protocol maintains short labels using only a limited number of messages.

In this paper we consider representation schemes in dynamic trees, operating under the *leaf-dynamic* tree model, in which at each step, a leaf may either join or leave the tree. We consider the controlled dynamic model, which was also considered in [1, 15], in which the topological changes do not occur spontaneously. Instead, when an entity wishes to cause a topology change at some node u, it enters a *request* at u, and performs the change only after the request is granted a permit from the update protocol. The controlled model may be found useful in Peer to Peer applications and in other popular overlay networks. See [15] for more details and motivations regarding the controlled model.

In this paper, we present several dynamic representation schemes, which are efficient in both their label size and their communication complexity. Specifically, all our dynamic schemes incur $O(\log^3 n)$ amortized message complexity, per topological change. We first present a compact representation scheme of a separator decomposition in dynamic trees, and then use this basic scheme to derive compact labeling schemes supporting the ancestry and NCA relations on dynamic trees. In addition, we present dynamic routing schemes which have optimal label size up to $O(\log \log n)$ multiplicative factor, for both the adversary and the designer port models. Finally, we show how to use our dynamic separator decomposition to extend a known result on dynamic distance labeling schemes.

Related work: An elegant and simple compact labeling scheme was presented in [12], for supporting the ancestry relation on static n-node trees using labels of size $2 \log n$. Applications to XML search engines motivated various attempts to improve the constant multiplicative factor in the label size. A labeling scheme supporting the ancestry relation using label size $1.5 \log n$ was presented in [4] and in [5] the label size was further reduced to $\log n + O(\sqrt{\log n})$.

Static compact labeling schemes were presented for two types of NCA relations on trees. For the id-based NCA relation (which is the type of NCA relation we consider in this paper), a static labeling scheme was developed in [18] using labels of $\Theta(\log^2 n)$ bits on n-node trees. A static labeling scheme supporting the label-based NCA relation using labels of $\Theta(\log n)$ bits on n-node trees was presented in [2]. In addition, [2] gave a survey on applications and previous results concerning NCA queries on trees, in both the distributed and centralized settings.

Labeling schemes for routing on static trees were investigated in a number of papers until finally optimized in [8, 9, 23]. For the *designer port* model, in which the designer of the scheme can freely enumerate the port numbers of the nodes, [8] shows how to construct a routing scheme using labels of size $O(\log n)$ 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 size $O(\log^2 n / \log \log n)$

on n-node trees. In [9] they show that both label sizes are asymptotically optimal. Independently, a routing scheme for trees of label size $(1 + o(1)) \log n$ was given in [23] for the designer port model.

Dynamic data structures for trees have been studied extensively in the centralized model, cf. [21, 6, 20]. For comprehensive surveys on centralized dynamic graph algorithms see [19, 10].

A survey on popular link state dynamic routing protocols (e.g. OSPF) can be found in [22]. Compared to our dynamic routing schemes, these routing schemes are more robust on weaker dynamic models, such as ones which allow spontaneous faults, however, their message complexity is higher.

In [1] they consider the controlled model and establish an efficient dynamic controller, which can operate in the *leaf-increasing* tree model, in which the only topology change allowed is that a leaf joins the tree. This controller can, in particular, be used to maintain a constant approximation of the number of nodes in the (leaf-increasing) tree, using $O(\log^2 n)$ amortized message complexity. In [15] an extended controller was derived for the controlled model, which can operate under both insertions and deletions of both leaves and internal nodes. In particular, using $O(\log^2 n)$ amortized message complexity, their controller can be used to maintain a constant approximation of the number of nodes in the dynamic tree, undergoing both deletions and additions of nodes. In addition, they show how to use their controller to translate dynamic labeling schemes from the leaf-increasing tree model to the leaf-dynamic tree model.

A dynamic routing scheme in the leaf-increasing tree model was given in [3] using identities of size $O(\log^2 n)$, database size $O(\Delta \log^3 n)$ (where Δ is the maximum degree in the tree) and amortized message complexity $O(\log n)$ per topological change.

Dynamic distance labeling schemes on trees were presented in [17, 16] for the *serialized model*, in which it is assumed that the topology changes are spaced enough so that the update protocol can complete its operation before the next topology change occurs. Two dynamic β-approximate distance labeling schemes (in which given two labels, one can infer a β-approximation to the distance between the corresponding nodes) were presented in [16]. The first scheme applies to a model in which the tree topology is fixed but the edge weights may change, and the second applies to a model in which the only topological event that may occur is that an edge increases its weight by one. The amortized message complexity of the first scheme depends on the local density parameter of the underlying graph and the amortized message complexity of the second scheme is polylogarithmic. Both schemes have label size $O(\log^2 n + \log n \log W)$ where W denotes the largest edge weight in the tree.

Two general translation methods for extending static labeling schemes on trees to the dynamic setting were considered in [17] and [13], for the serialized model. Both approaches fit a number of natural functions on trees, such as ancestry relation, routing, NCA relation etc. The translation methods incur overheads (over the static scheme) in both the label size and the message complexity. Specifically, the method of [13] yields dynamic compact labeling schemes, although the amortized message complexity is high, namely, $O(n^\epsilon)$. On the other hand, the label sizes of the dynamic labeling schemes in [13], which use polylogarithmic amortized message complexity, have a multiplicative overhead factor of $O(\log n / \log \log n)$ over the optimal size.

Our contributions: In this paper we consider a dynamic tree T operating under the leaf-dynamic tree model and the controlled model. We present several efficient dynamic schemes for T, all of which use $O(\log^3 n)$ amortized message complexity, per topological change, where n is the current number of nodes in T. We first present an efficient protocol maintaining a compact (i.e., with optimal label size) separator decomposition representation in T. Let us note that the general translation method of [13] may also yield such a dynamic compact scheme, however, their resulted scheme uses high amortized message complexity, namely, $O(n^\epsilon)$.

Our basic dynamic separator scheme is then used in order to construct several other dynamic labeling schemes for the dynamic tree T, which improve known results. Specifically, we present dynamic compact labeling schemes supporting the ancestry and the NCA relations, and we establish routing schemes for both the designer and the adversary port models, which use optimal label size up to a multiplicative $O(\log \log n)$ factor. For any of the above mentioned functions f, the best known label size for

dynamic labeling schemes supporting f, that use polylogarithmic amortized message complexity, has a multiplicative overhead of $O(\log n / \log \log n)$ over the optimal label size. In addition, the best known amortized message complexity for dynamic compact labeling schemes supporting f is $O(n^\epsilon)$.

Finally, we show that our dynamic separator decomposition can also be used to allow the dynamic distance labeling schemes of [16] to operate under a more general dynamic model. In addition to allowing the edges of the underlying tree to change their weight, the extended dynamic model allows also leaves to be added to or removed from the tree. The extended scheme incurs an extra $O(\log^3 n)$ additive factor to the amortized message complexity.

Paper outline: For clarity of presentation, we first consider the serialized model and in the full paper we show how to modify the schemes for operating under the controlled model. We first assume the leaf-increasing tree model. The adaptation to the leaf-dynamic model is done according to the method described in [15] (the idea is to ignore deletions, maintain an estimate to the number of topological changes and initialize the tree every $\Theta(n)$ topological changes).

Due to lack of space, this extended abstract contains mainly intuition regarding the construction of the dynamic separator decomposition and its applications to dynamic ancestry and routing labeling schemes. The formal description and analysis of the separator decomposition construction in the leaf-increasing tree model, as well as the intuition behind the other applications and the adaptation to the leaf-dynamic model appear in the Appendix. The formal description and analysis of the applications will appear in the full paper.

Our separator scheme for the leaf-increasing tree model is based on an adaptation of our static scheme (described in Section 3). The adaptation requires a number of components whose tasks are maintaining estimates on the sizes of the various subtrees managed in the decomposition, manipulating and reorganizing these subtrees, and maintaining the corresponding labels and topological data. These components are Protocol SHUFFLE, Protocol MAINTAIN_W and Protocol DYN_SEP. Generally speaking, Protocol MAINTAIN_W is used to allow each separator v to maintain a constant approximation to the number of nodes in the the subtree $T^*(v)$ for which v was chosen as a separator. Whenever the size of $T^*(v)$ grows by some constant factor, the main protocol DYN_SEP invoke Protocol SHUFFLE on $T^*(v)$ which calculates a new separator decomposition on $T^*(v)$ which is consistent with the global separator decomposition. The correct operation of each of these protocols relies on the assumption that certain properties hold at the beginning of their execution, and in turn, each of these components guarantees that certain properties hold upon their termination. Hence the correctness proof of the entire algorithm depends on establishing an intricate set of invariants and showing that these invariants hold throughout the execution.

2 Preliminaries

Our communication network model is restricted to tree topologies. Let T be a tree rooted at vertex r and let $T(v)$ denote the subtree of T rooted at v. For every vertex $v \in T$, let $D(v)$ denote the *depth* of v, i.e., the unweighted distance between v and the root r. For a non-root vertex v, denote by $p(v)$ its parent in the tree. The *ancestry* relation is defined as the transitive closure of the parenthood relation. Define the *weight* of v, denoted $\omega(v)$, as the number of vertices in $T(v)$, i.e., $\omega(v) = |T(v)|$. Let n denote the number of vertices in the tree, i.e., $n = \omega(r)$.

For every two numbers $a < b$, let $[a, b)$ denote the set of integers $a \le i < b$. For every integer $q \ge -1$ let $I_q = [3 \cdot 2^{q+2}, 3 \cdot 2^{q+3})$ and for every $m \le n$ and $-1 \le q \le \log m$, let $J_q(m) = [\frac{m}{2^{q+1}}, \frac{m}{2^q})$ and let $\widehat{J_q}(m) = [\frac{m}{2^{q+1}}, \frac{m}{2^{q-1}})$. In other words, $\widehat{J_q}(m) = J_q(m) \cup J_{q-1}(m)$.

Separator decomposition: We first define a *separator decomposition* of a tree T recursively as follows. At the first stage we choose some vertex v in T to be the *level-1* separator of T. By removing

v, T breaks into disconnected subtrees which are referred to as the subtrees *formed* by v. Each such subtree is decomposed recursively by choosing some vertex to be a level-2 separator, etc.

Let $T^{subtrees}$ be the collection of all subtrees obtained by the resulting partitioning, on all levels of the recursion. Note that the trees on each level are disjoint but the entire collection contains overlapping trees. Moreover, in this partitioning, each vertex v in T belongs to a unique subtree $T_l(v) \in T^{subtrees}$ on each level l of the recursion, up to the level $l(v)$ in which v itself is selected as the separator. The subtrees $T = T_1(v), T_2(v), \cdots, T_{l(v)}(v)$ are referred to as the *ancestor subtrees* of v. Define the *separator tree* T^{sep} to be the tree rooted at the level-1 separator of T, with the level-2 separators as its children, and generally, with each level $j + 1$ separator as the child of the level j separator above it in the decomposition. For a vertex v in T, let $s_j(v)$ denote the *level-j separator of v*, i.e., the ancestor of v in T^{sep} at depth j. We associate each vertex v with the subtree $T^*(v) = T_{l(v)}(v)$ for which v is chosen as its separator. If v is a level j separator, then $T^*(v)$ is referred to as a *level j subtree*. (See Figure 1.)
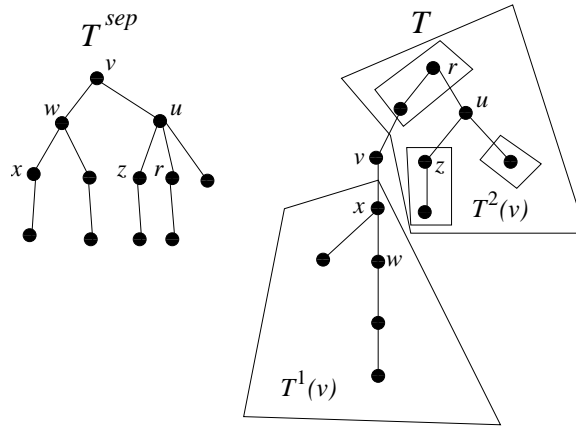

Figure 1: In the depicted tree T, rooted at r, the node v is the level-1 separator of T. By deleting v, T breaks into $T^1(v)$ and $T^2(v)$. Similarly, w is the separator of $T^1(v)$ and u is the separator of $T^2(v)$, therefore w and u are the children of v in T^{sep}. By deleting u, $T^2(v)$ breaks into three subtrees, one of which contains z as its separator. We have $T^*(v) = T$, $T^*(w) = T^1(v)$ and $T^*(u) = T^2(v)$.

For $1/2 \leq \delta < 1$, a δ-*separator* of T is a vertex v whose removal breaks T into disconnected subtrees of at most $\delta|T|$ vertices each. It is a well known fact that every tree has a δ-separator (even for $\delta = 1/2$), and that one can recursively partition the tree by δ-separators. Such a decomposition is termed δ-*separator decomposition*. It is clear that the depth of the corresponding separator tree T^{sep} is $O(\log |T|)$. In the special case where $\delta = 1/2$, the separator node is called a *perfect separator*, and the decomposition is called a *perfect separator decomposition*.

Representations for separator decompositions: One may define a distributed representation for separator decompositions in trees in various ways. For our purposes, we define a *separator decomposition representation* as follows. Each vertex v in a tree T is given a label $L(v)$ so that the following hold.
1. Each vertex has a unique label, i.e., $L(u) \neq L(v)$ for every two vertices $u, v \in T$.
2. Given the label $L(v)$ of some vertex v and an integer $1 \leq i \leq l(v)$, one can extract the label $L(u)$ where u is the level-i separator of v.

Note that by the first requirement, the maximum number of bits in a label in any n-node tree is $\Omega(\log n)$ for any separator decomposition representation.

The functions: We consider the following functions F on pairs of vertices u, v of a rooted tree.
(a) Routing: $F(u, v)$ is the port number at u leading to the next vertex on the path from u to v.
(b) Ancestry relation: if u is an ancestor of v in the tree then $F(u, v) = 1$, otherwise, $F(u, v) = 0$.
(c) NCA relation: assuming each vertex z has a unique identifier $id(z)$ (encoded using $O(\log n)$ bits), $F(u, v)$ is the identifier $id(z)$ of the nearest common ancestor (NCA) z of u and v, i.e., the common ancestor of both u and v of maximum depth.

Labeling schemes: An *F-labeling scheme* $\pi = \langle \mathcal{M}_\pi, \mathcal{D}_\pi \rangle$ is composed of the following components:

1. A *marker* algorithm $\mathcal{M}_\pi$ that given a tree, assigns a label $L(v)$ to each vertex v in the tree.
2. A polynomial time *decoder* algorithm $\mathcal{D}_\pi$ that given the labels $L(u)$ and $L(v)$ of two vertices u and v in the tree, outputs $F(u, v)$.

We note that in our schemes, the labels given to the vertices may contain several fields. In order to distinguish between the different fields of some label one can use an additional label $L'(v)$ for v, which has the same number of bits as $L(v)$ and whose 1's mark the locations where the fields of $L(v)$ begin. Clearly, adding $L'(v)$ does not increase the asymptotic label size.

The dynamic models: The following types of topology changes are considered.

Add-leaf: A new vertex u is *added* as a child of an existing vertex v.

Remove-leaf: A leaf u of a tree is *deleted*.

Subsequent to a topology change, both relevant nodes u and v are informed of it. When a new edge is attached to a node v, the corresponding port at v is assigned (either by an adversary or by v) a unique port-number (i.e., at any time, the port numbers at v are distinct), encoded using $O(\log n)$ bits.

Two main variants are considered in the literature. In the *designer port* model each node v is allowed at any time, to freely select and change the port numbers on its incident ports (as long as they remain disjoint) and in the *adversary port* model, the port numbers at each node are fixed by an adversary. In this paper, all our results, execpt for one of our schemes, assume the weaker adversary port model. The one exemption is one of our dynamic routing schemes which assumes the designer port model.

Various dynamic models are considered in the literature. In the *leaf-increasing* tree model, cf. [3, 17, 13], the only topology change allowed is that a leaf joins the tree, and in the more general *leaf-dynamic* tree model, cf. [1, 17, 13], leaves can either be added to or removed from the tree. All the results in this paper apply for the leaf-dynamic tree model.

For simplicity, we assume that the initial tree contains just one vertex, namely, its root. (In order to implement our scheme on an existing initial tree, one may use a preprocessing algorithm $\mathcal{P}$ which simulates our dynamic scheme on a scenario where starting with just the root, vertices are added to the trees so as to construct the initial tree.)

After every topological change, an update protocol $\mathcal{U}$ is activated in order to maintain the labels $L(v)$ of the vertices v to fit the requirements of the corresponding problem. As mentioned before, in the context of distributed networks, the messages are sent over the edges of the underlying graph.

For simplicity of presentation, in this abstract, we analyze our protocols assuming the *serialized model* (e.g., [17, 16, 13]) in which the topological changes occur in a serialized manner and are sufficiently spaced so that the update protocol has enough time to complete its operation in response to a given topological change, before the occurrence of the next change. This model allows us to concentrate on the combinatorial aspects of the problem without considering asynchrony issues. Let us remark, however, that our schemes can operate under the weaker *controlled model* (considered also in [15, 1], see [15] for more details and motivations). In this model, when a topological change τ wishes to occur at vertex v, a *request* R_τ to perform τ arrives at v. Vertex v performs the topological change τ only when the request R_τ is granted a *permit* from the update protocol. It is guaranteed that every request to perform a topological change is eventually granted a permit.

Moreover, in the leaf-increasing model, our dynamic schemes can operate under the weak *uncontrolled* model in which the topological changes may occur in rapid succession or even concurrently. Correctness, however, is required only at quiet times, i.e., times for which all updates concerning the previous topological changes have occurred. (It can easily be shown that no dynamic compact separator decomposition scheme, can be expected to operate correctly also in non-quiet times). Due to lack of space, the analysis of our schemes under the stronger models is deferred to the full paper.

For a static scheme π on n-node trees, model, we are interested in the following complexity measures. The *label size*, $\mathcal{LS}(\pi, n)$, is the maximum number of bits in $L(v)$ taken over any vertex v.

The *message complexity*, $\mathcal{MC}(\pi, n)$, is the maximum number of messages (of size $O(\log n)$) sent by a distribute algorithm assigning the labels of π.

For a dynamic scheme π, the above definitions are taken over all scenarios where at most n topology changes occur.

3   The static separator representation scheme π_{Stat_Sep}

Let us first note that a static compact separator representation scheme is implicitly described in [11]. However, we were not able to extend that scheme to the dynamic scenario. Instead, in this section we present a new static compact separator decomposition representation scheme π_{Stat_Sep} (which is in some sense a relaxation of the scheme in [11]), which we find easier to extend to the dynamic scenario. Scheme π_{Stat_Sep} enjoys label size $\Theta(\log n)$ and message complexity $O(n \log n)$.

Recall that in a δ-separator decomposition of the tree T, each node v is a separator of some level. Given a δ-separator decomposition, a simple way of constructing a representation for it is to assign each vertex a disjoint identity and then to label each vertex by the list of identities of v's ancestors in T^{sep}. However, this simple scheme has label size $O(\log^2 n)$. In order to reduce the label size to $O(\log n)$ we exploit the liberty of choosing the labels of the separators. As in the simple scheme described above, our marker algorithm assigns each vertex v a different label $L^{sep}(v)$ containing $l(v)$ fields. However, in contrast to the simple scheme mentioned above, for any $1 < l \leq l(v)$, the l'th field $L_l^{sep}(v)$ of $L^{sep}(v)$ does not contain the identity of the level-l separator of v. Instead, it contains a number proportionate to $|T_{l-1}(v)|/|T_l(v)|$. Moreover, the label of the level-k separator of v is the prefix of $L^{sep}(v)$ containing the first k fields in $L^{sep}(v)$. Informally, these properties are achieved in the following manner. Define the labels $L^{sep}(v)$ of the separators v by induction on their level. The label of the level-1 separator is set to be $\langle 0 \rangle$. Assume that we have defined the labels of all the level-$(l-1)$ separators. For each level-$(l-1)$ separator v, we now define the labels of its children $v_1, v_2, \cdots$ in T^{sep} as follows. For each k, v_k is first assigned a unique number $\rho(v_k)$ (in the sense that if $k \neq g$ then $\rho(v_k) \neq \rho(v_g)$) such that $\rho(v_k) \in [3 \cdot 2^{q+2}, 3 \cdot 2^{q+3})$ iff $|T^*(v)|/2^{q+1} < |T^*(v_k)| \leq |T^*(v)|/2^q$, or in other words, $\rho(v_k) \in I_q$ iff $|T^*(v_k)| \in J_q(|T^*(v)|)$.

Note that for each q, there could be at most 2^{q+1} children v_k of v in T^{sep} such that $|T^*(v)|/2^{q+1} < |T^*(v_k)| \leq |T^*(v)|/2^q$. Therefore, the interval $I_q = [3 \cdot 2^{q+2}, 3 \cdot 2^{q+3})$ contains sufficiently many integers so that every separator v_k satisfying $|T^*(v_k)| \in J_q(|T^*(v)|)$ can be issued a distinct integer in I_q.

For every k, after assigning each vertex v_k a number $\rho(v_k)$ as described above, the label of v_k is set to be the concatenation $L^{sep}(v_k) = L^{sep}(v) \circ \rho(v_k)$. The fact that for each k, $\rho(v_k)$ is unique is used to show that the labels are disjoint. Note that the label of a level-l separator u can be considered as a sequence of l fields $L^{sep}(u) = L_1^{sep}(u) \circ \cdots \circ L_l^{sep}(u)$. Moreover, for each $1 \leq j < l$, the label of the level-j separator of u is simply $L_1^{sep}(u) \circ \cdots \circ L_j^{sep}(u)$. In addition, for $1 \leq j \leq l$, the $j + 1$'st field $L_{j+1}^{sep}(u)$ is proportionate to $|T_j(u)|/|T_{j+1}(u)|$. This property is used to show that the label size is $O(\log n)$.

In order to implement π_{Stat_Sep} by a distributed protocol, when the separator v wishes to assign a unique value $\rho(v_k) \in I_q$ to one of its children (in T^{sep}), it somehow needs know which values it had already assigned in the range I_q. For this purpose, for every $-1 \leq q \leq \lceil \log n \rceil$, v maintains a counter $c_q(v)$ counting the number of values $\rho(v_k) \in I_q$ that were already assigned by it. Whenever v wishes to assign a new value $\rho(v_k) \in I_q$, it simply selects $3 \cdot 2^{q+2} + c_q(v)$ and then raises $c_q(v)$ by 1. The fact that $\rho(v_k)$ indeed belongs to I_q follows from the following invariant, which holds throughout the execution at every vertex v.

Counters invariant at v: For every $-1 \leq q \leq \lceil \log n \rceil$, the set of currently assigned values in I_q is a prefix of I_q, namely, $[3 \cdot 2^{q+2}, 3 \cdot 2^{q+2} + c_q(v) - 1]$.

The formal description and analysis of the distributed Protocol $\text{STAT\_SEP}(T)$, which is initiated at the root of a given tree T and assigns each vertex v the label $L^{sep}(v)$, are deferred to Appendix A.

Lemma 3.1 *Protocol $\text{STAT\_SEP}(T)$ constructs a compact separator decomposition on a static n-node tree T using $O(n \log n)$ messages.*

4 Protocol SHUFFLE

Protocol SHUFFLE is invoked in the dynamic scenario on subtrees $T' \in T^{subtrees}$ that are suspected to violate some balance properties required in order to maintain the compact separator decomposition on the whole tree T. The goal of Protocol SHUFFLE(T') is to recompute a separator decomposition representation on T' while keeping it consistent with the global separator decomposition representation on T. Specifically, we assume that each separator v keeps $\omega_0^*(v)$, the number of vertices in $T^*(v)$ after the last application of Protocol SHUFFLE on a subtree containing $T^*(v)$. Let v be a level-l separator and let $T^1(v), T^2(v), \cdots$ be the subtrees formed by v. Let T' be one of those subtrees, w.l.o.g. $T' = T^1(v)$.

The correct operation of Protocol SHUFFLE relies on the fact that throughout the dynamic scenario, the following invariants are maintained for every level-l separator v.

The balance invariants:
B1: For every vertex $u \in T^*(v)$, if $L_{l+1}^{sep}(u) \in I_q$ for some q then $|T_{l+1}(u)| \in \widehat{J_q}(\omega_0^*(v))$.
B2: $|T^*(v)| \in [\omega_0^*(v), \frac{5}{4} \cdot \omega_0^*(v)]$.

The growth property:
Just before Protocol SHUFFLE is invoked on $T^*(v)$, we have $|T^*(v)| \geq \gamma \cdot \omega_0^*(v)$ where $\gamma = \sqrt{5/4}$.

The fact that these properties are indeed maintained throughout the dynamic scenario is proved in Lemma C.1 in Appendix C.

Protocol SHUFFLE(T') is conceptually composed of three stages. In the first stage, all the labels in T' are initialized to be $L^{sep}(v)$ (which contains l fields). At the second stage, the $l + 1$'st field of the labels in T', $\rho(T')$, is initialized so that it is proportionate to $\omega_0^*(v)/|T'|$ and disjoint from $\rho(T^i)$ for every $i > 1$. At the third stage, Protocol STAT_SEP is invoked on T' to initialize the following (i.e., the $l + 2$'nd, $l + 3$'rd, etc) fields of the labels in T' according to a perfect separator decomposition of T'. It is relatively easy to implement the first and third stages of Protocol SHUFFLE(T').

Let us now describe informally how Protocol SHUFFLE implements the second stage. In order for the new assigned value $\rho_{new}(T')$ to be proportionate to $\omega_0^*(v)/|T'|$, it may need to be in some different interval I_q than before. We use the counters $c_q(v)$ (described in the previous section) to count the number of values in I_q that were already assigned. When v wishes to assign T' a new value $\rho_{new}(T') \in I_q$, it selects the value $3 \cdot 2^{q+2} + c_q(v)$ and then raises $c_q(v)$ by 1. However, in contrast to Protocol STAT_SEP, the counters invariant is not necessarily maintained. Instead, the fact that $3 \cdot 2^{q+2} + c_q(v) \in I_q$ results from the following more involved argument. After applying $\mathcal{S}$, the last Protocol SHUFFLE to be applied on a subtree containing v, $c_q(v)$ was relatively small. Let T'' be one of the subtrees $T^i(v)$ that received a new value in I_q after $\mathcal{S}$ was invoked and let $\mathcal{S}''$ be the SHUFFLE protocol applied on T'' after which T'' received this value. By combining the balance invariant B2 (for the separator of T'') with the growth property (for T''), we obtain that the number of vertices that have joined T'' from the time $\mathcal{S}$ was invoked until the time $\mathcal{S}''$ was invoked is proportionate to $\omega_0^*(v)/2^q$. On the other hand, by B2, the total number of nodes joining $T^*(v)$ from the time $\mathcal{S}$ was invoked is at most $\omega_0^*(v)/4$. Combining these two observations, we obtain that the number of subtrees that received a new value in I_q after $\mathcal{S}$ was invoked is small enough to guarantee that $3 \cdot 2^{q+2} + c_q(v) \in I_q$.

The formal description of Protocol SHUFFLE as well as its analysis are deferred to Appendix B, where we show the following Lemma.

Lemma 4.1 $\mathcal{MC}(\text{SHUFFLE}(T')) = O(|T'| \log |T'|)$.

5 Protocol MAINTAIN_W

The goal of Protocol MAINTAIN_W is to allow each separator v in the dynamically growing tree to maintain a constant approximation to the number of nodes in $T^*(v)$. In [1], they show how to allow the root to maintain a constant approximation to the number of nodes in a growing tree, using $O(n \log^2 n)$ messages. Let us denote their protocol by Protocol WEIGHT_WATCH. In this section, for each separator v, we consider $T^*(v)$ as rooted at v. Protocol MAINTAIN_W simply invokes Protocol WEIGHT_WATCH

on $T^*(v)$, for each separator v. Therefore, each vertex u participates in $l(u)$ applications of Protocol WEIGHT_WATCH, one for each subtree $T_i(u)$. This can be implemented easily assuming each node u knows, for each $1 \leq i < l(u)$, the port number leading to its parent in $T_i(u)$ and the port numbers leading to its children in $T_i(u)$. This assumption can be removed by slightly modifying protocol SHUFFLE.

Note that if a δ-separator decomposition is maintained then every vertex u participates in $l(u) = O(\log n)$ concurrent executions of Protocol WEIGHT_WATCH. We therefore obtain the following lemma.

Lemma 5.1 *Assuming the leaf-increasing tree model, Protocol* MAINTAIN_W *allows each vertex v in the dynamic tree to maintain a constant approximation to the number of nodes in $T^*(v)$. Moreover, if a δ-separator decomposition is maintained at all times then $\mathcal{MC}(\text{MAINTAIN\_W}, n) = O(n \log^3 n)$.*

6   Dynamic separator decomposition

We now briefly sketch Protocol DYN_SEP, whose goal is to maintain a compact separator decomposition representation in the leaf-increasing model. Protocol DYN_SEP uses Protocol MAINTAIN_W as a subroutine and from time to time invokes Protocol SHUFFLE on different subtrees. Therefore, the correctness of Protocol DYN_SEP depends on the correctness of Protocol MAINTAIN_W and on the fact that the SHUFFLE properties are maintained whenever a SHUFFLE protocol is invoked. However, these are only guaranteed assuming that the balance invariants and the growth property are maintained when the SHUFFLE protocols take place and assuming that a δ-separator decomposition is maintained at all times. Protocol DYN_SEP guarantees these assumptions by invoking Protocol SHUFFLE$(T^*(v))$ whenever the number of vertices in some $T^*(v)$ grows by some constant factor. This is implemented as follows. Every vertex v keeps the value $\omega_0^*(v)$ which is the number of vertices in $T^*(v)$ after the last SHUFFLE protocol on a subtree containing v. Whenever the counter $\tilde{\omega}^*(v)$ (which is used by v in order to estimate $|T^*(v)|$) satisfies $\tilde{\omega}^*(v) \geq \sqrt{5/4} \cdot \omega_0^*(v)$, vertex v invokes Protocol SHUFFLE$(T^*(v))$. Protocol DYN_SEP and the proof of the following theorem are deferred to Appendix C.

Theorem 6.1 *Assuming the leaf-increasing tree model, Protocol* DYN_SEP *maintains a compact separator decomposition using $O(n \log^3 n))$ message complexity.*

7   Applications: dynamic labeling schemes for trees

In this section we describe the ideas behind our dynamic labeling schemes, all of which use $O(\log^3 n)$ amortized message complexity. In this extended abstract we sketch the improved dynamic ancestry and routing schemes and in Appendix E we sketch the improved NCA labeling scheme and the extended distance labeling scheme. The formal description and analysis is deferred to the full paper. We begin with sketching the ideas behind our dynamic compact ancestry labeling schemes.

Improved ancestry labeling schemes on dynamic trees: We first introduce a new static compact labeling scheme, $\pi_{Stat\_Anc} = \langle \mathcal{M}_{SA}, \mathcal{D}_{SA} \rangle$, supporting the ancestry relation, and then show how to extend it to the dynamic setting. Scheme π_{Stat_Anc} uses the separator decomposition representation obtained by Scheme π_{Stat_Sep}. For every two vertices v and u, let $s(v, u)$ denote the NCA of v and u in T^{sep}. Scheme π_{Stat_Anc} is based on the fact that a vertex v is an ancestor of a vertex u iff v is an ancestor of $s(v, u)$ and u is a descendant of $s(v, u)$. The label $L(v)$ given by the marker algorithm $\mathcal{M}_{SA}$ to a vertex v is composed of two sublabels, namely, the *separation sublabel*, $L^{sep}(v)$, and the *relative sublabel*, $L^{rel}(v)$. The separation sublabel $L^{sep}(v)$ is the label given to v by the scheme π_{Stat_Sep}. The relative sublabel $L^{rel}(v)$ is composed of $l(v)$ fields. The j'th field of $L^{rel}(v)$ contains two bits indicating whether $s_j(v)$, the level-j separator of v, is an ancestor of v in T, descendant of v in T or neither.

Given two labels $L(v)$ and $L(u)$, of two vertices v and u, one can extract the level i of $s(v, u)$ using the corresponding separation sublabels and then find whether in T, v is an ancestor of $s(v, u)$ and u is a descendant of $s(v, u)$, using the i'th field of the corresponding relative sublabels.

In the dynamic scenario, the separation sublabels are maintained using the dynamic scheme π_{Dyn_Sep}. Throughout the dynamic scenario, whenever a vertex v is assigned a new level j separator, the j'th field in its relative sublabel is updated appropriately, according to whether v is an ancestor (or a descendant) of this separator. By Theorem 6.1 we therefore obtain the following theorem.

Theorem 7.1 *Assuming the leaf-increasing model, Scheme π_{Dyn_Anc} is a dynamic ancestry labeling scheme π_{Dyn_Anc} with label size $\Theta(\log n)$ and message complexity $O(n \log^3 n)$.*

Dynamic routing labeling schemes: We now sketch our dynamic routing schemes π_{rout} which have optimal label size up to a multiplicative factor of $O(\log \log n)$. I.e., the label size of π_{rout} is $O(\log n \cdot \log \log n)$ for the designer port model, and $O(\log^2 n)$ for the adversary port model.

Let v be some vertex, and let $l(v)$ be level for which v was chosen as a separator. For each $1 \le i \le l(v)$, let $s_i(v)$ be the i'th separator of v. The label of v given by π_{rout} is composed of three sublabels. The first is the *separator sublabel* $L^{sep}(v)$ which is the label given to v by π_{Dyn_Sep} (recall that $L^{sep}(v)$ contains $l(v)$ fields). The second and third sublabels are the *port-to-separator sublabel* $L^{to-sep}(v)$ and the *port-from-separator sublabel* $L^{from-sep}(v)$. Each of these sublabels also contains $l(v)$ fields. The i'th field in $L^{to-sep}(v)$, namely $L_i^{to-sep}(v)$, is the port number leading from v to the next vertex on the shortest path connecting v and $s_i(v)$. The i'th field in $L^{from-sep}(v)$, namely $L_i^{from-sep}(v)$, is the port number leading from $s_i(v)$ to the next vertex on the shortest path connecting $s_i(v)$ and v. By slightly modifying Protocol SHUFFLE, we can ensure that whenever Protocol Dyn_Sep updates the i'th field in $L^{sep}(v)$, the i'th fields in the sublabels $L^{to-sep}(v)$ and $L^{from-sep}(v)$ are also updated appropriately.

Given the labels $L(u)$ and $L(v)$ of two vertices u and v, the port number leading from u to the next vertex on the shortest path connecting u and v is determined as follows. If $L^{sep}(u)$ is a prefix of $L^{sep}(v)$ and $L^{sep}(u)$ contains i fields, then $u = s_i(v)$ and therefore the desired port number is $L_i^{from-sep}(v)$. If, on the other hand, $L^{sep}(u)$ is not a prefix of $L^{sep}(v)$ then let i be the last index such that $L_i^{sep}(u) = L_i^{sep}(v)$. In this case, the i'th separator of u, $s_i(u)$, must be on the path connecting u and v and must be different than u. Therefore, the desired port number is $L_i^{to-sep}(u)$.

Scheme π_{rout} is clearly a correct dynamic routing schemes. Let us now analyze its label size. First, for each vertex v, the separator sublabel $L^{sep}(v)$ contains $O(\log n)$ bits. Both the port-to-separator sublabel $L^{to-sep}(v)$ and the port-from-separator sublabel $L^{from-sep}(v)$ contain $O(\log n)$ fields, where each such field contains a port number. Recall that it is assumed that each port number is encoded using $O(\log n)$ bits. It follows that in the adversary port model, the label size of Scheme π_{rout} is $O(\log^2 n)$. Let us now consider the designer port model and describe the method by which each vertex u chooses its port numbers, so that the label size of Scheme π_{rout} is $O(\log n \cdot \log \log n)$. Let $E^{sep}(u)$ be the set of edges leading from u to the next vertex on the shortest path connecting u and one of its ancestors in T^{sep}. Since u has $l(u) = O(\log n)$ such ancestors, $E^{sep}(u)$ contains $O(\log n)$ edges. For each edge $e \in E^{sep}(u)$, vertex u chooses a unique port number in the range $\{1, 2, \cdots, l(u)\}$. Therefore, each such port number can be encoded using $O(\log \log n)$ bits. We therefore immediately get that for every vertex v, the port-to-separator sublabel $L^{to-sep}(v)$ can be encoded using $O(\log n \cdot \log \log n)$ bits. We now describe the method by which each vertex u chooses its remaining port numbers, i.e., the port numbers of the edges not in $E^{sep}(u)$. For each such edge e, let $T^i(u)$ be the corresponding subtree formed by u. The corresponding port number at u is set to be the number $l(u) + \rho(T^i(u))$, where $\rho(T^i(u))$ is the number given to $T^i(u)$ by Protocol Dyn_Sep. We therefore obtain that the port numbers incident to u are disjoint. For a fixed vertex v and $i \le l(v)$, the port number $L_i^{from-sep}(v)$ is leading from $s_i(v)$ to x, the next vertex on the shortest path connecting $s_i(v)$ and v. If the edge $(s_i(v), x)$ belongs to $E^{sep}(s_i(v))$ then $L_i^{from-sep}(v)$ is encoded using $O(\log \log n)$ bits. Otherwise, the number of bits in $L_i^{from-sep}(v)$ is $O(\log \log n)$ plus the number of bits used to encode the $i + 1$'st subfield in $L^{sep}(v)$. Therefore, the number of bits used to encode $L^{from-sep}(v)$ is at most $O(\log n \cdot \log \log n) + O(\log n) = O(\log n \cdot \log \log n)$. We therefore obtain the following theorem.

Theorem 7.2 *Assuming the leaf-increasing model, Scheme π_{rout} is a correct dynamic routing scheme with $O(n \log^3 n)$ message complexity. Moreover, its label size is optimal up to a multiplicative factor of*

$O(\log \log n)$. *I.e., the label size of π_{rout} is $O(\log^2 n)$ for the adversary port model, and $O(\log n \cdot \log \log n)$ for the designer port model.*

References

[1] 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.

[2] 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.

[3] Y. Afek, E. Gafni and M. Ricklin. Upper and Lower Bounds for Routing Schemes in Dynamic Networks. In *Proc. 30th Symp. on Foundations of Computer Science*, 1989, 370–375.

[4] S. Abiteboul, H. Kaplan and T. Milo. Compact Labeling Schemes for Ancestor Queries. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, Jan. 2001.

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

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

[7] 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.

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

[9] 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. Mar. 2002.

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

[11] 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.

[12] S. Kannan, M. Naor, and S. Rudich. Implicit Representation of Graphs. In *SIAM J. on Descrete Math* **5**, (1992), 596–603.

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

[14] A. Korman. Labeling Schemes for Vertex Connectivity. In *Proc. 34th Int. Colloq. on Automata, Languages and Prog. (ICALP)*, July 2007.

[15] A. Korman and S. Kutten. Controller and Estimator for Dynamic Networks. In *Proc. 26th Ann. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, August 2007.

[16] 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.

[17] A. Korman, D. Peleg and Y. Rodeh. Labeling Schemes for Dynamic Tree Networks. *Theory of Computing Systems* **37**, (2004), 49–75.

[18] 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.

[19] L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, 2007.

[20] B. Schieber and U. Vishkin. On finding Lowest Common Ancestors: Simplification and Parallelization. *SIAM Journal on Computing* **17(6)**, (1988), 1253–1262.

[21] D. D. Sleator and R. E. Tarjan. A Data Structure for Dynamic Trees. *Journal of Computer and System Sciences* **26(1)**, (1983), 362–391.

[22] A. S. Tanenbaum. *Computer Networks* Prentice Hall, 2003.

[23] 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.

A Description and analysis of Protocol STAT_SEP

We first define a δ-*heavychild* decomposition of T. Each non-leaf vertex v marks a *heavy edge*, i.e., an edge leading to one of its children, $h(v)$, such that every other child u satisfies $\omega(u) \leq \delta \cdot \omega(v)$. A *heavy path* is a path consisting solely of heavy edges.

We describe the distributed recursive Protocol STAT_SEP(T), which is initiated at the root of a given a tree T and assigns each vertex v the label $L^{sep}(v)$. On the l'th level of the recursion, Protocol STAT_SEP(T) operates on the subtrees $T^*(u)$ which correspond to the level-l separators u. Let v be a level-$(l-1)$ separator and let $T^1(v), T^2(v), \cdots$ be the subtrees formed by v. For every i, let v_i be the separator of $T^i(v)$. Note that the vertices $v_1, v_2, \cdots$ are precisely v's children in T^{sep}. On the l'th level of the recursion, v sets the l'th field of the labels in each $T^k(v)$ to be the number $\rho(v_k)$ assigned to v_k by v according to the method described in Section 3. Since the value $\rho(v_k)$ of the l'th field of the labels of all the vertices $T^k(v)$ is the same, we may also refer to this value as $\rho(T^k(v))$. Note, however, that in order for v to assign a desired value $\rho(T^k(v)) = \rho(v_k) \in I_q$ in a distributed manner, v somehow needs know which values have already been assigned in the range I_q to other subtrees $T^j(v)$. We therefore maintain at v, for every $-1 \leq q \leq \lceil \log n \rceil$, a counter $c_q(v)$ (initially set to 0), which is used during the procedure in order to count the number of values $\rho(T^j(v)) \in I_q$ that were already assigned to subtrees $T^j(v)$. Moreover, the following invariant is maintained throughout the execution at every vertex v.

Counters invariant at v: For every $-1 \leq q \leq \lceil \log n \rceil$, the set $\{\rho(T^j(v)) \mid \rho(T^j(v)) \in I_q\}$ of currently assigned values in I_q, is precisely $[3 \cdot 2^{q+2}, 3 \cdot 2^{q+2} + c_q(v) - 1]$.

When v wishes to assign a new value $\rho(T^k(v)) \in I_q$, it assigns it the value $3 \cdot 2^{q+2} + c_q(v)$ and then raises $c_q(v)$ by 1.

Let us now give a formal description of Protocol STAT_SEP, starting by describing its Sub-protocols FIND_SEP and SORT_WEIGHT.

Sub-protocol FIND_SEP(T) is the straightforward protocol for computing a separator. It is initialized at the root r of an n-node tree T and its output is a perfect separator of T. In addition, following the execution of this sub-protocol, each node v knows its weight $\omega(v)$.

Sub-protocol FIND_SEP(T)

1. The root r broadcasts a signal instructing all the tree vertices to calculate their exact weight through a convergcast process. During the convergcast process, each vertex v keeps a pointer $h(v)$ to its heaviest child, i.e., a child u of v such that no other child of v has more descendants than u. This yields a 1/2-heavychild decomposition of T.

2. The root informs the vertices about n, the correct number of vertices in the entire tree.

3. The root sends a signal along the heavy path containing it, until it reaches a vertex v (possibly r itself) such that the child $h(v)$ satisfies $\omega(h(v)) < n/2$.

4. The output is v.

The proof of the following claim is straightforward.

Claim A.1 *The output of* FIND_SEP(T) *is a perfect separator of T and the message complexity of* FIND_SEP(T) *is* $\mathcal{MC}(\text{FIND\_SEP}, n) = O(n)$.

We now describe Sub-protocol SORT_WEIGHT(m, v), which is initiated at a separator v of some m-node subtree $T^*(v)$ of T. When this sub-protocol terminates, every neighbor $x \in N^*(v)$ of v is assigned a unique number $\rho(x)$ which is proportionate to $m/|T_x|$.

We assume that each vertex $x \in N^*(v)$ initially holds the number

$$n_x = \begin{cases} \omega(x) & , x \text{ is a child of } v \text{ in } T, \\ m - \omega(x) + 1 & , x \text{ is } v\text{'s parent in } T. \end{cases}$$

Sub-protocol SORT_WEIGHT(m, v)

1. The initiating separator v sends a signal to each of its neighbors. Upon receiving this signal, each neighboring vertex $x \in N^*(v)$ returns a message to v containing n_x.

2. Upon receiving the value n_x from x, vertex v returns to x a number $\rho(x)$, which is defined by the following procedure. Let $q(x)$ be the integer satisfying $n_x \in J_{q(x)}(m)$. Upon receiving n_x from x, vertex v creates the number $\rho(x) = 3 \cdot 2^{q(x)+2} + c_{q(x)}(v)$ and then sets $c_{q(x)}(v) = c_{q(x)}(v) + 1$.

Claim A.2 *For every $1 \le q \le \lceil \log n \rceil$, at any time during the execution of the sub-protocol, the following hold.*

1. *$c_q(v) < 2^{q+1}$,*

2. *If $x \in N^*(v)$ has already been assigned a number $\rho(x)$, then $\rho(x) \in I_q$ iff $n_x \in J_q(m)$,*

3. *The counters invariant is satisfied.*

Proof: We prove the claim by induction on the number of times Step 2 is applied in Sub-protocol SORT_WEIGHT(m, v). Assume by the inductive hypothesis that the claim holds after the k'th application of Step 2 and consider the $k + 1$'st application of Step 2, in which some vertex $x \in N^*(v)$ satisfies $n_x \in J_{q(x)}(m)$.

By the inductive hypothesis and by the second and third parts of the claim, we obtain that $c_{q(x)}(v) - 1$ vertices $y \in N^*(v)$ have already been assigned a number $\rho(y) \in I_{q(x)}$, and that for each such vertex y, $n_y \in J_{q(x)}(m)$. Together with x, it follows that there exist at least $c_{q(x)}(v)$ neighbors $w \in N^*(v)$ such that $n_w \in J_{q(x)}(m)$. Since every two neighbors $u, w \in N^*(v)$ satisfy $T_u \bigcap T_w = \emptyset$, we obtain the first part of the claim. The second and third parts of the claim follow directly from the first part and from the description of Sub-protocol SORT_WEIGHT(m, v). The claim thus follows by induction. ∎

The proof of the following claim is straightforward.

Claim A.3 $\mathcal{MC}(\text{SORT\_WEIGHT}(n, v)) = O(deg(v))$.

We now describe Protocol STAT_SEP(T). Initially all the labels are identical; for every vertex $v \in T$, $L^{sep}(v) = 0$. Protocol STAT_SEP(T) recursively calls Protocol STAT_SEP(T'), where T' is a subtree of T rooted at some vertex s.

Protocol STAT_SEP(T) **(for T rooted at r)**

1. The root r invokes Sub-protocol FIND_SEP(T), which outputs a perfect separator v of T.

2. Now r broadcasts the value $\omega(r) = |T|$ (which was calculated during Protocol FIND_SEP) to all the tree vertices.

3. Upon receiving this value, every vertex $x \in N^*(v)$ creates the number n_x as follows.

   If x is a child of v in T, then set $n_x = \omega(x)$ (this value was calculated during Protocol FIND_SEP),

   Otherwise, set $n_x = \omega(r) - \omega(x) + 1$.

4. The chosen separator v invokes Sub-protocol SORT_WEIGHT$(\omega(r), v)$, after which every vertex $x \in N^*(v)$ is given unique a number $\rho(x)$.

5. For every vertex $x \in N^*(v)$ and every vertex $w \in T_x$, set $L^{sep}(w) = L^{sep}(w) \circ \rho(x)$.

6. Each vertex $x \in N^*(v)$ recursively invokes Protocol STAT_SEP(T_x) at x.

The following lemma follows directly from the description of Protocol STAT_SEP(T) and from the second part of Claim A.2.

Lemma A.4 *After Protocol* STAT_SEP(T) *is invoked, the following properties are satisfied for every* $1 \le k$ *and every two vertices* v *and* u *in* T.
Static separator properties :
SS1: *If* $u \ne v$ *then* $L^{sep}(u) \ne L^{sep}(v)$,
SS2: *If* u *is the level-k separator of* v *then* $L^{sep}(u) = L_1^{sep}(v) \circ L_2^{sep}(v) \circ \cdots \circ L_k^{sep}(v)$,
SS3: *If* $L_{k+1}^{sep}(u) \in I_q$ *then* $|T_{k+1}(v)| \in J_q(|T_k(v)|)$.

The correctness and complexity bounds of Protocol STAT_SEP(T) are described in the following lemma.

Lemma A.5 *The labels assigned by Protocol* STAT_SEP(T) *to the vertices of* T *form a separator decomposition representation of* T. *Moreover, Protocol* STAT_SEP(T) *has the following complexities.*

1. $\mathcal{LS}(\text{STAT\_SEP}, n) = O(\log n)$,

2. $\mathcal{MC}(\text{STAT\_SEP}, n) = O(n \log n)$,

Proof: The fact that Protocol STAT_SEP(T) imposes a separator decomposition representation is clear from the static separator properties SS1 and SS2 (which are guaranteed by Lemma A.4). We now show that the static property SS3 implies that for every vertex v, the number of bits in $L^{sep}(v)$ is $O(\log n)$. Fix a level-l vertex v. For every $1 \le k \le l$, let $m(k) = |T_k(v)|$ and let $q(k)$ be such that $L_k^{sep}(v) \in I_{q(k)}$. By the static separator property SS3, for every $1 < k \le l$, $m(k) \le m(k-1)/2^{q(k)}$. Therefore $\prod_{\{k|1 \le k \le l\}} 2^{q(k)} \le m(1) = n$, yielding $\sum_{\{k|1 \le k \le l\}} q(k) \le \log n$. Since $L_k^{sep}(v) \in I_{q(k)}$, we get that $L_k^{sep}(v)$ can be encoded using $q(k) + 5$ bits and therefore $L^{sep}(v)$ can be encoded using $\sum_{\{k|1 \le k \le l\}} (q(k) + 5) \le \log n + 5 \cdot l$ bits. Since the depth of a perfect separator decomposition is $O(\log n)$, we obtain that $l = O(\log n)$ and the first part of the lemma follows.

Let T^{sep} be the separator tree defined by Protocol STAT_SEP. At the l'th level of the recursion, Protocol STAT_SEP acts on the subtrees $T^*(v)$, which correspond to the vertices v at depth l in T^{sep}. In particular, Protocol STAT_SEP acts on disjoint subtrees in each level of its recursion. When Protocol STAT_SEP is invoked on some subtree $T^*(v)$, it follows from Claims A.1 and A.3 that $O(|T'|)$ messages are sent by Protocol STAT_SEP. Since the depth of T^{sep} is at most $\lceil \log n \rceil$, the second part of the lemma follows. The third part of the lemma follow from the second parts of claims A.1 and A.3. ∎

B Description and analysis of Protocol SHUFFLE

Let $T' \in T^{subtrees}$, and let z be the separator chosen for T', i.e., $T' = T^*(z)$. Let us now give a formal description of Protocol SHUFFLE(T') by first describing Sub-protocol INIT(T'). Sub-protocol INIT(T') (as well as Protocol SHUFFLE(T')) is initiated at z. We distinguish between two cases. If T' is the whole tree T, then Sub-protocol INIT(T) merely initializes the labels $L^{sep}(v)$ of every vertex v to be $\langle 0 \rangle$. In the case, where $T' \in T^{subtrees}$ and T' is a proper subtree of T, let v be the forming separator (i.e., such that T' is one of the subtrees obtained by removing v from $T^*(v)$). Let l be the level of the separator v. We assume that v keeps the value $\omega_0^*(v)$ which is the value of $T^*(v)$ after the last application of Protocol SHUFFLE on a subtree containing v. For $T' \in T^{subtrees}$, if T' is not the whole tree, then Sub-protocol INIT(T') operates as follows. For every vertex u and every $-1 \le q \le \lceil \log n \rceil$, let $c_q(u)$ be the counter c_q at u (which is also used by Protocol STAT_SEP).

Protocol $\text{INIT}(T')$

1. The vertex z initiating this protocol calculates $|T'|$ through a convergecast operation on T', and delivers a message to v containing the value $|T'|$.
   Let q be the integer satisfying $|T'| \in J_q(\omega_0^*(v))$. (Note that by the balance invariant B2 there must exist such $-1 \leq q \leq \lceil \log n \rceil$ since $|T'| \leq |T^*(v)| < 2 \cdot \omega_0^*(v)$.)

2. The separator v broadcasts a message containing $\langle L^{sep}(v), c_q(v) \rangle$ to every vertex $u \in T'$.

3. Every vertex $u \in T'$ sets $L^{sep}(u) = L^{sep}(v) \circ \rho(T')$, where $\rho(T') = 3 \cdot 2^{q+2} + c_q(v)$, and sets $c_q(u) = 0$ for every $-1 \leq q \leq \lceil \log n \rceil$.

4. v updates its counter $c_q(v)$ to be $c_q(v) + 1$.

We are now ready to describe Protocol $\text{SHUFFLE}(T')$.

Protocol $\text{SHUFFLE}(T')$

1. The initiator z invokes $\text{INIT}(T')$.

2. Next, z sends a signal to the root of T', which in turn, invokes $\text{STAT\_SEP}(T')$.

3. Recursively calculate the weights of the subtrees $T^*(x)$ of the perfect-separator decomposition obtained by $\text{STAT\_SEP}(T')$. For each such subtree $T^*(x)$, it separator x initializes its counter $\omega_0^*(x)$ to be $|T^*(x)|$.

Lemma B.1 *Assuming that the balance invariants and the growth property are maintained throughout the dynamic scenario, after every application of Protocol* $\text{SHUFFLE}(T')$, *the following properties are satisfied for every $k \geq l$ and every two vertices u and w in T'.*
SHUFFLE **properties :**
Sh1: *If $u \neq w$ then $L^{sep}(u) \neq L^{sep}(w)$.*
Sh2: *If u is the level-k separator of w then $L^{sep}(u) = L_1^{sep}(w) \circ L_2^{sep}(w) \circ \cdots \circ L_k^{sep}(w)$.*
Sh3: *If $\rho(T') \in I_q$ for some q then $|T'| \in J_q(|\omega_0^*(v)|)$.*
Sh4: *For $k > l$, if $L_{k+1}^{sep}(u) \in I_q$ then $|T_{k+1}(u)| \in J_q(|T_k(u)|)$.*

Proof: After an application of Protocol $\text{SHUFFLE}(T')$, the SHUFFLE properties Sh1, Sh2 and Sh4 are clearly satisfied by Step 2 of Protocol $\text{INIT}(T')$, Step 2 of Protocol $\text{SHUFFLE}(T')$ and Lemma A.4. Let us now show that the SHUFFLE property Sh3 holds as well. By Steps 1 and 2 of Protocol $\text{INIT}(T')$, $\rho(T') = 3 \cdot 2^{q+2} + c_q(v)$, where q is such that $|T'| \in J_q(\omega_0^*(v))$. It is therefore left to prove that at any quiet time t and for every q, $3 \cdot 2^{q+2} + c_q(v) \in I_q$. Fix $-1 \leq q \leq \lceil \log n \rceil$ and fix a quiet time t. Let t_0 be the first quiet time after the last application of Protocol SHUFFLE invoked before time t on a subtree containing v. Let Γ be the set of SHUFFLE protocols invoked between time t_0 and time t on subtrees $T^i(v)$ satisfying (at the time the SHUFFLE protocol took place) $|T^i(v)| \in J_q(\omega_0^*(v))$. First note that by the first part of Claim A.2, at time t_0, the counter $c_q(v)$ satisfies $c_q(v) < 2^{q+1}$. Therefore, by Steps 1 and 3 of Protocol INIT, it is left to show that $|\Gamma| \leq 3 \cdot 2^{q+2} - 2^{q+1} = 10 \cdot 2^q$.

Fix some $\mathcal{S} \in \Gamma$ and let $T(\mathcal{S})$ be the subtree on which $\mathcal{S}$ was invoked. Let t^s be the time in which $\mathcal{S}$ was invoked and let t_0^s be the last time before t^s that some SHUFFLE protocol was invoked on a subtree containing $T(\mathcal{S})$. Clearly $t_0 \leq t_0^s \leq t^s \leq t$. Let $\phi(\mathcal{S})$ be the number of nodes joining $T(\mathcal{S})$ from time t_0^s to time t^s. Let ω^s (respectively, ω_0^s) be the number of nodes in $T(\mathcal{S})$ at time t^s (resp., t_0^s). We therefore have $\phi(\mathcal{S}) = \omega^s - \omega_0^s$. By the balance invariant B2, $\omega^s/2 < \omega_0^s$ and therefore $\omega_0^*(v)/2^{q+2} < \omega_0^s$. On the other hand, by the growth property, $\omega^s - \omega_0^s \geq (\gamma - 1) \cdot \omega_0^s$ and therefore, by the fact that $\gamma = \sqrt{5/4}$, we obtain

$$\phi(\mathcal{S}) \geq (\gamma - 1) \cdot \frac{\omega_0^*(v)}{2^{q+2}} \geq \frac{\omega_0^*(v)}{10 \cdot 2^{q+2}} \ .$$

By the balance invariant B2, $\sum_{\mathcal{S} \in \Gamma} \phi(\mathcal{S}) \le \frac{\omega_0^*(v)}{4}$ and therefore

$$|\Gamma| \cdot \frac{\omega_0^*(v)}{10 \cdot 2^{q+2}} \le \frac{\omega_0^*(v)}{4} \quad .$$

This yields that $|\Gamma| \le 10 \cdot 2^q$ as desired. The lemma follows. ∎

Lemma B.2 $\mathcal{MC}(\text{SHUFFLE}(T')) = O(|T'| \log |T'|)$.

Proof: Clearly, Protocol $\text{INIT}(T')$ incurs $O(|T'|)$ messages. Step 3 of Protocol $\text{SHUFFLE}(T')$ incurs $O(|T'| \log |T'|)$ messages, and Step 2 incurs $O(\mathcal{MC}(\text{STAT\_SEP}(T')))$. The lemma follows from the second part of Lemma A.5. ∎

C   Description and analysis of Protocol DYN_SEP

We now describe Protocol DYN_SEP, whose goal is to maintain a compact separator decomposition representation on T. Throughout the execution, every separator v keeps the value $\omega_0^*(v)$, equalling the value of $|T^*(v)|$ after the last application of Protocol STAT_SEP that v participated in. After every application of Protocol SHUFFLE on a subtree containing v, $\omega_0^*(v)$ is set to be $\omega^*(v) = |T^*(v)|$.

Protocol DYN_SEP maintains the following invariants for every level-l separator v and every two vertices u and w in T.

Dynamic invariants:
D1: If $u \ne w$, then $L^{sep}(u) \ne L^{sep}(w)$,
D2: If u is the level-k separator of w, then $L^{sep}(u) = L_1^{sep}(w) \circ L_2^{sep}(w) \circ \cdots \circ L_k^{sep}(w)$,
D3: $\omega^*(v) \in [\omega_0^*(v), \frac{5}{4}\omega_0^*(v)]$,
D4: Let $T^1(v), T^2(v) \cdots$ be the subtrees formed by v. For every $i \ge 1$, if $\rho(T^i(v)) \in I_q$ then $|T^i(v)| \in \widehat{J_q}(|\omega_0^*(v)|)$,
D5: The growth property: just before Protocol SHUFFLE is invoked on $T^*(v)$, we have $\gamma \cdot \omega_0^*(v) \le \omega^*(v)$.

Note that the dynamic invariants D3 and D4 are actually the balance invariants.

Protocol DYN_SEP operates as follows.

Protocol DYN_SEP

1. Invoke protocol MAINTAIN_W.

2. If a new vertex u is added as a child of a level-l separator v, then u becomes both the separator and the vertex which is responsible for the $l+1$'st-level subtree $T_u = \{u\}$. We consider u to be a child of v in the separator decomposition tree T^{sep} and apply Protocol SHUFFLE($\{u\}$), after which u is assigned the label $L^{sep}(u)$, and initialize its counters appropriately.

3. For every subtree $T^*(v)$, whenever v observes that $\tilde{\omega}^*(v) > \gamma \cdot \omega_0^*(v)$, it invokes SHUFFLE($T^*(v)$).

Lemma C.1 *The dynamic invariants are maintained at any quiet time throughout Protocol* DYN_SEP.

Proof: Fix some quiet time t and assume, by the inductive hypothesis, that the dynamic invariants are maintained until the last quiet time t_0 before time t. In particular, by the dynamic invariants D3, D4 and D5, the balance invariants B1 and B2 as well as the growth property are maintained until time t_0, and therefore by Lemma B.1, the SHUFFLE properties are maintained after every application of Protocol SHUFFLE carried out prior to time t. Since the labels may only change due to an application of

Protocol SHUFFLE, the dynamic invariants D1 and D2 are guaranteed to hold at time t by the SHUFFLE properties Sh1 and Sh2.

Fix a level-l separator v. Let us now consider the dynamic invariant D3. By Step 3 of Protocol DYN_SEP, $\tilde{\omega}^*(v) \leq \gamma \cdot \omega_0^*(v)$ at time t, and by Lemma 5.1, $|T^*(v)| \leq \gamma \cdot \tilde{\omega}^*(v)$. Since $\omega_0^*(v) \leq |T^*(v)|$ at all times, we have $\omega_0^*(v) \leq |T^*(v)| \leq \gamma^2 \cdot \omega_0^*(v)$, or in other words, $\omega_0^*(v) \leq \omega^*(v) \leq \gamma^2 \cdot \omega_0^*(v)$. The dynamic invariant D3 follows since $\gamma^2 = 5/4$.

Let us now consider the dynamic invariant D4. Let $T^1(v), T^2(v), \cdots$ be the subtrees formed by v. By dynamic invariant D2, for every i we can identify the subtree $T^i(v)$ with the value $\rho(T^i(v)) = L_{l+1}^{sep}(x_i)$. Fix some i and let $T' = T^i(v)$. Let $\mathcal{S}$ be the last application of Protocol SHUFFLE applied on a subtree containing T' before time t and let q be such that $\rho(T') \in I_q$ after $\mathcal{S}$ was invoked. Then, by the shuffle properties, after $\mathcal{S}$ was invoked, we have $|T'| \in J_q(\omega_0^*(v))$ and therefore $\omega_0' \in J_q(\omega_0^*(v))$. Note that at time t, the value $\rho(T')$ remains in I_q. As mentioned before, $\omega_0' \leq |T'| \leq \gamma^2 \cdot \omega_0'$ at time t and therefore the following is also satisfied at time t.

$$\frac{\omega_0^*(v)}{2^{q+1}} \leq \omega_0' \leq |T'| \leq \gamma^2 \cdot \omega_0' < 2 \cdot \omega_0' \leq \frac{\omega_0^*(v)}{2^{q-1}}. \tag{1}$$

In particular, $|T'| \in \widehat{J}_q(\omega_0^*(v))$ and therefore at time t, the dynamic invariant D4 is maintained as well. Finally, the dynamic invariant D5 follows from Step 3 of Protocol DYN_SEP and from the fact that at all times $\tilde{\omega}(T') \leq \omega(T')$. The lemma follows. ∎

Theorem C.2 *Assuming the leaf-increasing tree model, at any quiet time, Protocol* DYN_SEP *maintains a separator decomposition with the following complexities.*

1. $\mathcal{LS}(\text{DYN\_SEP}, n) = O(\log n)$

2. $\mathcal{MC}(\text{DYN\_SEP}, n) = O(n \log^3 n))$.

Proof: Fix a quiet time t. The fact that the labels given by Protocol DYN_SEP maintain a separator decomposition representation is clear by the dynamic invariants D1 and D2. Let us first prove that the separator decomposition maintained by Protocol DYN_SEP is a δ-separator decomposition. Let v be a separator and let $T^1(v), T^2(v), \cdots$ be the subtrees formed by v. Let $\mathcal{S}$ be the last application of Protocol SHUFFLE that was invoked before time t on a subtree containing v. Let $\omega^i(v)$ be the number of vertices in $T^i(v)$ at time t and let $\omega_0^*(v)$ (respectively, $\omega_0^i(v)$) be the number of vertices in $T^*(v)$ (resp., $T^i(v)$) when $\mathcal{S}$ was invoked. Recall that by applying $\mathcal{S}$, the subtree on which $\mathcal{S}$ was applied was decomposed by $\mathcal{S}$ into a perfect separator decomposition. Therefore, we have $\omega_0^i(v) \leq \omega_0^*(v)/2$. By the dynamic invariant D3, at time t we have $\omega^*(v) \leq \frac{5}{4}\omega_0^*(v)$ and in particular, $\omega^i(v) - \omega_0^i(v) \leq \omega^*(v) - \omega_0^*(v) \leq \omega_0^*(v)/4$. Therefore, $\omega^i(v) \leq 3/4 \cdot \omega_0^*(v) \leq 3/4 \cdot \omega^*(v)$. Since this inequality is satisfied for every separator v, we get that the separator decomposition maintained by Protocol DYN_SEP is a δ-separator decomposition for $\delta = 3/4$.

We now turn to prove that for every vertex v, the number of bits in $L^{sep}(v)$ is $O(\log n)$. Fix a level-l vertex v. Clearly, $L^{sep}(v)$ contains l fields. For every $1 \leq k \leq l$, let $m(k) = |T_k(v)|$ and let $q(k)$ be such that $L_k^{sep}(v) \in I_{q(k)}$. By the dynamic invariant D4 and by the fact that $\omega_0^*(u) \leq \omega^*(u) = |T^*(u)|$ for every separator u, we obtain that $m(k) \leq m(k-1)/2^{q(k)-1}$ for every $1 \leq k \leq l$. Therefore $\prod_{\{k|1 \leq k \leq l\}} 2^{q(k)-1} \leq m(1) = n$, yielding $\sum_{\{k|1 \leq k \leq l\}}(q(k) - 1) \leq \log n$. Since $L_k^{sep}(v) \in I_{q(k)}$, we get that $L_k^{sep}(v)$ can be encoded using $q(k) + 5$ bits and therefore $L^{sep}(v)$ can be encoded using $\sum_{\{k|1 \leq k \leq l\}}(q(k) + 5) \leq 6 \cdot l + \sum_{\{k|1 \leq k \leq l\}}(q(k) - 1) \leq 6 \cdot l + \log n$ bits. Since the depth of a δ-separator decomposition is $O(\log n)$, we get that $l = O(\log n)$. It follows that $\mathcal{LS}(\text{DYN\_SEP}, n) = O(\log n)$.

We now turn to prove part 2. By Lemma 5.1, the first step of Protocol DYN_SEP incurs $O(n \log^3 n)$ messages during the dynamic scenario. Throughout the execution, Step 2 of Protocol DYN_SEP incurs $O(n)$ messages since SHUFFLE($\{u\}$) incurs $O(1)$ messages for every vertex u. It is therefore left to show that Step 3 of Protocol DYN_SEP incurs $O(n \log^3 n)$ messages. In fact, we show that Step 3 incurs $O(n \log^2 n)$ messages throughout the execution.

At any given time t and for every subtree $T' \in T^{subtrees}$, let $\tilde{\omega}(T')$ be the value of the counter of $r(T')$ which is used in order to estimate $|T'|$. In addition, let $\omega_0(T')$ be the value of $|T'|$ after the last application of Protocol SHUFFLE before time t on a subtree containing T'. For every shuffle $\mathcal{S}$, let $T(\mathcal{S})$ be the tree on which $\mathcal{S}$ is applied and let $\omega(\mathcal{S})$ be the number of messages incurred by $\mathcal{S}$. Let Γ be the collection of shuffles. By Lemma B.2, for every SHUFFLE $\mathcal{S}$, $\omega(\mathcal{S}) = O(|T(\mathcal{S})| \log |T(\mathcal{S})|)$. Therefore, the number of messages incurred by Steps 3 of Protocol DYN_SEP is bounded from above by $O(\sum_{\mathcal{S} \in \Gamma} |T(\mathcal{S})| \log |T(\mathcal{S})|)$.

For every subtree $T' \in T^{subtrees}$, let $\psi(T') = \tilde{\omega}(T') - \omega_0(T')$. Note that by Step 3 of Protocol SHUFFLE, for every $T' \in T^{subtrees}$, $\psi(T') \geq 0$ at all times. Let us now examine the behavior of the value $\Psi = \sum_{T' \in T^{subtrees}} \psi(T')$ during the execution.

First suppose a vertex v is added to the tree. Since there are $l(v) = O(\log n)$ subtrees $T' \in T^{subtrees}$ that v belongs to, Ψ increases by $O(\log n)$.

Now suppose a shuffle $\mathcal{S}$ is invoked on T' by Step 3 of Protocol DYN_SEP. In this case, just before $\mathcal{S}$ is applied, we have $\psi(T') = \tilde{\omega}(T') - \omega_0(T') > (\gamma - 1) \cdot \omega_0(T') = \Omega(\omega_0(T'))$. On the other hand, by the dynamic invariant D3, $|T'| = \Theta(\omega_0(T'))$. Therefore, just before $\mathcal{S}$ is applied, we have $\psi(T') = \Omega(|T'|)$.

Note that after $\mathcal{S}$ is invoked, for every subtree $T'' \in T^{subtrees}$, $\psi(T'')$ may only decrease. Moreover, after $\mathcal{S}$ is invoked, we have $\psi(T') = 0$. We therefore get that after $\mathcal{S}$ is invoked, Ψ decreases by $\Omega(|T'|)$. Consequently, at all quiet times, $\Psi \leq O(n \log n) - \Omega(\sum_{\mathcal{S} \in \Gamma}(|T(\mathcal{S})|))$. On the other hand, by the definition of Ψ, we have that $\Psi \geq 0$ at all times and therefore $\sum_{\mathcal{S} \in \Gamma}(|T(\mathcal{S})|) \leq O(n \log n)$. Consequently, $O(\sum_{\mathcal{S} \in \Gamma} |T(\mathcal{S})| \log |T(\mathcal{S})|) \leq O(n \log^2 n)$ and the second part of the lemma follows. ∎

D   Applications to ancestry, NCA and distance labeling schemes

We have sketched the ideas behind our dynamic ancestry labeling scheme π_{Dyn_Anc} and our dynamic routing schemes for the leaf-increasing tree model. We now sketch the ideas behind our dynamic NCA labeling schemes and our extended distance labeling scheme.

Improved NCA labeling schemes on dynamic trees: We first sketch a static labeling scheme π_{Stat_NCA} supporting the NCA relation on trees. The labels assigned by π_{Stat_NCA} are almost identical to the labels given by the corresponding NCA labeling scheme in [18], with the restriction that in [18] they use the DFS ancestry labeling scheme as a building block to their NCA scheme while we use our π_{Stat_Anc} ancestry scheme instead. Then, using our dynamic ancestry labeling scheme π_{Dyn_Anc} and Protocol HEAVY_CHILD from [17] (which maintains a dynamic δ-heavychild decomposition), we extend the static NCA labeling scheme π_{Stat_NCA} to the leaf-increasing tree model.

Let us note that the NCA problem assumes that unique identifiers in the range $[1, poly(n)]$ are assigned to the vertices of the tree. One can implement the above in the leaf-increasing tree model using either the scheme of [1] or the scheme of [15], which assign disjoint identifiers in the range $[1, 2n]$ to the vertices of the dynamically growing tree using message complexity $O(n \log^2 n)$. Alternatively, the scheme will also work assuming that once a vertex joins the tree, a unique identifier (in the range $[1, poly(n)]$) is assigned to it by an adversary.

Extended distance labeling schemes on dyanmic trees: In [16], they give two β-approximate distance labeling schemes on dynamic trees which operate under two dynamic models. In both dynamic models, the vertices are fixed but the edge weights may change (as long as they remain positive). Informally, both schemes are based on the following principle. In a preprocessing stage, a (static) separator decomposition is calculated on the tree. In this decomposition, each vertex belongs to $O(\log n)$ subtrees, one for each level of the recursion. A mechanism for estimating the distance to the root is applied separately to each of the decomposed subtrees. Therefore, each vertex v participates in $O(\log n)$ such protocols, each corresponding to a subtree that v belongs to. This enables v to maintain estimates to the distances between v and the roots of these subtrees. These distance estimates are then encoded

in v's label. The distance between any two nodes in the dynamic tree can be retrieved from their corresponding lists of estimates, which are encoded in their labels.

Using our dynamic separator decomposition π_{Dyn_Sep} for the leaf-dynamic model (see next section), and applying the above mentioned principle on the resulted dynamic separator decomposition, the schemes in [16] can be modified to operate correctly under more general dynamic models, allowing also leaves to be either added or removed from the tree. Moreover, the extended dynamic schemes incur only an extra additive $O(\log^3 n)$ factor to the amortized message complexity of the original schemes, where n is the current number of nodes in the tree.

E Extending the dynamic schemes to the leaf-dynamic tree model: sketch

We now informally describe how to extend our extended dynamic compact separator decomposition scheme $\hat{\pi}$ from the increasing-dynamic tree model to the leaf-dynamic model, i.e., how to support also deletions of leaves. Let us note that, in fact, this method holds for any of our dynamic schemes.

The idea behind the adaptation is similar to the one presented in [15]. We first mimic the execution of $\hat{\pi}$ on the scenario in which deletions never occur, i.e., messages are not sent to deleted vertices and whenever an operation is invoked on a subtree T' in which the number of nodes in T' is calculated, we calculate instead the number of nodes in T' including the deleted vertices. This is implemented by keeping at every vertex v, a variable $m'(v)$, for each subtree T' that v belongs to. For each such subtree T', it is maintained throughout the scenario that $\sum_{v \in T'} m'(v) = m(T')$, where $m(T')$ is the number of existing nodes in T' together with the deleted ones. Therefore, instead of calculating the number of nodes in T', we calculate $\sum_{v \in T'} m'(v)$. Note, however, that if the tree size falls significantly, then the current labels which are small with respect to $m(T)$, might be too large with respect to the current number of nodes in the forest. We therefore run, in parallel to π, the protocol from [15] for estimating the number of topological changes in the tree. (This protocol incurs $O(\log^2 n)$ amortized message complexity.) Every $\Theta(n)$ topological changes, we initialize the labels and data structures of the vertices by running π on the imaginary scenario in which we start with just the root and vertices are added to the tree one by one constructing T_0. We then restart Scheme π again as if the imaginary scenario really happened.

Let Π be the resulted scheme operating in the leaf-dynamic tree model. The following theorem is obtained.

Theorem E.1 *Assuming the leaf-dynamic tree model, our extended scheme Π is a dynamic compact separator decomposition schemes with message complexity $O(\sum_i \log^3 n_i)$, where n_i is the number of nodes in the tree, after the i'th topological change.*

Corollary E.2 *Let F be either 1) the routing function, 2) the ancestry relation or 3) the NCA function. Let π_F be our dynamic labeling scheme supporting f in the leaf-increasing model, and let Π_F be the corresponding extended scheme for the leaf-dynamic model. Then Π_F is a correct dynamic F-labeling scheme with the same asymptotic label size as π_F and message complexity $O(\sum_i \log^3 n_i)$, where n_i is the number of nodes in the tree, after the i'th topological change.*