Distributed Construction of Light Networks

Michael Elkin*, Arnold Filtser†, and Ofer Neiman‡

Ben-Gurion University of the Negev. Email: {elkinm,arnoldf,neimano}@cs.bgu.ac.il

Abstract

A *t-spanner* H of a weighted graph $G = (V, E, w)$ is a subgraph that approximates all pairwise distances up to a factor of t. The *lightness* of H is defined as the ratio between the weight of H to that of the minimum spanning tree. An (α, β)-*Shallow Light Tree* (SLT) is a tree of lightness β, that approximates all distances from a designated root vertex up to a factor of α. A long line of works resulted in efficient algorithms that produce (nearly) optimal light spanners and SLTs.

Some of the most notable algorithmic applications of light spanners and SLTs are in distributed settings. Surprisingly, so far there are no known efficient distributed algorithms for constructing these objects in general graphs. In this paper we devise efficient distributed algorithms in the CONGEST model for constructing light spanners and SLTs, with near optimal parameters. Specifically, for any $k \geq 1$ and $0 < \epsilon < 1$, we show a $(2k-1)\cdot(1+\epsilon)$-spanner with lightness $O(k \cdot n^{1/k})$ can be built in $\tilde{O}\left(n^{\frac{1}{2}+\frac{1}{4k+2}} + D\right)$ rounds (where $n = |V|$ and D is the hop-diameter of G). In addition, for any $\alpha > 1$ we provide an $(\alpha, 1 + \frac{O(1)}{\alpha-1})$-SLT in $(\sqrt{n} + D) \cdot n^{o(1)}$ rounds. The running time of our algorithms cannot be substantially improved.

We also consider spanners for the family of doubling graphs, and devise a $(\sqrt{n} + D) \cdot n^{o(1)}$ rounds algorithm in the CONGEST model that computes a $(1+\epsilon)$-spanner with lightness $(\log n)/\epsilon^{O(1)}$. As a stepping stone, which is interesting in its own right, we first develop a distributed algorithm for constructing nets (for arbitrary weighted graphs), generalizing previous algorithms that worked only for unweighted graphs.

*Supported in part by ISF grant (2344/19).

†Supported in part by ISF grant (1817/17) and in part by BSF grant 2015813

‡Supported in part by ISF grant (1817/17) and in part by BSF grant 2015813

Object	Distortion	Lightness	Size	Run time
Spanner	$(2k-1)\cdot(1+\varepsilon)$	$O(k\cdot n^{1/k})$	$O(k\cdot n^{1+1/k})$	$\tilde{O}\left(n^{\frac{1}{2}+\frac{1}{4k+2}}+D\right)$
SLT	$1+\frac{O(1)}{\alpha-1}$	α	NA	$\tilde{O}\left(\sqrt{n}+D\right)\cdot\text{poly}\left(\frac{1}{\alpha-1}\right)$
(γ,β)-net	NA	NA	NA	$(\sqrt{n}+D)\cdot 2^{\tilde{O}(\sqrt{\log n}\cdot\log\frac{\beta}{\gamma-\beta})}$
Spanner	$1+\varepsilon$	$\varepsilon^{-O(\text{ddim})}\cdot\log n$	$n\cdot\varepsilon^{-O(\text{ddim})}\cdot\log n$	$(\sqrt{n}+D)\cdot\varepsilon^{-\tilde{O}(\sqrt{\log n}+\text{ddim})}$

Table 1: A summary of our main results. Here n is the number of vertices, $k\geq 1$ and $\alpha\geq 1$ are parameters and $0<\varepsilon<1$ is a constant. For the nets $\gamma>\beta>0$. All results are in the CONGEST model.

1 Introduction

Let $G=(V,E,w)$ be a graph with edge weights $w:E\rightarrow\mathbb{R}_+$. For $u,v\in V$, denote by $d_G(u,v)$ the shortest path distance in G between u,v with respect to these weights. A subgraph $H=(V,E')$ with $E'\subseteq E$ is called a *t-spanner* of G, if for all $u,v\in V$, $d_H(u,v)\leq t\cdot d_G(u,v)$. The parameter t is called the *stretch* of H. The most relevant and studied attributes of a t-spanner are its sparsity (i.e., the number of edges $|E'|$), and the total weight of the edges $w(H)=\sum_{e\in E'}w(e)$. Since any spanner with finite stretch must be connected, its weight is at least the weight of the Minimum Spanning Tree (MST) of G, and the *lightness* of H is defined as $\frac{w(H)}{w(MST)}$.

Given a weighted graph $G=(V,E,w)$ with a designated root vertex rt, an (α,β)-*Shallow-Light Tree* (SLT) T of G [ABP92, KRY95] is a spanning tree which has lightness β, and approximates all distances from rt to the other vertices up to a factor of α.

In this paper we focus on the distributed CONGEST model of computation, where each vertex of the graph G hosts a processor, and these processors communicate with each other in discrete rounds via short messages on the graph edges (typically the message size is $O(1)$ RAM words). We devise efficient distributed algorithms that construct light spanners and shallow-light trees for general graphs, and also light spanners for doubling graphs. See Table 1 for a succinct summary.

1.1 Light Spanners for General Graphs

Spanners are a fundamental combinatorial object. They have been extensively studied and have found numerous algorithmic applications [Awe85, PS89, PU89, ADD+93, Coh98, ACIM99, EP01, BS07, Elk07, EZ06, TZ06, Pet09, DGPV08, Pet10, MPVX15, AB16, EN19]. The basic greedy algorithm [ADD+93], for a graph with n vertices and any integer $k\geq 1$, provides a $(2k-1)$-spanner with $O(n^{1+1/k})$ edges, which is best possible (assuming Erdos' girth conjecture). Light spanners have received much attention in recent years [CDNS95, ES16, ENS15, Got15, CW18, BFN19, ADF+17, BLW17, FN18, BLW19], and are particularly useful in a distributed setting; efficient broadcast protocols, network synchronization and computing global functions [ABP90, ABP92], network design [MP98, SCRS01] and routing [WCT02] are a few examples. The state-of-the-art is a $(2k-1)\cdot(1+\varepsilon)$-spanner of [CW18] with lightness $O(n^{1/k})$, for any constant $0<\varepsilon<1$. In [FS16] it was shown that the greedy algorithm is existentially optimal. Hence it also achieves this tradeoff.

The greedy algorithm provides a satisfactory answer to the existence of sparse and light spanners, but not to efficiently producing such a spanner (because the greedy algorithm inherently has large running time). Indeed, the problem of devising fast algorithms for constructing spanners is

very important for many algorithmic applications. [ES16] showed a near-linear time algorithm that constructs a $(2k-1)\cdot(1+\varepsilon)$-spanner with $O(k\cdot n^{1+1/k})$ edges and lightness $O(k\cdot n^{1/k})$. The sparsity and lightness were improved (still in near-linear time) in [ADF+17] to $O(\log k\cdot n^{1+1/k})$ and $O(\log k\cdot n^{1/k})$ respectively. In the distributed setting, [BS07] devised a randomized algorithm for a $(2k-1)$-spanner with $O(k\cdot n^{1+1/k})$ edges in $O(k)$ rounds in the CONGEST model. This was recently improved for unweighted graphs by [MPVX15, EN19] to $O(n^{1+1/k})$ edges. However, the weight of these spanners may be very large. Surprisingly, none of the previous works in the CONGEST model has a bound on the *lightness* of spanners for general graphs.

Unlike the sparsity of spanners, which can be preserved via a local algorithm, the lightness is a global measure. Indeed, we observe that the lower bound of [SHK+12] on the number of rounds required for any polynomial approximation of the weight of MST, implies a lower bound for computing light spanners. In particular, for a graph with n vertices and hop-diameter[1] D, any CONGEST algorithm requires at least $\tilde{\Omega}(\sqrt{n}+D)$ rounds for computing a light spanner (with any polynomial lightness). [2] See Theorem 6.

Our result. We provide the first algorithm with sub-linear number of rounds for constructing light spanners for general graphs in the CONGEST model. Specifically, for any integer parameter $k\geq 1$ and constant $0<\varepsilon<1$, we devise a randomized algorithm that w.h.p. outputs a $(2k-1)\cdot(1+\varepsilon)$-spanner with $O(k\cdot n^{1+1/k})$ edges and lightness $O(k\cdot n^{1/k})$, within $\tilde{O}\left(n^{\frac{1}{2}+\frac{1}{4k+2}}+D\right)$ rounds in the CONGEST model, thus nearly matching the lower bounds. See Theorem 2.

1.2 Shallow-Light Trees

Shallow-Light trees are widely used for various distributed tasks, such as network design, broadcasting in ad-hoc networks and multicasting [PV04, BDS04, YCC06]. In [KRY95], an optimal tradeoff between the lightness of the SLT to the stretch of the root distances was obtained. Specifically, for any $\alpha>1$ they obtained an SLT with lightness α and stretch $1+\frac{2}{\alpha-1}$. In addition, [KRY95] exhibited an efficient algorithm for constructing such a tree in near-linear time, and also in $O(\log n)$ rounds in the PRAM (CREW) model. However, their techniques are inapplicable to the CONGEST model, and it remained open whether an SLT can be built efficiently in this model. (Roughly speaking, [KRY95] uses pointer jumping that require massive communication between multiple pairs of distant vertices. Hence this approach appears to be unsuitable for the CONGEST model.)

Our result. We answer this open question in the affirmative, and devise a distributed deterministic algorithm, that for any $\alpha>1$, outputs an SLT with lightness α and stretch $1+\frac{O(1)}{\alpha-1}$, within $\tilde{O}\left(\sqrt{n}+D\right)\cdot\mathrm{poly}\left(\frac{1}{\alpha-1}\right)$ rounds. See Theorem 1. Recall that by [Elk04], any distributed SLT algorithm requires at least $\tilde{\Omega}(\sqrt{n}+D)$ rounds. Thus our result is nearly optimal.

[1] The hop diameter of a weighted graph is the diameter of the underlying unweighted graph.

[2] The notations $\tilde{O}(\cdot)$ and $\tilde{\Omega}(\cdot)$ hide $\mathrm{polylog}(n)$ factors.

1.3 Light Spanners for Doubling Graphs

A graph G has *doubling dimension* ddim if for every vertex $v \in V$ and radius $r > 0$, the ball[3] $B_G(v, 2r)$ can be covered by 2^{ddim} balls of radius r. For instance, a d-dimensional ℓ_p space has ddim $= \Theta(d)$, and every graph with n vertices has ddim $= O(\log n)$. This is a standard and well-studied notion of "growth restriction" on a graph [Ass83, GKL03, HM06], and it is believed that such graphs occur often in real-life networks and data [TSL00, NZ02]. One notable motivation for light spanners in doubling graphs[4] is their application for polynomial approximation schemes for the traveling salesperson and related problems (see, e.g., [Kle05, Got15]). While spanners with $1+\varepsilon$ stretch and constant lightness have been known to exists in low dimensional Euclidean space for a while [DHN93, ADD+93], only recently such $(1+\varepsilon)$-spanners with constant lightness $(\text{ddim}/\varepsilon)^{O(\text{ddim})}$ have been discovered for doubling graphs [Got15]. The lightness was improved by [BLW19] to the optimal $(1/\varepsilon)^{O(\text{ddim})}$.

Essentially all algorithms for constructing spanners for doubling graphs use *nets* in their construction. An (α, β)-*net* of a graph is a set $N \subseteq V$ which is both α-covering: for all $u \in V$ there is $v \in N$ with $d_G(u, v) \leq \alpha$, and β-separated: for all $x, y \in N$, $d_G(x, y) > \beta$. The standard definition of a net is when $\alpha = \beta$, but we shall allow $\alpha > \beta$ as well. The usefulness of nets in doubling graphs stems from the fact that any net restricted to a ball of certain radius has a small cardinality. While a simple greedy algorithm yields a net, it is not suitable for distributed models due to it being inherently sequential.

A *ruling set* is a net in an unweighted graph. There have been several works that compute a ruling set in distributed settings. In [AGLP89], a deterministic algorithm for a $(k \log n, k)$-ruling set running in $O(k \log n)$ rounds was developed, and a tradeoff extending this result was shown in [SEW13]. A consequence of the work of [AGLP89] provides a (k, k)-ruling set computed within $k \cdot 2^{\tilde{O}(\sqrt{\log n})}$ rounds, the running can be improved to $O(k \cdot \text{poly} \log n)$ using the recent breakthrough of [RG19]. A randomized algorithm for a (k, k)-ruling set was given in [Lub86] with $O(k \log n)$ rounds, and the running time was improved for graphs of small maximum degree in [BEPS12, Gha16]. However, all these results apply only for unweighted graphs. The problem of efficiently constructing a net in distributed models remained open.

Our results. We design a randomized distributed algorithm, that for a given graph with n vertices and hop-diameter D and any $0 < \beta < \alpha < 2\beta$, w.h.p. finds an (α, β)-net within $(\sqrt{n} + D) \cdot 2^{\tilde{O}(\sqrt{\log n} \cdot \log \frac{\beta}{\alpha - \beta})}$ rounds in the CONGEST model, see Theorem 3. We show that the running time must be at least $\tilde{\Omega}(\sqrt{n} + D)$ for general graphs, via a reduction to the problem of approximating the MST weight. So our running time is best possible (up to lower order terms). See Theorem 7. However, we do not know if a faster algorithm is achievable when the input graph has a constant doubling dimension.

Then, we utilize this algorithm for constructing nets, and devise a randomized algorithm that for a graph with doubling dimension ddim and any $0 < \varepsilon < 1$, w.h.p. produces a $(1+\varepsilon)$-spanner with lightness $\varepsilon^{-O(\text{ddim})} \cdot \log n$ in $(\sqrt{n} + D) \cdot \varepsilon^{-\tilde{O}(\sqrt{\log n} + \text{ddim})}$ rounds.

[3] A ball is defined as $B_G(v, r) = \{u \in V \ : \ d_G(u, v) \leq r\}$.

[4] A graph family is called *doubling* if its members have constant doubling dimension.

1.4 Overview of Techniques

In this section we provide an overview of the algorithms, techniques and ideas used in the paper.

Eulerian Tour of the MST. Let T be the MST. The first step in both our constructions of an SLT and a light spanner for general graphs is a distributed computation of an Eulerian traversal $\mathcal{L}$ of T. As an outcome of this computation, each vertex knows all its visiting times in $\mathcal{L}$. Our algorithm is a simplification of a similar algorithm from [EN18].

Light Spanner for General Graphs. From a high level, our approach is similar to the algorithms of [CDNS95, ES16, ENS15]. We divide the graph edges into $O(\log n)$ buckets, according to their weight. Denote by L the weight of the Eulerian traversal $\mathcal{L} = \{x_0, x_1, \ldots, x_{2n-2}\}$ of the MST (each vertex can appear several times). In the lowest level, we have all the edges of weight at most L/n. For this bucket we use the distributed spanner of [BS07] for weighted graphs.

Consider the i-th bucket E_i, where all edges have weight in $\left(\frac{L}{(1+\varepsilon)^{i+1}}, \frac{L}{(1+\varepsilon)^i}\right]$. We use the MST traversal $\mathcal{L}$ to divide the graph into $O(\frac{(1+\varepsilon)^i}{\varepsilon})$ clusters of diameter $\frac{\varepsilon \cdot L}{(1+\varepsilon)^i}$. Next, define an unweighted cluster graph $\mathcal{G}_i$ whose vertices are the clusters, and inter-cluster edges are taken only from E_i.

We then simulate the spanner algorithm of [EN19] for unweighted graphs on $\mathcal{G}_i$, and obtain a spanner $\mathcal{H}_i$. For every edge $e \in \mathcal{H}_i$, we add a corresponding edge $e' \in E_i$ to the final spanner. The main technical challenge is this simulation, which is non-trivial since the communication graph G is not the graph $\mathcal{G}_i$ for which we want a spanner. To resolve this issue, we distinguish between small and large clusters; for the former we use $\mathcal{L}$ to pipeline information inside the clusters, while for the latter we convergecast all the relevant information to a single vertex, and then broadcast the decisions made by this vertex to the entire graph. For this approach to be efficient we need to refine the partition to clusters, so that small clusters will have bounded hop-diameter. One also has to ensure that there are few large clusters, as otherwise the convergecast and broadcast would be too expensive.

Shallow Light Tree (SLT). Our algorithm for constructing SLT is employs as subroutines algorithms for constructing the MST T and a shortest path tree (SPT) rooted in rt. (This is also the case for other algorithms for this problem [ABP92, KRY95].) As currently the fastest known exact SPT algorithms [GL18, Elk17a] require more than $\tilde{O}(\sqrt{n} + D)$ rounds, we use instead an approximate SPT , T' [HKN16, EN17, BKKL17]. Our basic strategy (following [ABP92, KRY95]) is to choose a subset of vertices called break points (BP) on the Eulerian traversal $\mathcal{L}$. Then we construct a subgraph H by taking T, and adding to H the unique path in T' from rt to every break point $v \in$ BP. The SLT is computed as yet another approximate SPT rooted in rt, but now using H edges only.

Ideally, we would like to choose BP $= \{\mathrm{x}_0, \mathrm{x}_{\mathrm{i}_1}, \mathrm{x}_{\mathrm{i}_2}, \ldots\}$ such that (1) every pair of consecutive points $x_{i_j}, x_{i_{j+1}} \in$ BP are far from one another, specifically $d_{\mathcal{L}}(x_{i_j}, x_{i_{j+1}}) > \varepsilon \cdot d_G(rt, x_{i_{j+1}})$, and (2) every node $x_q \in \mathcal{L}$ has a nearby break point $x_{i_j} \in$ BP, specifically $d_{\mathcal{L}}(x_{i_j}, x_q) \leq \varepsilon \cdot d_G(rt, x_q)$. The first condition is used to bound the lightness, while the second condition is used to bound the stretch.

The choice of BP described above can be easily performed in a greedy manner by sequentially traversing the nodes in $\mathcal{L}$. Unfortunately, one cannot implement this sequential algorithm efficiently

in a distributed setting. Instead, we break $\mathcal{L}$ into $O(\sqrt{n})$ intervals, each containing at most $\sqrt{n}$ nodes. We add the first node in each interval to a temporary break point set BP'. Using these temporary break points as anchors, we perform the sequential algorithm simultaneously in all intervals, and add (permanent) break points. Finally, we broadcast BP' to rt, which performs a local computation in order to sparsify this set. Specifically, it chooses a subset of BP' to serve as permanent break points using the sequential algorithm, and broadcasts the chosen break points to the entire network. Intuitively, we are building a separate SLT for the set BP', which filters out some of its points. We show that this two-step choice of break points is up to a constant factor in the lightness as good as the sequential (one-step) choice of breakpoints.

Net Construction. Our algorithm for an (α, β)-net imitates, on a high level, previous ruling sets algorithms (like [Lub86, MRSZ11]).[5] Ideally, the net construction works as follows. Initially, all vertices are active. In each round, sample a permutation π. Each (active) vertex v which is the first in the permutation with respect to (w.r.t.) its β-neighborhood joins the net. Every vertex for which some vertex from its α-neighborhood joined the net, becomes inactive. Repeat until all vertices become inactive.

In order to check whether a vertex is the first in the permutation w.r.t. its β-neighborhood, we use Least Element (LE) lists [Coh97]. Given a permutation π, a vertex u belongs to the LE list of a vertex v, if u is the first in the permutation among all the vertices at distance at most $d_G(v,u)$ from v. In particular, given the LE list of v, we can check whether it should join the net or not. Efficient distributed computation of an LE list is presented in [FL16]. However, rather than computing the list w.r.t. the graph G, [FL16] compute LE list w.r.t. an auxiliary graph H that approximates G distances up to a $1+\varepsilon$ factor. Fortunately, we can cope with the approximation by taking $\alpha > (1+\varepsilon)\beta$. Once we compute the lists and choose which vertices will be added to the net, we compute an (approximate) shortest path tree rooted in the net points. All vertices at distance at most α from net points become inactive. This concludes a single round. After $O(\log n)$ rounds all vertices become inactive w.h.p.. The running time is dominated by the LE lists computations.

Light Spanner for Doubling Metrics. The basic idea for constructing spanners for doubling metrics is quite simple and well known. For every distance scale Δ, construct an (α, β)-net N_Δ where $\alpha, \beta \approx \varepsilon\Delta$, and connect by a shortest path every pair of net points at distance at most Δ. The stretch bound follows standard arguments, based on the covering property of nets. To prove lightness, we will use a packing argument, stating that every net point has at most $\varepsilon^{-O(\mathrm{ddim})}$ other net points at distance Δ, and every net point must contribute to the MST weight at least $\varepsilon \cdot \Delta$.

The main issue is implementing this algorithm efficiently in the CONGEST model. The main obstacle is to connect nearby net points. The problem is that the shortest path between nearby net points may contain many vertices, and so we cannot afford to add these sequentially. We resolve this issue by conducting a Δ-bounded multi-source approximate shortest paths (from each net point) based on *hopsets*. Roughly speaking, a hopset is a set of (virtual) edges added to the graph, so that every pair has an approximate shortest path containing few edges. We use the *path-reporting* hopsets of [EN16], and so the actual paths are added to the spanner. The running time is indeed bounded: we use the packing property of nets to show that every vertex participates in a small number of such approximate shortest path computations.

[5] In fact, these papers showed algorithms for Maximal Independent Sets (MIS), but a (k,k)-ruling set is an MIS for the graph G^k.

1.5 Related Work

In the distributed LOCAL model[6], [DPP06] devised light spanners for a certain graph family, called *unit ball graphs in a doubling metric space*[7]. Specifically, they showed an $O(\log^* n)$ rounds algorithm for a $(1+\varepsilon)$-spanner with lightness $(1/\varepsilon)^{O(\mathrm{ddim})} \cdot \log \Lambda$, where Λ is the aspect ratio of G (the ratio between the largest to smallest edge weights). We note that to obtain such a low number of rounds, they imposed restrictions on both the distributed model and the graph family.

1.6 Organization

In Section 3 we devise an Eulerian traversal of the MST, which will be used in the following sections. In Section 4 we present our distributed construction of an SLT. In Section 5 we show our light spanners for general graphs. The construction of nets for general graphs is shown in Section 6, and their application to light spanners for doubling graphs is in Section 7. Finally, the lower bounds are in Section 8.

2 Preliminaries

Let $G = (V, E, w)$ be a weighted graph with n vertices, and let d_G be the induced shortest path metric with respect to the weights. We assume that the minimal edge weight is 1, and that the maximal weight is $\mathrm{poly}(n)$. For $v \in V$ denote by $N(v) = \{u \in V \,:\, \{u, v\} \in E\}$ its set of neighbors, and by $N^+(v) = N(v) \cup \{v\}$. For a set $C \subseteq V$, the induced graph on C is $G[C]$. The *weak diameter* of C is $\max_{u,v \in C}\{d_G(u, v)\}$ and its strong diameter is $\max_{u,v \in C}\{d_{G[C]}(u, v)\}$. The hop-diameter of G is defined as its diameter while ignoring the weights.

In the CONGEST model of distributed computation, the graph G represents a network, and every vertex initially knows only the edges incident on it. Communication between vertices occurs in synchronous *rounds*. On every round, each vertex may send a small message to each of its neighbors. Every message has size at most $O(\log n)$ bits. The time complexity is measured by the number of rounds it takes to complete a task (we assume local computation does not cost anything). Often the time depends on n, the number of vertices, and D, the *hop-diameter* of the graph. The following lemma formalizes the broadcast ability of a distributed network (see, e.g., [Pel00]).

Lemma 1. *Suppose every $v \in V$ holds m_v messages, each of $O(1)$ words[8], for a total of $M = \sum_{v \in V} m_v$. Then all vertices can receive all the messages within $O(M + D)$ rounds.*

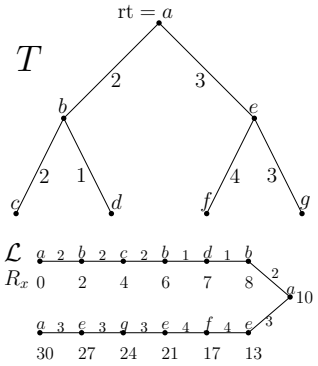
A Breadth First Search (BFS) tree τ of G of hop-diameter D (ignoring the weights) can be computed in $O(D)$ rounds. Since all our algorithms have a larger running time, we always assume that we have such a tree at our disposal.

[6] The LOCAL model is similar to CONGEST, but the size of messages is not bounded.

[7] A unit ball graph is a graph whose vertices lie in a metric space, and edges connect vertices of distance at most 1. In this scenario the metric is doubling.

[8] We assume a word size is $\log n$ bits.

3 Eulerian Tour of the MST



Let $G = (V, E, w)$ be a weighted graph on n vertices with hop-diameter D. Let T be the minimum spanning tree of G with a root vertex $rt \in V$. We compute an Eulerian path $\mathcal{L} = \{rt = x_0, x_1, \ldots, x_{2n-2}\}$ drawn by taking a preorder traversal of T. The order between the children of a vertex is determined using their id. We remark that in [EN18] it was described how to compute a DFS search of a tree in $\tilde{O}(\sqrt{n} + D)$ rounds. However, that paper also had the property that each vertex uses at most $O(\log n)$ words of memory. We give the full details here for completeness, and also since the presentation is somewhat simpler without the bound on the memory usage.

For a vertex $x \in \mathcal{L}$, let $R_x = d_{\mathcal{L}}(rt, x)$ be the time visiting x in $\mathcal{L}$. The total length of the traversal $\mathcal{L}$ (that is $R_{x_{2n-2}}$) equals $2 \cdot w(T)$. The number of appearances of each vertex $v \in V$ in $\mathcal{L}$ equals to its degree in T (other that the root rt who has $\deg(rt) + 1$ appearances). We will treat each such appearance as a separate vertex. That is $\mathcal{L}$ is a path graph. See figure on the right for an illustration. For a vertex $v \in V$, let $\mathcal{L}(v) \subseteq \mathcal{L}$ be the set of appearances of v in $\mathcal{L}$. In Appendix A we will prove the following lemma, that computes the traversal $\mathcal{L}$ in $\tilde{O}(\sqrt{n} + D)$ rounds, meaning that each vertex v will know $\mathcal{L}(v)$ and the visiting time of every vertex $x \in \mathcal{L}(v)$.

Lemma 2 (MST traversal)**.** *Let $G = (V, E, w)$ be a weighted graph with n vertices, hop-diameter D and root $rt \in V$, then there is a deterministic algorithm in the CONGEST model that computes $\mathcal{L}$ in $\tilde{O}(\sqrt{n} + D)$ rounds.*

4 Shallow Light Tree (SLT)

In this section we present our SLT construction. Recall that an (α, β)-SLT of G with a root rt is a tree T_{SLT} that satisfies: 1) $\forall\ v \in V,\ d_{T_{\text{SLT}}}(rt, v) \leq \alpha \cdot d_G(rt, v)$, and 2) $w(T_{\text{SLT}}) \leq \beta \cdot w(MST)$. We show the following theorem.

Theorem 1 (SLT)**.** *There is a deterministic distributed algorithm in the CONGEST model, that given a weighted graph G with n vertices and hop-diameter D, root vertex rt and parameter $\varepsilon > 0$, constructs an $(1 + \varepsilon, 1 + O(\frac{1}{\varepsilon}))$-SLT in $\tilde{O}\left(\sqrt{n} + D\right) \cdot \text{poly}(\varepsilon^{-1})$ rounds.*

Initially we will assume that $\varepsilon \in (0, 1)$. Afterwards, we will show how to generalize our result to $\varepsilon \geq 1$. Intuitively, in order to construct an SLT, one should combine the MST tree T of G with a shortest path tree rooted at rt. Unfortunately, currently existing algorithms for constructing exact shortest path tree [Elk17a, GL18] require more than $\tilde{O}(\sqrt{n} + D)$ rounds. Instead, we will use an approximate shortest path tree of [BKKL17]. Specifically, they show that given a root vertex rt and a parameter $\varepsilon \in (0, 1]$, one can compute an approximate shortest path tree T_{rt} in $\tilde{O}((\sqrt{n} + D)/\text{poly}(\varepsilon))$ rounds. The approximation here is in the sense that for every vertex v,

$$d_G(rt, v) \leq d_{T_{rt}}(rt, v) \leq (1 + \varepsilon) \cdot d_G(rt, v)\ . \tag{1}$$

Our strategy to construct the SLT is similar to the framework of [ABP92, KRY95]. First, construct an MST T and an approximate shortest path T_{rt} (rooted at rt). Next, choose a subset of vertices BP called *Break Points*. An intermediate graph H will be constructed as a union of T, and

Algorithm 1 `Distributed SLT construction`

input : parameter $\varepsilon \in (0,1)$, root vertex rt
output: $(1+\varepsilon, 1+O(\frac{1}{\varepsilon}))$-SLT

construct MST T with traversal $\mathcal{L}$ according to Lemma 2
construct $1+\varepsilon$ approximate shortest path tree T_{rt} from rt using [BKKL17]
compute a break points set BP using the `Break Points Selection` procedure (Algorithm 2)
compute subgraph $H = T \cup \bigcup_{b \in \mathrm{BP}} P_b$ // *P_b being the unique path from rt to b in T_{rt}.*
construct $1+\varepsilon$ approximate shortest path tree T_{SLT} from rt w.r.t H using [BKKL17]
return T_{SLT}

the paths in T_{rt} from rt to all the vertices in BP. We will argue that H has lightness $O(\frac{1}{\varepsilon})$, and approximate distance to rt up to a $1+O(\varepsilon)$ factor. Our final SLT will be constructed as yet another approximate shortest path tree in H (rooted at rt). The pseudo-code of our algorithm appears in Algorithm 1. The main difference from previous works is that a refined selection of breakpoints is required, in order to ensure efficient implementation in the CONGEST model. In previous algorithms BP was chosen sequentially, i.e., break points were determined one after another. In contrast, we have two phases. In the first phase we choose the BP locally, while in the second phase we somewhat sparsify the set BP using a global computation.

We remark that [KRY95] gave an efficient implementation of their algorithm in the PRAM CREW model with n processors in $O(\log n)$ rounds. However, this implementation uses pointer jumping techniques which cannot be translated to the CONGEST model.

4.1 Break Points Selection

Before picking the break points, we create traversal $\mathcal{L}$ of the MST T (rooted at rt) as in Section 3, such that each vertex $v \in V$ knows its appearances $\mathcal{L}(v)$ and visiting times R_x for any $x \in \mathcal{L}(v)$. We will treat vertices with duplications according to their appearances on $\mathcal{L}$. That is, v will simulate different vertices in $\mathcal{L}$ (and even can be chosen to BP several times). Note that every neighbor u of v in G is a neighbor of exactly two vertices in $\mathcal{L}(v)$ (w.r.t $\mathcal{L}$), and therefore v indeed can act in several roles without congestion issues. In addition, each vertex $x_i \in \mathcal{L}$ will know its index i (i.e., how many vertices precede him in $\mathcal{L}$ and not only the weighted visiting time R_{x_i}). This information can be obtained by running the same algorithm that finds visiting times, ignoring the weights.

Set $\alpha = \lceil \sqrt{n} \rceil$. We will construct BP in several steps. The initial set of break points will be $\mathrm{BP}' = \{\mathrm{x}_0, \mathrm{x}_\alpha, \mathrm{x}_{2\alpha}, \mathrm{x}_{3\alpha}, \dots\}$, i.e., all the vertices whose index is a multiple of α. Next, we create a break point set BP_1 from $\mathcal{L} \setminus \mathrm{BP}'$. In each interval $I_i = \{x_{i\alpha}, x_{i\alpha+1}, \dots, x_{(i+1)\alpha-1}\}$ in parallel we will add points to BP_1. Initially, for every i, $x_{i\alpha}$ sends a message to $x_{i\alpha+1}$ with the information $(x_{i\alpha}, R_{x_{i\alpha}})$. Generally, each vertex $x_j \in I_i$ will get at some point a message (y, R_y) from x_{j-1}. The interpretation is that y is the most recent addition to BP_1, with the additional information of R_y. Now, x_j will join BP_1 if the following condition holds:

$$d_{\mathcal{L}}(x_j, y) = R_{x_j} - R_y > \varepsilon \cdot d_{T_{rt}}(rt, x_j) \ . \tag{2}$$

Note that $d_{T_{rt}}(rt, x_j)$ is information locally known to x_j from the approximate shortest path computation. Now, x_j will send a message to x_{j+1}. If x_j joined BP_1, the message will be $\left(x_j, R_{x_j}\right)$.

Algorithm 2 `Break Points Selection`

input : MST T with traversal $\mathcal{L} = \{rt = x_0, x_1, \ldots, x_{2n-2}\}$
parameter $\varepsilon \in (0,1)$, $1+\varepsilon$ approximate shortest path tree T_{rt}
output: A subset of vertices BP $\subseteq$ V

set $\alpha = \lceil \sqrt{n} \rceil$, $\text{BP}' = \{\text{x}_0, \text{x}_\alpha, \text{x}_{2\alpha}, \text{x}_{3\alpha}, \ldots\}$, $\text{BP}_1 \leftarrow \emptyset$, $\text{BP}_2 \leftarrow \emptyset$
foreach *interval* $I_i = \{x_{i\alpha}, x_{i\alpha+1}, \ldots, x_{(i+1)\alpha-1}\} \subseteq \mathcal{L}$ **simultaneously do**
    set $y \leftarrow x_{i\alpha}$
    for j *from* 1 *to* α **do**
        $x_{i\alpha+j-1}$ sends (y, R_y) to $x_{i\alpha+j}$
        if $R_{x_{i\alpha+j}} - R_y > \varepsilon \cdot d_{T_{rt}}(rt, x_{i\alpha+j})$ **then**
            add $x_{i\alpha+j}$ to BP_1, set $y \leftarrow x_{i\alpha+j}$
convergecast $\{(x_i, R_{x_i})\}_{x_i \in \text{BP}'}$ to rt
set $y \leftarrow rt$
locally in rt**, for** i *from* 1 *to* $\lceil \frac{n}{\alpha} \rceil$ **do**
    if $R_{x_{i\alpha}} - R_y > \varepsilon \cdot d_{T_{rt}}(rt, x_{i\alpha})$ **then**
        add $x_{i\alpha}$ to BP_2, set $y \leftarrow x_{i\alpha}$
broadcast BP_2 from rt
return $\text{BP} = \text{BP}_1 \cup \text{BP}_2$

Otherwise the message will be (y, R_y). After $\alpha - 1$ rounds this procedure ends, and each internal vertex $x \notin \text{BP}'$ knows whether it joins BP_1 or not.

We cannot allow all the vertices in BP' to become break points (as we have no bound on the weight of all the shortest paths from rt towards them). Filtering which vertices among BP' will actually join BP will be done in a centralized fashion. All the vertices $x_{i\alpha} \in \text{BP}'$ will broadcast to rt the message $(x_{i\alpha}, R_{x_{i\alpha}})$. By Lemma 1 this can be done in $O(\sqrt{n} + D)$ rounds (since there are at most $2\sqrt{n}$ relevant indices $0 < i \leq 2n-2$). The root rt locally creates the set $\text{BP}_2 \subseteq \text{BP}'$ as follows. Initially $rt = x_0$ joins BP_2. Next, sequentially, for $x_{i\alpha}$ where $y \in \text{BP}'$ was the last vertex to join BP_2, $x_{i\alpha}$ will join BP_2 if $d_{\mathcal{L}}(x_{i\alpha}, y) = R_{x_{i\alpha}} - R_y > \varepsilon \cdot d_{T_{rt}}(rt, x_{i\alpha})$. After this local computation, rt will broadcast to the entire graph the set BP_2 in $O(\sqrt{n} + D)$ rounds. Define the final set of breakpoints as $\text{BP} = \text{BP}_1 \cup \text{BP}_2$. (Intuitively, this step computes an SLT for the submetric of the original metric, induced by the set BP'.)

4.2 The Creation of H

For a point $b \in \text{BP}$, let P_b be the unique path from rt to b in T_{rt}. Let $H = T \cup \bigcup_{b \in \text{BP}} P_b$. Denote by A_{BP} the set of vertices whose subtree in T_{rt} contains a vertex of BP. Each vertex knows whether it belongs to BP, and we would like to ensure that every $v \in A_{\text{BP}}$ will add an edge to its parent in T_{rt}. Then adding the MST edges will conclude the construction of H. In the remaining part of this sub-section we show the computation of A_{BP}.

We start by creating a set $\mathcal{F}$ of $O(\sqrt{n})$ fragments, which are subtrees of T_{rt} of hop-diameter $O(\sqrt{n})$ (this could be done by applying the first phase of the MST algorithm of [KP98], see Section A.1 for more details). Each fragment can locally (in parallel) compute in $O(\sqrt{n})$ rounds whether it contains a break point, since it has bounded hop-diameter. Next, each fragment sends

to rt its id, whether it contains a break point, and all of its outgoing edges in T_{rt}. Note there is a total of $O(\sqrt{n})$ messages in this broadcast, so by Lemma 1 this will take $O(\sqrt{n}+D)$ rounds (in fact, all vertices will receive all these messages). Now, rt can form a virtual tree T' whose vertices are the fragments $\mathcal{F} = \{F_1, F_2, \dots\}$, and its edges connect fragments F_i, F_j if there is an edge of T_{rt} between a vertex of F_i to a vertex of F_j. Now, we can also assign roots $r_1, r_2, \dots$ (where $r_i \in F_i$ and $r_1 = rt$), so that r_i is the vertex with an edge in T_{rt} to a vertex in the parent of F_i in T'. (See Section 3 for more details and a picture of the virtual tree and its roots.)

Then, rt is able to compute locally for every root $r_i \in F_i$ whether $r_i \in A_{\text{BP}}$ (i.e. its subtree contains a break point), simply by inspecting whether F_i has a descendant in T' with a break point. Then broadcast this information on all roots in $O(\sqrt{n}+D)$ rounds. Note that now every local leaf $v \in F_i$ knows whether its subtree contains a break point. Finally, locally in parallel in $O(\sqrt{n})$ rounds, in all the fragments F_i we can compute for every vertex $v \in F_i$ whether $v \in A_{\text{BP}}$.

4.3 Stretch and Lightness Analysis

In this subsection we argue that H has the desired lightness and stretch. We name the break points $\text{BP}_1 = \{\text{b}_0, \text{b}_1, \dots\}$, $\text{BP}_2 = \{\tilde{\text{b}}_0, \tilde{\text{b}}_1, \dots\}$ according to the order of their appearance in $\mathcal{L}$. It is clear by construction that for every pair of consecutive break points $\tilde{b}_{j-1}, \tilde{b}_j \in \text{BP}_2$, $d_{\mathcal{L}}(\tilde{b}_j, \tilde{b}_{j-1}) > \varepsilon \cdot d_{T_{rt}}(rt, \tilde{b}_j)$. We claim that this property remain true also for every consecutive break points $b_{j-1}, b_j \in \text{BP}_1$. Indeed, let i such that $b_j \in I_i$. If $b_{j-1} \in I_i$ then it follows by construction. Otherwise, b_j is the first break point in I_i, and by Equation (2) it holds that $d_{\mathcal{L}}(b_j, b_{j-1}) > d_{\mathcal{L}}(b_j, x_{i\alpha}) > \varepsilon \cdot d_{T_{rt}}(rt, b_j)$.

Corollary 3. $w(H) \leq (1 + \frac{4}{\varepsilon}) \cdot w(T)$.

Proof. The graph H consists of three parts, $w(H) \leq w(T) + \sum_{b \in \text{BP}_1} w(P_b) + \sum_{\tilde{b} \in \text{BP}_2} w(P_{\tilde{b}})$. We first bound the weight of the edges added due to BP_1:

$$\sum_{j \geq 1} w(P_{b_j}) = \sum_{j \geq 1} d_{T_{rt}}(rt, b_j) < \sum_{j \geq 1} \frac{1}{\varepsilon} \cdot d_{\mathcal{L}}(b_{j-1}, b_j) \leq \frac{1}{\varepsilon} \cdot w(\mathcal{L}) = \frac{2}{\varepsilon} \cdot w(T) \ .$$

Similarly for BP_2, $\sum_{j \geq 1} w(P_{\tilde{b}_j}) \leq \frac{2}{\varepsilon} \cdot w(T)$. The corollary follows. □

Lemma 4. *For every $v \in V$, $d_H(rt, v) \leq (1 + 25\varepsilon) \cdot d_G(rt, v)$.*

Proof. Consider a vertex $v \in V$. Let x be an arbitrary vertex from $\mathcal{L}(v)$. By construction there is a point $y \in \text{BP}' \cup \text{BP}_1$ such that

$$d_{\mathcal{L}}(x, y) \leq \varepsilon \cdot d_{T_{rt}}(rt, x) \ . \tag{3}$$

Moreover, for y there is a point $y' \in \text{BP}$ such that

$$d_{\mathcal{L}}(y, y') \leq \varepsilon \cdot d_{T_{rt}}(rt, y) \ , \tag{4}$$

(it might be that $x = y$ or $y = y'$). We first bound $d_{\mathcal{L}}(x, y')$, using that $d_G \leq d_{\mathcal{L}}$ and the assumption

$\varepsilon \leq 1$.

$$\begin{aligned}
d_{\mathcal{L}}(x,y') &= d_{\mathcal{L}}(x,y) + d_{\mathcal{L}}(y,y') \\
&\overset{(4)}{\leq} d_{\mathcal{L}}(x,y) + \varepsilon \cdot d_{T_{rt}}(rt,y) \\
&\overset{(1)}{\leq} d_{\mathcal{L}}(x,y) + \varepsilon \cdot (1+\varepsilon) \cdot d_G(rt,y) \\
&\leq d_{\mathcal{L}}(x,y) + 2\varepsilon \cdot (d_G(x,y) + d_G(rt,x)) \\
&\leq (1+2\varepsilon) \cdot d_{\mathcal{L}}(x,y) + 2\varepsilon \cdot d_G(rt,x) \\
&\overset{(3)}{\leq} (1+2\varepsilon) \cdot \varepsilon \cdot d_{T_{rt}}(rt,x) + 2\varepsilon \cdot d_G(rt,x) \\
&\overset{(1)}{\leq} (1+2\varepsilon) \cdot \varepsilon \cdot (1+\varepsilon) \cdot d_G(rt,x) + 2\varepsilon \cdot d_G(rt,x) \\
&\leq 8\varepsilon \cdot d_G(rt,x) \ .
\end{aligned}$$

We conclude,

$$\begin{aligned}
d_H(rt,x) &\leq d_{T_{rt}}(rt,y') + d_{\mathcal{L}}(x,y') \\
&\leq (1+\varepsilon) \cdot \big(d_G(rt,x) + d_G(x,y')\big) + d_{\mathcal{L}}(x,y') \\
&\leq (1+\varepsilon) \cdot d_G(rt,x) + 3 \cdot d_{\mathcal{L}}(x,y') \leq (1+25\varepsilon) \cdot d_G(rt,x) \ .
\end{aligned}$$

□

4.4 Finishing the Construction and Generalization to $\varepsilon > 1$

After creating the subgraph H, we create a $(1+\varepsilon)$-shortest path tree T_{SLT} (using [BKKL17]) of H rooted at rt in $\tilde{O}(\sqrt{n}+D)/\mathrm{poly}(\varepsilon)$ rounds. The tree T_{SLT} has weight at most $w(T_{\mathrm{SLT}}) \leq w(H) \leq (1+\frac{4}{\varepsilon}) \cdot w(T)$ by Corollary 3. Moreover, by Lemma 4 for every vertex $v \in V$ it holds that

$$d_{T_{\mathrm{SLT}}}(rt,v) \leq (1+\varepsilon) \cdot d_H(rt,v) \leq (1+\varepsilon) \cdot (1+25\varepsilon) \cdot d_G(rt,v) \leq (1+51\varepsilon) \cdot d_G(rt,v) \ .$$

By rescaling ε, we conclude that for every $\varepsilon \in (0,1)$ we can construct an $(1+\varepsilon, O(\frac{1}{\varepsilon}))$-SLT. This is the right behavior (up to constant factors) when the distortion is small, as shown in [KRY95]. We would like to obtain the inverse tradeoff, when the lightness is close to 1, say $1+\gamma$ for $0 < \gamma < 1$. If we will directly apply our construction for large $1 < \varepsilon = 1/\gamma$, then it can be checked that we will get distortion $O(1/\gamma^2)$ and lightness $1+\gamma$, instead of the desired $O(1/\gamma)$ distortion (roughly speaking, this is because a breakpoint in BP' may have been removed, so in the analysis we applied a chain of two breakpoints). Fortunately, we can use a reduction due to [BFN19].

Lemma 5 ([BFN19])**.** *Let $G = (V,E)$ be a graph, $0 < \delta < 1$ a parameter and $t : \binom{V}{2} \to \mathbb{R}_+$ some function. Suppose that we have an algorithm that for any given weight function $w : E \to \mathbb{R}_+$ constructs a spanner H with lightness ℓ such that every pair $u,v \in V$ suffers distortion at most $t(u,v)$. Then for every weight function w there exists a spanner H with lightness $1+\delta\ell$ and such that every pair u,v suffers distortion at most $t(u,v)/\delta$.*

The reduction algorithm works by first changing the edge weights, and then executing the original algorithm. To compute the new weight of an edge $e \in E$, we only need to know the

parameter δ, the original weight $w(e)$ and whether e belongs the MST. Thus we can easily use this reduction in the CONGEST model as well.

We presented an algorithm that constructs a subgraph H with constant lightness (say c) and distortion 2 from rt. We will use distortion function below,

$$t(u,v) = \begin{cases} 2 & rt \in \{u,v\} \\ \infty & \text{otherwise} \end{cases} .$$

Thus, given any $0 < \gamma < 1$, we can apply Lemma 5 with the parameter $\delta = \gamma/c$, and obtain lightness $1+\gamma$ and distortion $O(1/\gamma)$. Theorem 1 now follows.

5 Distributed Light Spanner

In this section we devise an efficient distributed algorithm for light spanners in general graphs. In particular, we prove the following:

Theorem 2 (Light Spanner)**.** *There is an randomized distributed algorithm in the CONGEST model, that given a weighted graph $G = (V, E, w)$ with n vertices and hop-diameter D, and parameters $k \in \mathbb{N}$, $\varepsilon \in (0,1)$, in $\tilde{O}_\varepsilon\left(n^{\frac{1}{2}+\frac{1}{4k+2}} + D\right)$ rounds, w.h.p. returns a $(2k-1)(1+\varepsilon)$ spanner H with $O_\varepsilon(k \cdot n^{1+\frac{1}{k}})$ edges and lightness $O_\varepsilon(k \cdot n^{\frac{1}{k}})$.*

Our algorithm is similar in spirit to the algorithms of [CDNS95, ES16, ENS15]. The pseudo-code of the algorithm appears in Algorithm 3. It begins by computing a traversal $\mathcal{L} = \{rt = x_0, x_1, \ldots, x_{2n-2}\}$ of the MST T, as in Section 3. In particular, every vertex v knows the set of its appearances $\mathcal{L}(v)$, and the visiting times and indices of every $x \in \mathcal{L}(v)$. Let $L = w(\mathcal{L}) = 2w(T)$ denote the length of $\mathcal{L}$. Note that the value L is known to all the vertices (or can be broadcasted in $O(D)$ rounds).

Set $E' = \{e \in E \ : \ w(e) \leq L/n\}$, and for every $i \in \{0, 1, \ldots, \lceil \log_{1+\varepsilon} n \rceil\}$ set $E_i = \{e \in E \ : \ \frac{L}{(1+\varepsilon)^{i+1}} < w(e) \leq \frac{L}{(1+\varepsilon)^i}\}$. The algorithm constructs a different spanner for each edge set, and the final spanner will be a union of all these spanners. First, build a spanner H' for the low weight edges E'. This is done using the algorithm of Baswana and Sen [BS07]. Specifically we run [BS07] on the graph $G' = (V, E')$. In $O(k)$ rounds[9] we get a $(2k-1)$-spanner H' of G', where the expected number of edges is bounded by $O(k \cdot n^{1+1/k})$.

Next, for every $i \in \{0, 1, \ldots, \lceil \log_{1+\varepsilon} n \rceil\}$ we will define a cluster graph $\mathcal{G}_i$, on which we will simulate a spanner for unweighted graphs. For each i, we partition V into clusters $\mathcal{C}_i$. Let $\mathcal{G}_i$ be an unweighted graph with $\mathcal{C}_i$ as its vertex set, and there is an edge between two clusters A, B if there are vertices $a \in A$, $b \in B$ such that $\{a, b\} \in E_i$.

In [ES16, ENS15] the greedy spanner was applied on each $\mathcal{G}_i$. However, we cannot do so efficiently in a distributed setting. Instead, we will use the randomized algorithm of [EN19] on each $\mathcal{G}_i$. For an unweighted graph with N vertices, that algorithm provides (with constant probability) a $(2k-1)$-spanner with $O(N^{1+1/k})$ edges, computed in k rounds. Even though [EN19] gave an efficient distributed implementation, the input graph $\mathcal{G}_i$ is not the communication graph G. Our

[9] The original paper claimed $O(k^2)$ rounds, but it has been observed that their algorithm can be implemented in $O(k)$ rounds.

Algorithm 3 `Distributed spanner construction`

input : parameters $k \in \mathbb{N}$ and $\varepsilon \in (0,1)$. BFS tree τ rooted at rt.
output: $(2k-1)(1+\varepsilon)$ spanner H.

construct MST T with traversal $\mathcal{L}$ according to Lemma 2, denote $L = 2 \cdot w(T)$
set $E' = \{e \in E \ : \ w(e) \leq L/n\}$, construct a $(2k-1)$ spanner H' w.r.t. the weighted graph $G' = (V, E', w_{\restriction E'})$ using [BS07]
For $i = 0$ to $\log_{1+\varepsilon} n$, set $w_i = \frac{L}{(1+\varepsilon)^i}$, $E_i = \{e \in E \ : \ \frac{L}{(1+\varepsilon)^{i+1}} < w(e) \leq \frac{L}{(1+\varepsilon)^i}\}$
for $i = 0$ *to* $\log_{1+\varepsilon}(\varepsilon \cdot n^{\frac{k}{2k+1}})$ **do**
    initialize empty clusters $\mathcal{C}_i = \left\{C_0, C_1, \ldots, C_{\frac{L}{\varepsilon \cdot w_i}}\right\}$
    for each $v \in V$, pick arbitrary $x \in \mathcal{L}(v)$, add v to the cluster $C_{\lceil R_x/\varepsilon \cdot w_i \rceil}$
    let $\mathcal{G}_i$ be the unweighted cluster graph w.r.t. $\mathcal{C}_i$ and E_i
    construct a $2k-1$-spanner $\mathcal{H}_i$ for $\mathcal{G}_i$ by simulting [EN19] algorithm (computations are done centrally using τ)
    compute set $H_i \subseteq E_i$ by picking representators of $\mathcal{H}_i$ edges
for $i = \log_{1+\varepsilon}(\varepsilon \cdot n^{\frac{k}{2k+1}})$ *to* $\log_{1+\varepsilon}(n)$ **do**
    for each $x_j \in \mathcal{L}$ such that either $R_{x_{j-1}} < s \cdot (\varepsilon \cdot w_i) \leq R_{x_j}$ or j is a multiple of $\left\lceil \frac{\varepsilon \cdot n}{(1+\varepsilon)^i} \right\rceil$, set x_j to be a cluster center of the cluster $C(x_j)$
    for every vertex $v \in V$, pick arbitrary $x_j \in \mathcal{L}(v)$, let $j' \leq j$ be the maximal s.t. $x_{j'}$ is a cluster center, add v to the cluster $C(x_{j'})$ of $x_{j'}$
    for each center x_a set $I(x_a) = \{x_a, x_{a+1}, x_{a+2}, \ldots, x_{b-1}\}$ be the communication interval of cluster $C(x_a)$ where x_b is the next consecutive cluster center
    let $\mathcal{G}_i$ be the unweighted cluster graph w.r.t. $\mathcal{C}_i$ and E_i
    construct a $2k-1$-spanner $\mathcal{H}_i$ for $\mathcal{G}_i$ by simulting [EN19] algorithm (computations are done locally using the communication intervals)
    compute set $H_i \subseteq E_i$ by picking representators of $\mathcal{H}_i$ edges
return $H = T \cup H' \cup \bigcup_{i=0}^{\log_{1+\varepsilon} n} H_i$

main technical contribution in this section is an adaptation of that algorithm for the cluster graphs $\mathcal{G}_i$, which also requires some changes in the partition that generates these graphs.

The algorithm of [EN19] runs in k rounds. Initially, every vertex x independently samples a value $r(x)$ from some distribution. In the first round x initializes $m(x) = r(x)$, $s(x) = x$ and sends $(s(x), m(x) - 1)$ to all its neighbors. In each following round, every vertex x that received messages $\{(s(v), m(v))\}_{v \in N(x)}$ from its neighbors in the previous round, computes $u = \text{argmax}_{\text{v} \in \text{N}^+(\text{x})}\{\text{m(v)}\}$, updates $m(x) = m(u)$ and $s(x) = s(u)$, and sends $(s(x), m(x) - 1)$ to all its neighbors. After k rounds, each vertex x adds to the spanner edges: for every vertex y, add one edge to an arbitrary vertex in the set $\{v \in N(x) \ : \ m(v) \geq m(x) - 1 \wedge s(v) = y\}$, if exists (in other words, for every source y whose message reached x with value at least $m(x) - 1$, we add 1 edge to the spanner, from x to a neighbor v that sent x the message on y). A useful property of the algorithm is that the stretch is guaranteed[10] while the number of edges is bounded in expectation.

[10] For the stretch bound, the random samples $r(x)$ need to satisfy $r(x) < k$, which can be verified locally. The stretch analysis in [EN19] is conditioned on the event $\forall x \in V \ : \ r(x) < k$.

In order to implement this algorithm in $\mathcal{G}_i$, the vertices in each cluster $C \in \mathcal{C}_i$ need to compute the maximum over all the values they received from their neighbors in the previous round, and then send this value. Finally, we need to make sure that for every pair of clusters we want to connect, only one edge is added. We will distinguish between two cases, as long as the hop diameter of clusters is not too large, they can compute locally the maximum value. When the hop diameter is too large, we will ensure that there are few clusters, and all the relevant information will be broadcasted to the entire graph.

Case 1: $i \leq \log_{1+\varepsilon}(\varepsilon \cdot n^{\frac{k}{2k+1}})$**.** Set $w_i = \frac{L}{(1+\varepsilon)^i}$. We now describe the partition of V into clusters $\mathcal{C}_i$. Each cluster $C \in \mathcal{C}_i$ has a name in $\{0, 1, 2, \ldots, \frac{L}{\varepsilon \cdot w_i}\}$, and its weak diameter is at most $\varepsilon \cdot w_i$ w.r.t the MST metric (i.e. for any $u, v \in C$, $d_T(u, v) \leq \varepsilon \cdot w_i$). Let $v \in V$, and $x \in \mathcal{L}(v)$ be an arbitrary appearance. Then v will belong to the cluster $\left\lceil \frac{R_x}{\varepsilon \cdot w_i} \right\rceil$ (recall that $R_x = d_{\mathcal{L}}(rt, x)$). The weak diameter is indeed bounded, as for every $v, u \in V$ which both belong to the same cluster j, it holds that there are $x' \in \mathcal{L}(v)$, $x'' \in \mathcal{L}(u)$ such that $|R_{x'} - R_{x''}| \leq \varepsilon \cdot w_i$, hence $d_T(u, v) \leq d_{\mathcal{L}}(x', x'') \leq \varepsilon \cdot w_i$. Note that each vertex belongs to a single cluster. The number of clusters is bounded by $\left\lceil \frac{L}{\varepsilon \cdot w_i} \right\rceil + 1 = \left\lceil \frac{(1+\varepsilon)^i}{\varepsilon} \right\rceil + 1 \leq \left\lceil n^{\frac{k}{2k+1}} \right\rceil + 1$.

Before the rounds simulations, each vertex $v \in V$ sends the identity of its cluster to all its neighbors. Additionally, rt samples a value r_A for every cluster $A \in \mathcal{C}_i$, and broadcasts all these values to all the vertices in $O(|\mathcal{C}_i| + D)$ rounds, using Lemma 1. Next, we describe how to implement a single round. In the beginning of the round, each vertex knows the message $(s(A), m(A))$ that all clusters $A \in \mathcal{C}_i$ sent in the previous round. The simulation has three phases: (1) Local phase: each vertex $v \in A$, computes the maximum $m(B)$ over all neighboring clusters B. No communication required, as v knows the clusters of its neighbors and their messages. (2) Convergecast phase: we convergecast $(s(A), m(A))$ towards rt on the BFS tree τ. Each vertex v that received all messages from its children in τ for a cluster A, will only forward the one with maximum $m(A)$. Therefore each vertex will forward only $|\mathcal{C}_i|$ messages, and we can pipeline all the messages of the second phase in $O(|\mathcal{C}_i| + D)$ rounds. (3) Broadcast phase: the root rt broadcasts all the new messages $(s(A), m(A))$ for all clusters $A \in \mathcal{C}_i$ to all the graph in $O(|\mathcal{C}_i| + D)$ rounds.

After k such rounds we add edges to the spanner by a convergecast of all the spanner edges towards rt using τ. Let $\mathcal{H}_i$ be the spanner of $\mathcal{G}_i$. In [EN19] it is shown that in expectation $|\mathcal{H}_i| = O(|\mathcal{C}_i|^{1+\frac{1}{k}})$. Consider a vertex $v \in A$. For every cluster B such that $\{A, B\} \in \mathcal{H}_i$ and there is a neighbor $u \in B$ of v, v will send $((u, v), (A, B))$ towards rt. On the other hand, each vertex receiving edges from $A \times B$, will forward only a single such edge. After $O(|\mathcal{C}_i|^{1+\frac{1}{k}} + D)$ rounds the center rt knows $\mathcal{H}_i$, and for every edge $(A, B) \in \mathcal{H}_i$ it knows a representative $(a, b) \in A \times B$. In additional $O(|\mathcal{C}_i|^{1+\frac{1}{k}} + D)$ rounds rt broadcasts all these edges and H_i is created accordingly. The total number of rounds to implement each iteration of [EN19] is

$$O(|\mathcal{C}_i|^{1+\frac{1}{k}} + D) \leq O\left(\left(n^{\frac{k}{2k+1}}\right)^{\frac{k+1}{k}} + D\right) = O\left(n^{\frac{1}{2}+\frac{1}{4k+2}} + D\right) .$$

Case 2: $\log_{1+\varepsilon}(\varepsilon \cdot n^{\frac{k}{2k+1}}) < i \leq \log_{1+\varepsilon}(n)$**.** Set $w_i = \frac{L}{(1+\varepsilon)^i}$. Similarly to the previous regime, we will partition the graph into clusters $\mathcal{C}_i$ with weak diameter $\varepsilon \cdot w_i$. However, as the number of clusters will be large, computations will be done locally in the clusters. In order to make the local computations efficient, we will refine the partition into clusters such that each cluster will have

bounded (weak) hop-diameter . We start by choosing cluster centers. A vertex $x_j \in \mathcal{L}$ is a cluster center if one of the following conditions is fulfilled:

1. There is an integer s such that $R_{x_{j-1}} < s \cdot (\varepsilon \cdot w_i) \leq R_{x_j}$.
2. j is a multiple of $\left\lceil \frac{\varepsilon \cdot n}{(1+\varepsilon)^i} \right\rceil$ (that is, there is an integer q such that $j = q \cdot \left\lceil \frac{\varepsilon \cdot n}{(1+\varepsilon)^i} \right\rceil$).

Note that x_0 is a center. For every vertex $x_b \in \mathcal{L}$, consider the closest center x_a left of x_b (w.r.t $\mathcal{L}$). It holds that $R_{x_b} - R_{x_a} < \varepsilon \cdot w_i$ and $b - a < \frac{\varepsilon \cdot n}{(1+\varepsilon)^i}$. Moreover, the total number of centers is bounded by $\frac{L}{\varepsilon \cdot w_i} + \frac{n}{\frac{\varepsilon \cdot n}{(1+\varepsilon)^i}} = \frac{2 \cdot (1+\varepsilon)^i}{\varepsilon}$. In particular, each vertex can compute whether it is a center locally. For every vertex $v \in V$, pick an arbitrary $x_j \in \mathcal{L}(v)$, and let $j' \leq j$ be the largest such that $x_{j'}$ is a center. Then v joins the cluster $C(x_{j'})$ of $x_{j'}$. If x_a, x_b are two consecutive cluster centers, then $I(x_a) = \{x_a, x_{a+1}, x_{a+2}, \ldots, x_{b-1}\}$ will be the communication interval for the cluster $C(x_a)$. Note that $C(x_a) \subseteq I(x_a)$ (they need not be equal, since each vertex $u \in V$ has several possible representatives in $\mathcal{L}(u)$).

Note that the hop-diameter of $I(x_a)$ is bounded by $\frac{\varepsilon \cdot n}{(1+\varepsilon)^i} \leq n^{\frac{1}{2}+\frac{1}{4k+2}}$, and also for any $u, v \in C(x_a)$ we have $d_T(u, v) \leq \varepsilon \cdot w_i$. Each vertex $v \in V$ belongs to a single cluster. However, v might belong to many communication intervals. Nevertheless, every MST edge appears twice in $\mathcal{L}$, and therefore it belongs to at most two communication intervals.

In order to enable the partition to clusters, each cluster center x_a declares itself via $I(x_a)$. That is, it sends to the right neighbor (on $\mathcal{L}$) a message declaring itself, which is forwarded until it reaches the next center x_b. This declaration takes $\frac{\varepsilon \cdot n}{(1+\varepsilon)^i}$ rounds. At the end, each vertex chooses to which cluster it joins, becomes aware of all the communication intervals it belongs to, and sends its cluster i.d. to all its neighbors.

Now that we have defined the clustering, the simulation of each iteration of [EN19] is done in essentially the same manner as the previous case, with the communication interval taking the role of the global BFS tree τ. That is, in parallel for every cluster $C(x_a)$ we find the maximum over the $m(v)$ by convergecast in $I(x_a)$. In the last round we convergecast the spanner edges touching the cluster $C(x_a)$, so we need a bound on that number. In [EN19] it is shown that w.h.p. every vertex (cluster) adds at most $O(|\mathcal{C}_i|^{1/k} \log n) = O(n^{1/k} \log n)$ edges to the spanner. So the number of rounds required for a simulation of a single iteration is at most

$$O(n^{\frac{1}{k}} \log n + n^{\frac{1}{2}+\frac{1}{4k+2}}) = O(n^{\frac{1}{2}+\frac{1}{4k+2}}) \ ,$$

(assuming $k > 1$.) The total number of rounds (for each i in this range) is thus $O\left(k \cdot n^{\frac{1}{2}+\frac{1}{4k+2}}\right)$. This concludes the second case.

Our final spanner H will be a union of the MST T, with the spanner H' of G', and with the spanners H_i for all $0 \leq i \leq \lceil \log_{1+\varepsilon} n \rceil$. As the there are $O(\log_{1+\varepsilon} n)$ different scales, we conclude that the total construction of the spanner H of G takes $\tilde{O}(n^{\frac{1}{2}+\frac{1}{4k+2}} + D)$ rounds.

5.1 Analysis

In this section we finish the proof of Theorem 2 by analyzing the stretch, lightness, and sparsity of the spanner H.

Stretch. By the triangle inequality, it suffices to show that for every edge $\{u,v\} = e \in E$, it holds that $d_H(u,v) \leq (2k-1)(1+\varepsilon)w(e)$. In fact, we will show a bound of $(2k-1)(1+O(\varepsilon))$ on the stretch. This can be fixed later by rescaling ε. Fix $\{u,v\} = e \in E$. We can assume that $w(e) \leq L$, as otherwise we fulfill the requirement using the MST edges only. If $e \in E'$, then $d_H(u,v) \leq d_{H'}(u,v) \leq (2k-1) \cdot w(e)$. Otherwise, let $i \geq 0$ such that $e \in E_i$, that is $\frac{w_i}{(1+\varepsilon)} < w(e) \leq w_i$ for $w_i = \frac{L}{(1+\varepsilon)^i}$. Let $A_u, A_v \in \mathcal{C}_i$ be the clusters containing u, v respectively. If $A_u = A_v$, then $d_H(u,v) \leq d_T(u,v) \leq \varepsilon \cdot w_i \leq w(e)$ (assuming $\varepsilon < 1/2$, say). Otherwise, $\{A_u, A_v\}$ is an edge of G_i, and therefore there is a path $A_u = A_0, A_1, \ldots, A_t = A_v$ between A_u, A_v in $\mathcal{H}_i$ where $t \leq 2k-1$. In particular, for every $0 \leq j < t$, we added some edge $\{v_j, u_{j+1}\} \in A_j \times A_{j+1} \cap E_i$ to H_i. Let $u_0 = u$ and $v_t = v$. As the distance between every pair of vertices in any cluster is bounded by $\varepsilon \cdot w_i$ and the weight of all the edges in H_i is bounded by w_i we conclude

$$\begin{aligned} d_H(u,v) \leq d_{H_i \cup T}(u_0, v_t) &\leq d_T(u_0, v_0) + \sum_{j=0}^{t-1} \left(w(v_j, u_{j+1}) + d_T(u_{j+1}, v_{j+1}) \right) \\ &\leq (t+1) \cdot \varepsilon \cdot w_i + t \cdot w_i \\ &\leq (2k-1) \cdot (1 + O(\varepsilon)) \cdot w(e) \ . \end{aligned}$$

Lightness. We bound the lightness of H' and each of the spanners H_i. First consider H'. Since the weight of every edge $e \in E'$ is at most L/n, we have

$$w(H') \leq |H'| \cdot \frac{L}{n} = O(k \cdot n^{1+\frac{1}{k}} \cdot \frac{L}{n}) = O(k \cdot n^{\frac{1}{k}} \cdot L) \ .$$

Next consider H_i, which has expected $O(|\mathcal{C}_i|^{1+\frac{1}{k}}) = O\left(\left(\frac{(1+\varepsilon)^i}{\varepsilon} \right)^{1+\frac{1}{k}} \right)$ edges, all of weight bounded by $w_i = \frac{L}{(1+\varepsilon)^i}$. So the expected weight of all these H_i together is

$$\begin{aligned} \sum_{i=0}^{\lceil \log_{1+\varepsilon} n \rceil} \mathbb{E}\left[w(H_i) \right] \leq \sum_{i=0}^{\lceil \log_{1+\varepsilon} n \rceil} \mathbb{E}[|H_i|] \cdot w_i &= \sum_{i=0}^{\lceil \log_{1+\varepsilon} n \rceil} O\left(\left(\frac{(1+\varepsilon)^i}{\varepsilon} \right)^{1+\frac{1}{k}} \right) \cdot \frac{L}{(1+\varepsilon)^i} \\ &= O\left(\frac{L}{\varepsilon^{1+1/k}} \right) \cdot \sum_{i=0}^{\lceil \log_{1+\varepsilon} n \rceil} (1+\varepsilon)^{\frac{i}{k}} = O\left(\frac{L}{\varepsilon^{1+1/k}} \right) \cdot \frac{(1+\varepsilon)^{\frac{\lceil \log_{1+\varepsilon} n \rceil + 1}{k}} - 1}{(1+\varepsilon)^{1/k} - 1} \\ &= O\left(\frac{L \cdot k \cdot n^{1/k}}{\varepsilon^{2+1/k}} \right) \ , \end{aligned}$$

where the last equality follows as $(1+\varepsilon)^{\frac{1}{k}} - 1 \geq e^{\frac{\varepsilon}{2} \cdot \frac{1}{k}} - 1 \geq \frac{\varepsilon}{2k}$. We conclude that the expected weight of H is

$$\mathbb{E}[w(H)] \leq w(T) + w(H') + \sum_{i=0}^{\lceil \log_{1+\varepsilon} n \rceil} \mathbb{E}[w(H_i)] = O_\varepsilon\left(k \cdot n^{\frac{1}{k}} \cdot L \right) \ .$$

Sparsity. Following the analysis of the lightness, we have

$$\sum_{i=0}^{\lceil \log_{1+\varepsilon} n \rceil} \mathbb{E}[|H_i|] \leq \sum_{i=0}^{\lceil \log_{1+\varepsilon} n \rceil} O\left(\left(\frac{(1+\varepsilon)^i}{\varepsilon}\right)^{1+\frac{1}{k}}\right) = O\left(\frac{1}{\varepsilon^{1+\frac{1}{k}}} \cdot \frac{n^{1+\frac{1}{k}}}{(1+\varepsilon)^{1+\frac{1}{k}}-1}\right) = O\left(\frac{n^{1+\frac{1}{k}}}{\varepsilon^{2+\frac{1}{k}}}\right) ,$$

We conclude,

$$\mathbb{E}[|H|] \leq |T| + |H'| + \sum_{i=0}^{\lceil \log_{1+\varepsilon} n \rceil} \mathbb{E}[|H_i|] = O_\varepsilon\left(k \cdot n^{1+\frac{1}{k}}\right) .$$

Remark 1. *We note that the number of edges mostly comes from the spanner H'. We can in fact use the techniques developed here in order to efficiently implement the algorithm of [EN19] for weighted graphs in the CONGEST model, which provides a $(2k-1)\cdot(1+\varepsilon)$-spanner with $O_\varepsilon(\log k \cdot n^{1+1/k})$ edges. That algorithm partitions the edges E to $\approx \log k$ sets, and for each set, applies the unweighted version on a cluster graph. Since we already have an efficient distributed implementation of that unweighted algorithm, we conclude that our sparsity bound may be improved to $O_\varepsilon(\log k \cdot n^{1+1/k})$. We leave the details to the full version.*

Successes Probability. Note that once the computation concludes, we can easily compute the size and lightness of the spanner in $O(D)$ rounds via the BFS tree τ. Thus we can repeat the computation for H' and each H_i until they meet the required bounds, which will happen w.h.p. after at most $O(\log n)$ tries. Recall that the stretch bound is guaranteed to hold.

6 Distributed Construction of Nets

In this section we devise an efficient distributed algorithm for computing nets in general graphs. Let $G=(V,E,w)$ be a weighted graph. Recall that for $\alpha > 0$, a set $N \subseteq V$ is *α-covering* if for every vertex $x \in V$ there is $y \in N$ with $d_G(x,y) \leq \alpha$. A set $N \subseteq V$ is *β-separated* if for every $x, y \in N$, $d_G(x,y) > \beta$. We say that a set N is an *(α,β)-net* if it is both α-covering and β-separated. (In the literature nets are often defined with $\alpha = \beta$, but we will need the more general definition, since we will only be able to provide nets with $\alpha > \beta$.)

Our construction of nets is inspired by the MIS (maximal independent set) algorithm of [MRSZ11] (which itself is inspired by [Lub86]). The MIS algorithm works in $O(\log n)$ rounds, where in each round a permutation is sampled. A vertex joins the MIS iff it is local minimum (i.e., it appears before all its neighbors in the permutation). We will also sample a permutation. However, instead of checking only the neighbors, a vertex v will join the net iff it is a local minimum in a geometric sense. I.e., v appears before all the vertices of $B_G(v,\beta)$ in the permutation.

In order to implement this algorithm efficiently we will require several tools that have found distributed constructions recently, such as *Least-Element lists* and *shortest path trees*. However, we do not know how to compute these exactly in the allotted number of rounds, so we will settle for approximations. The rest of the section is dedicated to proving the following theorem.

Theorem 3. *Given a weighted graph $G=(V,E,w)$ with hop-diameter D and parameters $\Delta > 0$, $\delta \in (0,1)$, there is a randomized algorithm in the CONGEST model that computes w.h.p. a $\left((1+\delta)\cdot\Delta, \frac{\Delta}{1+\delta}\right)$-net in $(\sqrt{n}+D)\cdot 2^{\tilde{O}(\sqrt{\log n}\cdot\log(1/\delta))}$ rounds.*

Least Element Lists. LE lists were introduced by [Coh97]:

Definition 1. *Given a weighted graph $G = (V, E, w)$, a set $A \subseteq V$ of vertices, and a permutation $\pi : A \to [|A|]$ on A, the LE list of a vertex $v \in A$ is defined as*

$$LE_{G,A,\pi}(v) = \{(u, d_G(u, v)) \ : \ u \in A, \nexists w \in A \text{ s.t. } d_G(v, w) \leq d_G(v, u) \text{ and } \pi(w) < \pi(u)\} \ .$$

In words, a vertex $u \in A$ joins $LE_{G,A,\pi}(v)$, the LE list of v, if u is first in the permutation among all the vertices at distance at most $d_G(v, u)$ from v (alternatively, u is the closest vertex to v among the first $\pi(u)$ vertices in the permutation.).

Khan et. al. [KKM+12] showed that with high probability over the choice of the permutation π, it holds that $|\mathrm{LE}_{G,A,\pi}(v)| = O(\log |A|)$ simultaneously for all the vertices $v \in A$. Using hopsets, Friedrichs and Lenzen [FL16] were able to efficiently compute LE lists in the CONGEST model (improving upon Ghaffari and Lenzen [GL14]) for a graph H that is a good approximation of G. (We remark that their algorithm was given in the case $A = V$, but it is a simple adaptation to adjust it to the more general case.)

Theorem 4 ([FL16])**.** *Consider a graph $G = (V, E)$ with n vertices and hop-diameter D, a set $A \subseteq V$, and let $\delta \in (0, 1)$ be any parameter. There is a randomized algorithm in the CONGEST model that uniformly samples a permutation π and computes $\{LE_{H,A,\pi}(v)\}_{v \in V}$ for a graph H such that $d_G(u, v) \leq d_H(u, v) \leq (1 + \delta) \cdot d_G(u, v)$. The algorithm is successful w.h.p., and the number of rounds is $(\sqrt{n} + D) \cdot 2^{\tilde{O}(\sqrt{\log n} \cdot \log(1/\delta))}$.*

Algorithm. Here we describe the algorithm promised in Theorem 3. The pseudo-code of the algorithm appears in Algorithm 4. The algorithm will run in $O(\log n)$ iterations. Initially, set $A_1 = V$ the set of active vertices, and $N = \emptyset$. We denote by A_i the set of active vertices for the i'th iteration, and by N_i the net just after the i'th iteration (so $N_0 = \emptyset$). In the i'th iteration, apply Theorem 4 on the graph G with the set A_i and the parameter δ. We obtain a (uniformly random) permutation π_i on A_i, alongside with LE lists for all $v \in A_i$ w.r.t π_i and the graph H_i (which is a $1 + \delta$ approximation of G). Every $v \in A_i$ will join N_i iff it is the first in the permutation order among its Δ-neighborhood (w.r.t H_i). In other words, v joins N_i if there is no $u \in \mathrm{LE}_{H_i,A_i,\pi_i}(v)$, $u \neq v$, such that $d_{H_i}(u, v) \leq \Delta$.

Next, we will construct an $1+\delta$ approximate shortest path tree T_i (using [BKKL17], say) rooted in N_i. Remove from A_i every vertex at distance at most $(1 + \delta) \cdot \Delta$ from N_i (in T_i), to form A_{i+1}. This concludes a single iteration. Continue until iteration i where $A_i = \emptyset$.

Running time. The number of rounds for computing the LE lists is $(\sqrt{n}+D) \cdot 2^{\tilde{O}(\sqrt{\log n} \cdot \log(1/\delta))}$, and the shortest path tree takes $\tilde{O}((\sqrt{n} + D)/\mathrm{poly}(\delta))$ rounds. (Deciding whether to join N_i is done locally once the LE lists are computed.) We will soon show that w.h.p. there are $O(\log n)$ iterations, thus the total number of rounds is $(\sqrt{n} + D) \cdot 2^{\tilde{O}(\sqrt{\log n} \cdot \log(1/\delta))}$.

Analysis. First we argue that once the algorithm concludes, N is a $\left((1 + \delta) \cdot \Delta, \frac{\Delta}{1+\delta}\right)$-net. To see the packing property, consider $u, v \in N$, and we want to show that $d_G(u, v) > \frac{\Delta}{1+\delta}$. If u and v joined N in the same iteration i, then they both were maximal in their Δ neighborhoods w.r.t H_i,

Algorithm 4 `Distributed construction of nets`

input : Parameters $\Delta > 0$ and $\delta \in (1, 0)$.

output: $\left((1+\delta)\cdot\Delta, \frac{\Delta}{1+\delta}\right)$-net.

set $A \leftarrow V$, $N \leftarrow \emptyset$
while $A \neq \emptyset$ **do**
  sample pemutation π of the vertex set A using Theorem 4 with parameter δ, as a result each vertex v knows $\mathrm{LE}_{H,A,\pi}(v)$
  simultaneously foreach $v \in A$ **do**
    if *v is the first in π among its Δ-neigborhood (w.r.t. H)* **then**
      add v to N
  construct $1+\delta$ approximate shortest path tree T rooted in N using [BKKL17]
  simultaneously foreach $v \in A$ **do**
    if $d_T(v, N) \leq (1+\delta)\cdot\Delta$ **then**
      remove v from A
return N

and therefore $d_G(u,v) \geq \frac{d_{H_i}(u,v)}{1+\delta} > \frac{\Delta}{1+\delta}$. Otherwise, assume w.l.o.g that u joined N at iteration i, while v joined N at iteration $i' > i$. As v remained active, necessarily $(1+\delta)\cdot\Delta < d_{T_i}(v, N_i) \leq (1+\delta)\cdot d_G(u,v)$. For the covering property, let v be some vertex that becomes inactive at iteration i. Then there is a vertex $u \in N$ such that $d_G(u,v) \leq d_{T_i}(u,v) \leq (1+\delta)\cdot\Delta$.

It remains to show that after $O(\log n)$ iterations w.h.p. no active vertices remain. We say that a pair of vertices $\{u, v\}$ at distance at most Δ (w.r.t. d_G) is *active* if both u, v are active. Set $\mathcal{E}_0 = \{\{u,v\} \mid d_G(u,v) \leq \Delta\}$ to be the set of initially active pairs. $\mathcal{E}_i$ will denote the set of pairs at distance at most Δ that remain active after the i'th iteration. For an active vertex $v \in A_i$, denote by $A_i(v) = A_i \cap B_G(v, \Delta)$. Note that $u \in A_i(v) \iff \{v,u\} \in \mathcal{E}_{i-1}$. We say that $w \in A_i(v)$ *kills* v at the i'th iteration if w is first in the permutation among $A_i(v) \cup A_i(w)$. Suppose that w kills v in the i'th iteration. Then we claim that w joins N_i, as for every vertex $w' \in A_i$ such that $d_{H_i}(w,w') \leq \Delta$ it holds that $d_G(w,w') \leq \Delta$ as well, and therefore $\pi(w) < \pi(w')$. In particular, v will cease to be active as $d_{T_i}(v, N_i) \leq (1+\delta)\cdot d_G(v,w) \leq (1+\delta)\cdot\Delta$. Note that at most one vertex w can kill v (as it must be the first in the permutation in $A_i(v)$). [11] Therefore the probability that v becomes inactive in iteration i is at least $\sum_{w\in A_i} \Pr[v \text{ is killed by } w]$. As we pick the permutation π uniformly at random, the probability that w kills v is exactly $|A_i(v) \cup A_i(w)|^{-1}$.

Our next goal is to show that in expectation, the number of active pairs is halved in each iteration. Consider a pair $\{u,v\} \in \mathcal{E}_{i-1}$. Then

$$\begin{aligned}\Pr[\{u,v\} \notin \mathcal{E}_i] &\geq \frac{1}{2}\cdot(\Pr[u \text{ became inactive}] + \Pr[v \text{ became inactive}]) \\ &\geq \frac{1}{2}\cdot\sum_{w\in A_i}\Pr([w \text{ kills } u] + \Pr[w \text{ kills } v])\ .\end{aligned}$$

[11] Note also that it is possible for a vertex to cease to be active without being killed. This can happen if the first vertex in the permutation among $A_i(v)$ does not join N_i, while a different vertex in $A_i(v)$ does join. We do not take advantage of this possibility.

$\mathcal{E}_{i-1} \setminus \mathcal{E}_i$ is the set of pairs who cease to be active in the i'th iteration. We conclude

$$\begin{aligned}
\mathbb{E}\left[|\mathcal{E}_{i-1} \setminus \mathcal{E}_i|\right] &= \sum_{\{u,v\}\in\mathcal{E}_{i-1}} \Pr\left[\{u,v\} \notin \mathcal{E}_i\right] \\
&\geq \frac{1}{2} \cdot \sum_{\{u,v\}\in\mathcal{E}_{i-1}} \sum_{w\in A_i} \Pr\left([w \text{ kills } u] + \Pr\left[w \text{ kills } v\right]\right) \\
&= \frac{1}{2} \sum_{u\in A_i} \sum_{w\in A_i(u)} \Pr\left[w \text{ kills } u\right] \cdot |A_i(u)| \\
&= \frac{1}{2} \sum_{u\in A_i} \sum_{w\in A_i(u)} \frac{|A_i(u)|}{|A_i(u) \cup A_i(w)|} \\
&\geq \frac{1}{2} \sum_{u\in A_i} \sum_{w\in A_i(u)} \frac{|A_i(u)|}{|A_i(u)| + |A_i(w)|} \\
&= \frac{1}{2} \sum_{\{u,w\}\in\mathcal{E}_{i-1}} \frac{|A_i(u)| + |A_i(w)|}{|A_i(u)| + |A_i(w)|} = \frac{1}{2} \cdot |\mathcal{E}_{i-1}| \ .
\end{aligned}$$

Set $p = \Pr\left[|\mathcal{E}_{i-1} \setminus \mathcal{E}_i| > \frac{1}{4} \cdot |\mathcal{E}_{i-1}|\right]$. Then $\frac{1}{2} \cdot |\mathcal{E}_{i-1}| \leq \mathbb{E}\left[|\mathcal{E}_{i-1} \setminus \mathcal{E}_i|\right] \leq p \cdot |\mathcal{E}_{i-1}| + (1-p) \cdot \frac{1}{4} \cdot |\mathcal{E}_{i-1}|$, which implies $p \geq \frac{1}{3}$. After $O(\log n)$ iterations, by Chernoff inequality we had w.h.p. at least $\log_{\frac{4}{3}} n$ iterations where $|\mathcal{E}_{i-1} \setminus \mathcal{E}_i| > \frac{1}{4} \cdot |\mathcal{E}_{i-1}|$, and therefore $\mathcal{E}_{O(\log n)} = \emptyset$. Note that if no active edges remain, then each active vertex is the only active vertex in its entire Δ neighborhood. In particular, in the next iteration all the active vertices will join N and cease to be active, as required.

7 Light Spanner for Doubling Metrics

In this section we show a distributed algorithm that produces light spanners for doubling metrics.

Theorem 5 (Light Spanner for Doubling Graphs)**.** *There is an randomized distributed algorithm in the CONGEST model, that given a weighted graph $G = (V, E, w)$ with n vertices, hop-diameter D, doubling dimension* ddim*, and parameter $\varepsilon \in (0, 1)$, in $(\sqrt{n}+D) \cdot \varepsilon^{-\tilde{O}(\sqrt{\log n}+\text{ddim})}$ rounds, w.h.p. returns a $(1+\varepsilon)$-spanner H with $O\left(n \cdot \varepsilon^{-O(\text{ddim})} \cdot \log n\right)$ edges and lightness $\varepsilon^{-O(\text{ddim})} \cdot \log n$.*

Our spanner construction will go as follows. We take an $\varepsilon\Delta$-net for every distance scale Δ, and connect net points that are within distance Δ of each other. An efficient construction of nets was described in Section 6. To efficiently implement the net point connections, we use approximate shortest path computations based on *hopsets* from every net point. This will ensure that every such approximate shortest path has few edges, thus the running time can be controlled.

The following lemma gives the standard packing property of doubling metrics (see, e.g., [GKL03]).

Lemma 6. *Let (M, ρ) be a metric space with doubling dimension* ddim*. If $S \subseteq M$ is a subset of points with minimum interpoint distance r that is contained in a ball of radius R, then $|S| \leq \left(\frac{2R}{r}\right)^{O(\text{ddim})}$.*

Algorithm 5 `Distributed construction of spanner for doubling metric`

input : parameters $\Delta > 0$ and $\delta \in (1, 0)$.

output: $\left((1+\delta)\cdot\Delta, \frac{\Delta}{1+\delta}\right)$-net.

set $A \leftarrow V$, $N \leftarrow \emptyset$, $H \leftarrow \emptyset$, $\beta = 1/\varepsilon^{\tilde{O}(\sqrt{\log n})}$
compute an MST tree T, set $L = w(T)$
construct a (β, ε) hopset F using [EN16]
for $i = 1$ *to* $\log_{1+\varepsilon} L$ **do**
    set $\Delta = (1+\varepsilon)^i$
    compute an $(2\varepsilon\cdot\Delta, \frac{\varepsilon}{2}\cdot\Delta)$-net N_i using Algorithm 4
    simultaneously foreach *net point* $v \in N_i$ **do**
        construct a shortest path tree T_v in $G \cup F$ rooted in v, grow the tree only up to radius 2Δ and using at most β hops
        add to H all the edges in T_v on paths from v to other net points in N_i
return H

7.1 Spanner Construction

The pseudo-code of our algorithm appears in Algorithm 5. Let L be the weight of the MST of G. For every $\Delta \in \{1, 1+\varepsilon, (1+\varepsilon)^2, \ldots, (1+\varepsilon)^{\log_{1+\varepsilon} L}\}$ we compute a $(2\varepsilon\cdot\Delta, \frac{\varepsilon}{2}\cdot\Delta)$-net as in Theorem 3 (e.g., we can take $\delta = 1/2$). Let N_i be the net for $\Delta = (1+\varepsilon)^i$, and for every net point $u \in N_i$, run in parallel a 2Δ-bounded $(1+\varepsilon)$-approximate shortest path tree rooted at u, and add to the spanner H a $(1+\varepsilon)$-approximate shortest path from u to all other net points $v \in N_i$ found within this distance bound.

Shortest paths via hopsets. To implement these bounded shortest paths computations, we use the algorithm of [EN16] based on hopsets. A (β, ε)-hopset F for a graph $G' = (V', E')$, is a set of edges F that do not reduce distances, and for every $u, v \in V'$

$$d^{(\beta)}_{G' \cup F}(u, v) \leq (1+\varepsilon)\cdot d_{G'}(u, v) \ ,$$

where $d^{(\beta)}(\cdot,\cdot)$ is the length of the shortest path containing at most β edges. The graph G' is created by choosing the set $V' \subseteq V$ of size $\approx \sqrt{n}\ln n$ at random, so that w.h.p. it intersects every shortest path in G of length at least $\sqrt{n}$. The edges E' are the $\sqrt{n}$-bounded distances in G between the vertices of V'. Fix $\beta = 1/\varepsilon^{\tilde{O}(\sqrt{\log n})}$. In [EN16] it is shown how to compute a (β, ε) hopset for G' of size $O(\sqrt{n}\cdot\beta^2)$ in $O((\sqrt{n}+D)\cdot\beta^2)$ rounds. Furthermore, that hopset is *path reporting*, that is, there is a path P_e in G for every hopset edge $e \in F$, such that the length of P_e is exactly $w(e)$, and every vertex in P_e knows it lies on a path implementing e.

Once a hopset F is computed for G', computing $(1+\varepsilon)$-approximate shortest paths in G from a root $v \in V$ amounts to running β iterations of Bellman-Ford in $G \cup E' \cup F$ In order to perform a single Bellman-Ford iteration, we first send messages over the edges of E for $2\sqrt{n}$ rounds (to reach a vertex of V', and then discover the relevant edges of E'). Second, every $u \in V'$ broadcasts its distance estimate to the root $d(v)$ to all the graph using the BFS tree τ of depth D (to implement the hopset edges). Since there are only $O(\sqrt{n}\cdot\ln n)$ vertices in V', this can be done in $O(\sqrt{n}\cdot\ln n + D)$ rounds via Lemma 1.

In our setting we would like to compute in parallel multiple source 2Δ-bounded approximate shortest paths. To this end, we will use the fact the shortest path metric is doubling, hence every vertex will participate in a small number of such computations.

7.2 Analysis

Running Time. Fix some $\Delta = (1+\varepsilon)^i$. Computing the $(2\varepsilon \cdot \Delta, \frac{\varepsilon}{2} \cdot \Delta)$-net N_i takes $(\sqrt{n}+D) \cdot 2^{\tilde{O}(\sqrt{\log n})}$ rounds. We next analyze the running time of the shortest paths computation from all net points in N_i, which is conducted in parallel. By Lemma 6 we have that the number of net points N_i in any ball of radius 2Δ is $\varepsilon^{-O(\mathrm{ddim})}$. This suggests that for any $v \in V$ and any hopset edge $(x,y) \in F$, there are at most $\varepsilon^{-O(\mathrm{ddim})}$ sources of shortest paths computations that will reach them. Thus the number of rounds required to implement in parallel all approximate shortest path computations is $O\left((\sqrt{n}+D) \cdot \beta \cdot \varepsilon^{-O(\mathrm{ddim})}\right) = (\sqrt{n}+D) \cdot \varepsilon^{-\tilde{O}(\sqrt{\log n}+\mathrm{ddim})}$. This is proportional to the time required to compute the hopset, and as there are $O(\log L) = O(\log n)$ different scales, the final running time is $(\sqrt{n}+D) \cdot \varepsilon^{-\tilde{O}(\sqrt{\log n}+\mathrm{ddim})}$.

Stretch Bound. For simplicity, we will prove stretch $1+c\cdot\varepsilon$ for some constant c to be determined later. One can get stretch $1+\varepsilon$ by rescaling ε. Consider a pair $u, v \in V$ such that $(1+\varepsilon)^{i-1} < d_G(u,v) \le (1+\varepsilon)^i = \Delta$. Assume by induction that every pair u', v' at distance at most $(1+\varepsilon)^{i-1}$ already enjoys stretch at most $1+c\cdot\varepsilon$ in H. The base case $i=0$ is trivial since there are no pairs $u' \neq v'$ of distance less than 1. Let $\tilde{u}, \tilde{v} \in N_i$ be net points such that $d_G(u,\tilde{u}), d_G(v,\tilde{v}) \le 2\varepsilon\cdot\Delta$. By the triangle inequality $d_G(\tilde{u},\tilde{v}) \le (1+4\varepsilon)\cdot\Delta$, and since $\varepsilon < 1/8$ we have that $(1+\varepsilon)\cdot(1+4\varepsilon)\cdot\Delta \le 2\Delta$, so the 2Δ-bounded shortest path exploration from $\tilde{u}$ must have discovered $\tilde{v}$. Therefore we added a $1+\varepsilon$ approximate shortest path between $\tilde{u}$ and $\tilde{v}$ to H. Using the induction hypothesis on the pairs $\{u,\tilde{u}\}$ and $\{v,\tilde{v}\}$, we conclude

$$\begin{aligned} d_H(v,u) &\le d_H(v,\tilde{v}) + d_H(\tilde{v},\tilde{u}) + d_H(u,\tilde{u}) \\ &\le (1+c\varepsilon)\cdot 2\varepsilon\Delta + (1+\varepsilon)(1+4\varepsilon)\Delta + (1+c\varepsilon)\cdot 2\varepsilon\Delta \\ &\overset{(*)}{<} \frac{1+c\varepsilon}{1+\varepsilon}\cdot\Delta \le (1+c\varepsilon)\cdot d_G(u,v) \ , \end{aligned}$$

where the inequality $(*)$ follows for any constant $c \ge 30$, using that $\varepsilon < 1/8$.

Lightness bound. Let $n_i = |N_i|$. In [FN18] the following claim was shown.

Claim 7 ([FN18])**.** *Consider a weighted graph G with MST of weight L, such that there is an r-separated set N. Then $|N| \le \left\lceil \frac{2L}{r} \right\rceil$.*

It follows that $n_i = O(\frac{L}{\varepsilon\Delta})$. Recall that Lemma 6 implies that the number of net points N_i in a ball of radius 2Δ is $\varepsilon^{-O(\mathrm{ddim})}$. So for every net point $u \in N_i$ we added at most $\varepsilon^{-O(\mathrm{ddim})}$ paths of weight at most 2Δ each. Thus the total weight of edges added for the i'th scale is bounded by $n_i \cdot \varepsilon^{-O(\mathrm{ddim})} \cdot 2\Delta = \varepsilon^{-O(\mathrm{ddim})} \cdot L$. In particular, the sum of weights of the edges added in the $2\log_{1+\varepsilon} n$ scales $i \in \{\log_{1+\varepsilon}\frac{L}{n^2}, \ldots, \log_{1+\varepsilon} L\}$ is bounded by $\varepsilon^{-O(\mathrm{ddim})} \cdot \log n \cdot L$. The contribution of all the other scales is negligible as all these scales adds at most n^2 edges of weight less than $\frac{L}{n^2}$. The bound on the lightness follows.

Sparsity bound. Consider a vertex $v \in V$ and the set of edges added for scale $\Delta = (1+\varepsilon)^i$. Consider the ball B of radius 2Δ around v, recall that $|B \cap N_i| = \varepsilon^{-O(\mathrm{ddim})}$. We added a path to the spanner only between net points of distance at most 2Δ. Therefore v can participate in such paths only when both net points are in B (as otherwise the length of a path going through v will be greater than 2Δ). Therefore v might participate in at most $(\varepsilon^{-O(\mathrm{ddim})})^2 = \varepsilon^{-O(\mathrm{ddim})}$ such paths. As we added at most 2 edges touching v per path, the number of edges added which are incident on v is bounded by $\varepsilon^{-O(\mathrm{ddim})}$. As there are $\log_{1+\varepsilon} L$ scales (and n vertices), we can bound the total number of edges added to the spanner by $n \cdot \varepsilon^{-O(\mathrm{ddim})} \cdot \log n$ (recall we assumed G has diameter $\mathrm{poly}(n)$, thus also $L = \mathrm{poly}(n)$).

8 Lower Bounds

In a seminal paper, Das Sarma et al. [SHK+12] showed that approximating the weight of an MST up to polynomial factors takes $\tilde{\Omega}(\sqrt{n})$ rounds. As both an SLT and a light spanner provide such an approximation to the weight of the MST, we conclude:

Theorem 6. *Every distributed algorithm in the CONGEST model that computes an SLT or a spanner with polynomial lightness, takes $\tilde{\Omega}(\sqrt{n}+D)$ rounds.*

Next we argue that computing nets takes $\tilde{\Omega}(\sqrt{n}+D)$ rounds as well. Our lower bound is for general graphs. Therefore, it is possible that computing nets for graphs with bounded doubling dimension can be performed faster. Nevertheless, we conclude that Theorem 3 is tight (up to lower order terms).

Theorem 7. *Suppose that there is a distributed algorithm in the CONGEST model that for every parameter Δ computes an $(\alpha \cdot \Delta, \Delta)$-net for some $1 \le \alpha \le \mathrm{poly}(n)$. Then the algorithm runs in $\tilde{\Omega}(\sqrt{n}+D)$ rounds.*

Proof. Let G be a graph from the family [SHK+12] used for their lower bound for approximating the weight of the MST. The only property we will use is that G has polynomial diameter Λ. (W.l.o.g. the minimal distance is 1, and Λ is the largest distance in G.) We will create nets in all the distance scales, and use their cardinality in order to provide a polynomial approximation to the weight of the MST. As there are only $O(\log n)$ distance scales, and the cardinality of all the nets can be computed in $O(D + \log n)$ time, the lower will follow.

For every $i \ge -\lceil \log \alpha \rceil$, compute an $(\alpha \cdot 2^i, 2^i)$-net N_i. We stop in the first time that $|N_i| = 1$. Note that we compute at most $O(\log n)$ nets, since $\log \alpha = O(\log n)$, and when 2^i is larger than $\Lambda = \mathrm{poly}(n)$, there can be only a single net point. Next, we compute the cardinality of all the net points and return $\Psi = \sum_i n_i \cdot \alpha \cdot 2^{i+1}$ where $n_i = |N_i|$. To finish the proof, we will argue that $L \le \Psi \le O(\alpha \cdot \log n) \cdot L$ (where L is the weight of the MST).

For every i, as N_i is 2^i-separated, by Claim 7 $n_i \le \left\lceil \frac{2L}{2^i} \right\rceil$. In particular, $\Psi = \sum_i n_i \cdot \alpha \cdot 2^{i+1} \le \alpha \cdot \sum_i \left\lceil \frac{2L}{2^i} \right\rceil \cdot 2^{i+1} = O(\alpha \cdot \log n) \cdot L$.

For the second inequality, we will construct a connected subgraph H of G. For every net point $x \in N_i$, let $y \in N_{i+1}$ be the closest point to x among points in N_{i+1}. Then by the covering property $d_G(x,y) \le \alpha \cdot 2^{i+1}$. Add to H the shortest path from x to y. Do this for all the nets. Note that $N_{-\lceil \log \alpha \rceil} = V$ (since any point can cover only itself), and that the maximal net consist of a single point. Therefore H is connected. We conclude $L \le w(H) \le \sum_i n_i \cdot \alpha \cdot 2^{i+1} = \Psi$. □

References

[AB16] A. Abboud and G. Bodwin. The 4/3 additive spanner exponent is tight. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 351–361, 2016, doi:10.1145/2897518.2897555. 1

[ABP90] B. Awerbuch, A. E. Baratz, and D. Peleg. Cost-sensitive analysis of communication protocols. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, August 22-24, 1990*, pages 177–187, 1990, doi:10.1145/93385.93417. 1

[ABP92] B. Awerbuch, A. E. Baratz, and D. Peleg. Efficient broadcast and light-weight spanners. Technical Report CS92-22, The Weizmann Institute of Science, Rehovot, Israel., 1992. 1, 4, 7

[ACIM99] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM J. Comput.*, 28(4):1167–1181, 1999, doi:10.1137/S0097539796303421. 1

[ADD+93] I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993, doi:10.1007/BF02189308. 1, 3

[ADF+17] S. Alstrup, S. Dahlgaard, A. Filtser, M. Stöckel, and C. Wulff-Nilsen. Constructing light spanners deterministically in near-linear time. *CoRR*, abs/1709.01960, 2017, arXiv:1709.01960. 1

[AGLP89] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin. Network decomposition and locality in distributed computation. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 364–369, 1989, doi:10.1109/SFCS.1989.63504. 3

[Ass83] P. Assouad. Plongements lipschitziens dans $\mathbb{R}^n$. *Bull. Soc. Math. France*, 111(4):429–448, 1983. http://eudml.org/doc/87452. 2

[Awe85] B. Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, October 1985, doi:10.1145/4221.4227. 1

[BDS04] Y. Ben-Shimol, A. Dvir, and M. Segal. SPLAST: a novel approach for multicasting in mobile wireless ad hoc networks. In *Proceedings of the IEEE 15th International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2004, 5-8 September 2004, Barcelona, Spain*, pages 1011–1015, 2004, doi:10.1109/PIMRC.2004.1373851. 2

[BEPS12] L. Barenboim, M. Elkin, S. Pettie, and J. Schneider. The locality of distributed symmetry breaking. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 321–330, 2012, doi:10.1109/FOCS.2012.60. 3

[BFN19] Y. Bartal, A. Filtser, and O. Neiman. On notions of distortion and an almost minimum spanning tree with constant average distortion. *Journal of Computer and System Sciences*, 2019. Preliminary version appeared in SODA16, doi:https://doi.org/10.1016/j.jcss.2019.04.006. 1, 11

[BKKL17] R. Becker, A. Karrenbauer, S. Krinninger, and C. Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 7:1–7:16, 2017, doi:10.4230/LIPIcs.DISC.2017.7. 4, 7, 8, 11, 18, 19

[BLW17] G. Borradaile, H. Le, and C. Wulff-Nilsen. Minor-free graphs have light spanners. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 767–778, 2017, doi:10.1109/FOCS.2017.76. 1

[BLW19] G. Borradaile, H. Le, and C. Wulff-Nilsen. Greedy spanners are optimal in doubling metrics. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2371–2379, 2019, doi:10.1137/1.9781611975482.145. 1, 3

[BS07] S. Baswana and S. Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30(4):532–563, 2007, doi:10.1002/rsa.20130. 1, 2, 4, 12, 13

[CDNS95] B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. *Int. J. Comput. Geometry Appl.*, 5:125–144, 1995, doi:10.1142/S0218195995000088. 1, 3, 12

[Coh97] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997, doi:10.1006/jcss.1997.1534. 5, 17

[Coh98] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.*, 28(1):210–236, 1998, doi:10.1137/S0097539794261295. 1

[CW18] S. Chechik and C. Wulff-Nilsen. Near-optimal light spanners. *ACM Trans. Algorithms*, 14(3):33:1–33:15, 2018, doi:10.1145/3199607. 1

[DGPV08] B. Derbel, C. Gavoille, D. Peleg, and L. Viennot. On the locality of distributed sparse spanner construction. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*, pages 273–282, 2008, doi:10.1145/1400751.1400788. 1

[DHN93] G. Das, P. J. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional euclidean space. In *Proceedings of the Ninth Annual Symposium on Computational GeometrySan Diego, CA, USA, May 19-21, 1993*, pages 53–62, 1993, doi:10.1145/160985.160998. 3

[DPP06] M. Damian, S. Pandit, and S. Pemmaraju. Distributed spanner construction in doubling metric spaces. In *Proceedings of the 10th International Conference on Principles of Distributed Systems*, OPODIS'06, pages 157–171, Berlin, Heidelberg, 2006. Springer-Verlag, doi:10.1007/11945529_12. 5

[Elk04] M. Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 331–340, 2004, doi:10.1145/1007352.1007407. 2

[Elk07] M. Elkin. A near-optimal distributed fully dynamic algorithm for maintaining sparse spanners. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*, pages 185–194, 2007, doi:10.1145/1281100.1281128. 1

[Elk17a] M. Elkin. Distributed exact shortest paths in sublinear time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 757–770, 2017, doi:10.1145/3055399.3055452. 4, 7

[Elk17b] M. Elkin. A simple deterministic distributed MST algorithm, with near-optimal time and message complexities. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 157–163, 2017, doi:10.1145/3087801.3087823. 30

[EN16] M. Elkin and O. Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 128–137, 2016, doi:10.1109/FOCS.2016.22. 5, 21

[EN17] M. Elkin and O. Neiman. Linear-size hopsets with small hopbound, and distributed routing with low memory. *CoRR*, abs/1704.08468, 2017, arXiv:1704.08468. 4

[EN18] M. Elkin and O. Neiman. Near-optimal distributed routing with low memory. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 207–216, New York, NY, USA, 2018. ACM, doi:10.1145/3212734.3212761. 3, 7

[EN19] M. Elkin and O. Neiman. Efficient algorithms for constructing very sparse spanners and emulators. *ACM Trans. Algorithms*, 15(1):4:1–4:29, 2019. Preliminary version appeared in SODA17, doi:10.1145/3274651. 1, 2, 4, 12, 13, 14, 15, 17

[ENS15] M. Elkin, O. Neiman, and S. Solomon. Light spanners. *SIAM J. Discrete Math.*, 29(3):1312–1321, 2015, doi:10.1137/140979538. 1, 3, 12

[EP01] M. Elkin and D. Peleg. (1+epsilon, beta)-spanner constructions for general graphs. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 173–182, 2001, doi:10.1145/380752.380797. 1

[ES16] M. Elkin and S. Solomon. Fast constructions of lightweight spanners for general graphs. *ACM Trans. Algorithms*, 12(3):29:1–29:21, 2016. See also SODA'13, doi:10.1145/2836167. 1, 3, 12

[EZ06] M. Elkin and J. Zhang. Efficient algorithms for constructing (1+epsilon, beta)-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006, doi:10.1007/s00446-005-0147-2. 1

[FL16] S. Friedrichs and C. Lenzen. Parallel metric tree embedding based on an algebraic view on moore-bellman-ford. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 455–466, 2016, doi:10.1145/2935764.2935777. 5, 18

[FN18] A. Filtser and O. Neiman. Light spanners for high dimensional norms via stochastic decompositions. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 29:1–29:15, 2018, doi:10.4230/LIPIcs.ESA.2018.29. 1, 22

[FS16] A. Filtser and S. Solomon. The greedy spanner is existentially optimal. In *PODC 2016*, pages 9–17, 2016, doi:10.1145/2933057.2933114. 1

[Gha16] M. Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 270–277, 2016, doi:10.1137/1.9781611974331.ch20. 3

[GKL03] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 534–543, 2003, doi:10.1109/SFCS.2003.1238226. 2, 20

[GL14] M. Ghaffari and C. Lenzen. Near-optimal distributed tree embedding. In *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, pages 197–211, 2014, doi:10.1007/978-3-662-45174-8_14. 18

[GL18] M. Ghaffari and J. Li. Improved distributed algorithms for exact shortest paths. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 431–444, 2018, doi:10.1145/3188745.3188948. 4, 7

[Got15] L. Gottlieb. A light metric spanner. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 759–772, 2015, doi:10.1109/FOCS.2015.52. 1, 3

[HKN16] M. Henzinger, S. Krinninger, and D. Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 489–498, 2016, doi:10.1145/2897518.2897638. 4

[HM06] S. Har-Peled and M. Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006, doi:10.1137/S0097539704446281. 2

[KKM+12] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distributed Computing*, 25(3):189–205, 2012, doi:10.1007/s00446-012-0157-9. 18

[Kle05] P. N. Klein. A linear-time approximation scheme for planar weighted TSP. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, pages 647–657, 2005, doi:10.1109/SFCS.2005.7. 3

[KP98] S. Kutten and D. Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998, doi:10.1006/jagm.1998.0929. 9, 30

[KRY95] S. Khuller, B. Raghavachari, and N. E. Young. Balancing minimum spanning trees and shortest-path trees. *Algorithmica*, 14(4):305–321, 1995, doi:10.1007/BF01294129. 1, 2, 4, 7, 8, 11

[Lub86] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1055, November 1986, doi:10.1137/0215074. 3, 5, 17

[MP98] Y. Mansour and D. Peleg. An approximation algorithm for minimum-cost network design. Technical report, Weizmann Institute of Science, Rehovot, 1998. 1

[MPVX15] G. L. Miller, R. Peng, A. Vladu, and S. C. Xu. Improved parallel algorithms for spanners and hopsets. In *Proc. of 27th SPAA*, pages 192–201, 2015, doi:10.1145/2755573.2755574. 1, 2

[MRSZ11] Y. Métivier, J. M. Robson, N. Saheb-Djahromi, and A. Zemmari. An optimal bit complexity randomized distributed MIS algorithm. *Distributed Computing*, 23(5-6):331–340, 2011, doi:10.1007/s00446-010-0121-5. 5, 17

[NZ02] T. S. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 178–187, 2002, doi:10.1109/INFCOM.2002.1019258. 2

[Pel00] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, 2000, doi:10.1137/1.9780898719772. 6

[Pet09] S. Pettie. Low distortion spanners. *ACM Trans. Algorithms*, 6(1):7:1–7:22, 2009, doi:10.1145/1644015.1644022. 1

[Pet10] S. Pettie. Distributed algorithms for ultrasparse spanners and linear size skeletons. *Distributed Computing*, 22(3):147–166, 2010, doi:10.1007/s00446-009-0091-7. 1

[PS89] D. Peleg and A. A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989, doi:10.1002/jgt.3190130114. 1

[PU89] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989, doi:10.1137/0218050. 1

[PV04] P. Penna and C. Ventre. Energy-efficient broadcasting in ad-hoc networks: combining msts with shortest-path trees. In *Proceedings of the 1st ACM International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks, PE-WASUN 2004, Venezia, Italy, October 4, 2004*, pages 61–68, 2004, doi:10.1145/1023756.1023769. 2

[RG19] V. Rozhon and M. Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. *CoRR*, abs/1907.10937, 2019, arXiv:1907.10937. 3

[SCRS01] F. S. Salman, J. Cheriyan, R. Ravi, and S. Subramanian. Approximating the single-sink link-installation problem in network design. *SIAM Journal on Optimization*, 11(3):595–610, 2001, doi:10.1137/S1052623497321432. 1

[SEW13] J. Schneider, M. Elkin, and R. Wattenhofer. Symmetry breaking depending on the chromatic number or the neighborhood growth. *Theor. Comput. Sci.*, 509:40–50, 2013, doi:10.1016/j.tcs.2012.09.004. 3

[SHK+12] A. D. Sarma, S. Holzer, L. Kor, A. Korman, D. Nanongkai, G. Pandurangan, D. Peleg, and R. Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012, doi:10.1137/11085178X. 2, 23

[TSL00] J. B. Tenenbaum, V. d. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000, doi:10.1126/science.290.5500.2319. 2

[TZ06] M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 802–809, 2006. http://dl.acm.org/citation.cfm?id=1109557.1109645. 1

[WCT02] B. Y. Wu, K.-M. Chao, and C. Y. Tang. Light graphs with small routing cost. *Networks*, 39(3):130–138, 2002, doi:10.1002/net.10019. 1

[YCC06] J. Yu, L. Chen, and G. Chen. Priority based overlay multicast with filtering mechanism for distributed interactive applications. In *10th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2006), 2-4 October 2006, Malaga, Spain*, pages 127–134, 2006, doi:10.1109/DS-RT.2006.29. 2

A Proof of Lemma 2

Recall the statement of the lemma:

Lemma 2 (MST traversal)**.** *Let $G = (V, E, w)$ be a weighted graph with n vertices, hop-diameter D and root $rt \in V$, then there is a deterministic algorithm in the CONGEST model that computes $\mathcal{L}$ in $\tilde{O}(\sqrt{n} + D)$ rounds.*

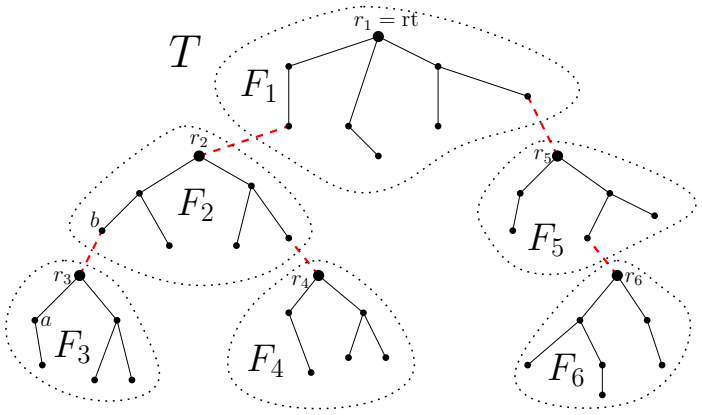


Figure 1: *The tree T in the figure is divided to 6 base fragments circled by a dotted line. The internal edges are colored black, while the external edges are dashed and colored red. The fragment R_i is rooted in r_i, a vertex with outgoing edge towards a parent fragment. For example, consider the case where all the edges in T have unit weight. Sample of local lengths: $\ell(a) = 2$, $\ell(b) = 0$, $\ell(r_1) = 14$, and of global lengths $g(a) = 2$, $g(b) = 12$, $g(r_1) = 78$.*

A.1 Computing the MST Fragments Tree

In [Elk17b], following [KP98], a deterministic MST construction in the CONGEST model with $\tilde{O}(\sqrt{n} + D)$ rounds was shown. The algorithm has two phases, according to the hop-diameter of the fragments. At the end of the first phase, there is a set of $O(\sqrt{n})$ fragments $\mathcal{F} = \{F_1, F_2, \dots\}$, each with hop-diameter $O(\sqrt{n})$. These fragments are called *base fragments*. The edges added in the first phase are called *internal* edges (as each such edge is internal to some base fragment). In the second phase of the algorithm, the remaining $O(\sqrt{n})$ edges are added and connect the fragments to a tree. We call these edges *external* edges, as they cross between base fragments.

Let T' be a virtual tree with the base fragments $\mathcal{F}$ as vertices, and with the external edges as its edge set (i.e. there is an edge between F_i and F_j if there is an external edge between a vertex in F_i to a vertex in F_j). Since there are $O(\sqrt{n})$ vertices in T', in $O(\sqrt{n}+D)$ rounds we can broadcast T' to all the vertices V. We will think of the MST T as a tree rooted in rt, and of T' as a tree rooted at the fragment F_1 containing rt. Using the information above, every vertex in each base fragment F_i can learn the structure of the MST on the fragments T', and infer its parent base fragment $p(F_i)$. If the MST edge connecting F_i to $p(F_i)$ is (u, v) (where $u \in F_i$), then $p(u) = v$, and we set $r_i = u$ to be the root of the base fragment F_i. For F_1, $r_1 = rt$ will be its root. Set $\mathcal{R} = \{r_1, r_2, \dots\}$ to be the set of base fragment root vertices. See Figure 1 for an illustration.

A.2 Computing Tour Lengths

For $v \in F_i$, let $\ell(v)$ denote the length of the tour of the subtree of F_i rooted at v. This is the local tour length for v, which is simply twice the sum of edge weights in that subtree. In addition, denote by $g(x)$ the length of the tour of the subtree of T rooted at v, which is the global tour length for v. See Figure 1 for an example.

The computations of the local tour lengths is done locally in each base fragment, i.e. in all the base fragments in parallel. Consider F_i. Initially $\ell(v) = 0$ for every leaf $v \in F_i$. Every

intermediate vertex $u \in F_i$ that received messages from all its children in F_i, denoted $z_1, \ldots, z_k$, computes $\ell(u) = \sum_{i=1}^{k} (\ell(z_i) + 2 \cdot w(u, z_i))$, and sends $\ell(u)$ to its own parent. Using the bounded hop-diameter of the base fragments, this procedure will terminate in $O(\sqrt{n})$ rounds. When this stage concludes, every $v \in V$ knows $\ell(v)$.

After the computation of the local tour lengths, all the root vertices $\mathcal{R}$ broadcast to the entire graph their local tour lengths $\ell(r_1), \ell(r_2) \ldots$ in $O(\sqrt{n} + D)$ rounds. As the tree T' is known to the entire graph, all the vertices can compute locally the global tour lengths $g(r_1), g(r_2) \ldots$ of the roots. Specifically, for r_i the root of the base fragment F_i, consider its descendent base fragments $\mathcal{F}'$ in T'. Denote by r_F the root of the base fragment F and by e_F the external edge connecting F to $p(F)$, its parent fragment. Then

$$g(r_i) = \ell(r_i) + \sum_{F \in \mathcal{F}'} (\ell(r_F) + 2 \cdot w(e_F)) \ .$$

The computation of the global tour lengths for non-roots is done locally in a similar manner as the local tour lengths. For a vertex $v \in F_i$, let $\tilde{z}_1, \ldots, \tilde{z}_k$ be its children in T, then

$$g(v) = \sum_{i=1}^{k} (g(\tilde{z}_i) + 2 \cdot w(v, \tilde{z}_i)) \ .$$

If v is a leaf of some F_i then it can compute $g(v)$ (since all its children are in $\mathcal{R}$), and then send $g(v)$ to it parent. Every intermediate vertex v of F_i can compute $g(v)$ as soon as it received messages from all its children in F_i, and then send $g(v)$ to its own parent. As we run this procedure in all the fragments in parallel, it terminates in $O(\sqrt{n})$ rounds. When this stage concludes, every $v \in V$ knows $g(v)$.

A.3 Computing Tour Visit Times

Finally we compute for every vertex $v \in V$ the set $\mathcal{L}(v)$ and all its tour visiting times. This can be achieved by running a DFS search from the root rt. Direct implementation of the DFS algorithm will take $\Omega(n)$ rounds. Instead, we will use a similar idea to the one above to speed-up the computation. First we will compute "local" DFS in the base fragments. Then aggregate these times into a global DFS, first for the roots $\mathcal{R}$, and afterwards to all of V.

First we compute the "local" DFS visiting times. We will compute the visiting times in all the fragments in parallel. Consider F_i. For every $v \in F_i$ we will compute the entering and exit DFS time, for the global subtree of T rooted at r_i (rather than only in F_i). For a vertex $v \in F_i$, denote by $t_i(v)$ the interval between the first "local" entrance to the last exit. The computation is performed top to bottom. First for r_i, $t_i(r_i) = [0, g(r_i)]$. Next, consider a vertex $v \in F_i$ that received its interval $t_i(v) = [a, b]$. By induction $b - a = g(v)$. Denote by $z_1, \ldots, z_k$ all the children of v in T (inside and outside F_i). Then v will send each child z_j its interval, where

$$t_i(z_j) = \left[a + \sum_{q<j} (g(z_q) + 2w(v, z_q)) + w(v, z_j) \ , \ a + \sum_{q<j} (g(z_q) + 2w(v, z_q)) + w(v, z_j) + g(z_j) \right] \ .$$

Note that the length of the interval $t_i(z_j)$ is exactly $g(z_j)$. (We remark that roots in $\mathcal{R}$ do not initiate another interval assignment when they receive message from their parent in T.) This procedure

will terminate in $O(\sqrt{n})$ rounds (the hop-diameter of any base fragment), where each vertex $v \in F_i$ knows $t_i(v)$. Moreover, consider a root vertex r_i (other than rt), such that $p(r_i) \in F_j$. Then by the description of the algorithm, in addition to $t_i(r_i)$, t_i also knows $t_j(r_i)$, its interval in the DFS tour in the subtree rooted in r_j.

Finally we are ready to compute the global DFS intervals. Note that all we are actually missing here, is the first time visit of each root vertex. That is, we will compute a shift s_i for every root r_i. First, each root vertex r_i broadcasts to rt (through τ), its local interval $t_i(r_i)$ and its local interval in its parent fragment $t_j(r_i)$ (assuming $p(r_i) \in F_j$). This take $O(\sqrt{n} + D)$ rounds. Now, rt has all the required information in order to compute the DFS intervals for $\mathcal{R}$. Initially $t(rt) = t_1(rt)$. Next, by induction assume that rt computed the interval $t(r_j) = [s_j, s_j + g(r_j)]$ for some $r_j \in \mathcal{R}$. Then for every $r_i \in \mathcal{R}$ such $p(r_i) \in F_j$ and $t_j(r_i) = [b, b + g(r_i)]$, we compute $t(r_i) = [s_j + b, s_j + b + g(r_i)] = [s_i, s_i + g(r_i)]$. Eventually rt knows the global DFS intervals of all the roots $\mathcal{R}$ and can broadcast it to all the vertices in $O(\sqrt{n} + D)$ rounds. Consider a vertex $v \in F_j$. Given its local interval $t_j(v) = [a, b]$ and the shift s_j, v computes its global interval $t(v) = [s_j + a, s_j + b]$. This is done locally in all the fragments.

We conclude that in $\tilde{O}(\sqrt{n} + D)$ rounds every vertex v can know its DFS interval. As every vertex can also compute the DFS intervals of it's children (in T), both $\mathcal{L}(v)$ and the visiting times of $\mathcal{L}(v)$ are easily computed.