Distributed Construction of Light Networks

Extended Abstract*

Michael Elkin†
Ben-Gurion University of the Negev
elkinm@cs.bgu.ac.il

Arnold Filtser‡
Columbia University
arnold273@gmail.com

Ofer Neiman§
Ben-Gurion University of the Negev
neiman@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 times 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.

CCS CONCEPTS

• **Theory of computation → Routing and network design problems**; *Distributed algorithms.*

KEYWORDS

CONGEST, Light Spanners, Shallow light tree, Doubling Dimension

ACM Reference Format:
Michael Elkin, Arnold Filtser, and Ofer Neiman. 2020. Distributed Construction of Light Networks: Extended Abstract. In *ACM Symposium on Principles of Distributed Computing (PODC '20), August 3–7, 2020, Virtual Event, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3382734.3405701

1 INTRODUCTION

Let $G = (V, E, w)$ be a graph with edge weights $w : E \to \mathbb{R}_+$. 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 [8, 44] 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 [1, 2, 4, 6, 12, 20, 23, 25, 30, 32, 34, 50, 53, 54, 56, 57, 63]. The basic greedy algorithm [4], 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 [3, 11, 15–18, 31, 33, 35, 40], and are particularly useful in a distributed setting; efficient broadcast protocols, network synchronization and computing global functions [7, 8], network design [48, 59] and routing [64] are a few examples. The state-of-the-art is a $(2k-1)\cdot(1+\epsilon)$-spanner of [18] with lightness $O(n^{1/k})$, for any constant $0 < \epsilon < 1$. In [36]

*The reader is encouraged to read the full version of the paper, found here.

†This research was supported by the ISF grant No. (724/15).
‡Supported by the Simons Foundation. The work was done while the author was affiliated with Ben-Gurion University of the Negev.
§Supported in part by ISF grant 1817/17, and by BSF Grant 2015813.

PODC '20, August 3–7, 2020, Virtual Event, Italy

ACM ISBN 978-1-4503-7582-5/20/08…$15.00
https://doi.org/10.1145/3382734.3405701

Object	Distortion	Lightness	Size	Run time
Spanner	$(2k-1)\cdot(1+\epsilon)$	$O(k\cdot n^{1/k})$	$O(k\cdot n^{1+1/k})$	$\tilde{O}\left(n^{\frac{1}{2}+\frac{1}{4k+2}}+D\right)$
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	$1+\epsilon$	$\epsilon^{-O(\text{ddim})}\cdot\log n$	$n\cdot\epsilon^{-O(\text{ddim})}\cdot\log n$	$(\sqrt{n}+D)\cdot\epsilon^{-\tilde{O}(\sqrt{\log n}+\text{ddim})}$

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 $0<\epsilon<1$ is a constant. For the nets $\gamma>\beta>0$. All results are in the CONGEST model.

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. [33] showed a near-linear time algorithm that constructs a $(2k-1)\cdot(1+\epsilon)$-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 [3] to $O(\log k\cdot n^{1+1/k})$ and $O(\log k\cdot n^{1/k})$ respectively. In the distributed setting, [12] devised a randomized algorithm for a $(2k-1)$-spanner with $O(k\cdot n^{1+1/k})$ edges in $O(k)$ rounds[1] in the CONGEST model. This was recently improved for unweighted graphs by [30, 50] 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 [60] 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[2] D, any CONGEST algorithm requires at least $\tilde{\Omega}(\sqrt{n}+D)$ rounds for computing a light spanner (with any polynomial lightness). [3]

Our result. We provide the first algorithm with sub-linear number of rounds for constructing light spanners for general graphs in the CONGEST model. In detail, for any integer parameter $k\geq 1$ and constant $0<\epsilon<1$, we devise a randomized algorithm that w.h.p. outputs a $(2k-1)\cdot(1+\epsilon)$-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.

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 [14, 55, 65]. In [44], 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, [44] 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, [44] 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 $\tilde{O}\left(\sqrt{n}+D\right)\cdot\text{poly}\left(\frac{1}{\alpha-1}\right)$ rounds. See Theorem 1. Recall that by [24], any distributed SLT algorithm requires at least $\tilde{\Omega}(\sqrt{n}+D)$ rounds. Thus our result is nearly optimal.

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[4] $B_G(v,2r)$ 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 [5, 41, 42], and it is believed that such graphs occur often in real-life networks and data [51, 62]. One notable motivation for light spanners in doubling graphs[5] is their application for polynomial approximation schemes for the traveling salesperson and related problems (see, e.g., [40, 45]). While spanners with $1+\epsilon$ stretch and constant lightness have been known to exist in low dimensional Euclidean space for a while [4, 22], only recently such $(1+\epsilon)$-spanners with constant lightness $(\text{ddim}/\epsilon)^{O(\text{ddim})}$ have been discovered for doubling graphs [40]. The lightness was improved by [16] to the optimal $(1/\epsilon)^{O(\text{ddim})}$.

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.

[1] The paper claimed only $O(k^2)$ rounds, but it has been observed that $O(k)$ is also possible.

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

[3] The notations $\tilde{O}(\cdot)$ hide poly-logarithmic factors. That is $f=\tilde{O}(g)$ implies $f=O(g\cdot\text{polylog}(g))$. Similarly for $\tilde{\Omega}(\cdot)$.

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

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

A *ruling set* is a net in an unweighted graph. There have been several works that compute a ruling set in distributed settings. In [9], 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 [61]. A consequence of the work of [9] provides a (k, k)-ruling set computed within $k \cdot 2^{\tilde{O}(\sqrt{\log n})}$ rounds, the running time can be improved to $O(k \cdot \text{poly} \log n)$ using the recent breakthrough of [58]. A randomized algorithm for a (k, k)-ruling set was given in [47] with $O(k \log n)$ rounds, and the running time was improved for graphs of small maximum degree in [10, 38]. 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 (the proof is deferred to the full version). 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) (the proof is deferred to the full version). 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 $0 < \epsilon < 1$, w.h.p. produces a $(1 + \epsilon)$-spanner with lightness $\epsilon^{-O(\text{ddim})} \cdot \log n$ in $(\sqrt{n} + D) \cdot \epsilon^{-\tilde{O}(\sqrt{\log n} + \text{ddim})}$ rounds.

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 [29].

Light Spanner for General Graphs. From a high level, our approach is similar to the algorithms of [17, 31, 33]. 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 [12] for weighted graphs.

Consider the i-th bucket E_i, where all edges have weight in $\left(\frac{L}{(1+\epsilon)^{i+1}}, \frac{L}{(1+\epsilon)^i}\right]$. We use the MST traversal $\mathcal{L}$ to divide the graph into $O(\frac{(1+\epsilon)^i}{\epsilon})$ clusters of diameter $\frac{\epsilon \cdot L}{(1+\epsilon)^i}$. 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 [30] 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 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 [8, 44].) As currently the fastest known exact SPT algorithms [26, 39] require more than $\tilde{O}(\sqrt{n} + D)$ rounds, we use instead an approximate SPT , T' [13, 28, 43]. Our basic strategy (following [8, 44]) 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 $= \{x_0, x_{i_1}, x_{i_2}, \ldots\}$ such that (1) every pair of consecutive points $x_{i_j}, x_{i_{j+1}} \in$ BP are far from one another, specifically $d_{\mathcal{L}}(x_{i_j}, x_{i_{j+1}}) > \epsilon \cdot d_G(rt, x_{i_{j+1}})$, and (2) every node $x_q \in \mathcal{L}$ has a nearby break point $x_{i_j} \in$ BP, specifically $d_{\mathcal{L}}(x_{i_j}, x_q) \leq \epsilon \cdot d_G(rt, x_q)$. 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 [47, 49]).[6] 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 [19]. Given a permutation π, a vertex u belongs to the LE list of a vertex v, if u

[6] 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.

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 [37]. However, rather than computing the list w.r.t. the graph G, [37] compute LE list w.r.t. an auxiliary graph H that approximates G distances up to a $1+\epsilon$ factor. Fortunately, we can cope with the approximation by taking $\alpha > (1+\epsilon)\beta$. 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 $\alpha,\beta \approx \epsilon\Delta$, 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 $\epsilon^{-O(\mathrm{ddim})}$ other net points at distance Δ, and every net point must contribute to the MST weight at least $\epsilon\cdot\Delta$.

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 [27], 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.

1.5 Related Work

In the distributed LOCAL model[7], [21] devised light spanners for a certain graph family, called *unit ball graphs in a doubling metric space*[8]. Specifically, they showed an $O(\log^* n)$ rounds algorithm for a $(1+\epsilon)$-spanner with lightness $(1/\epsilon)^{O(\mathrm{ddim})}\cdot\log\Lambda$, 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, their application to light spanners for doubling graphs and the lower bounds are deferred to the full version.

[7]The LOCAL model is similar to CONGEST, but the size of messages is not bounded.
[8]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.

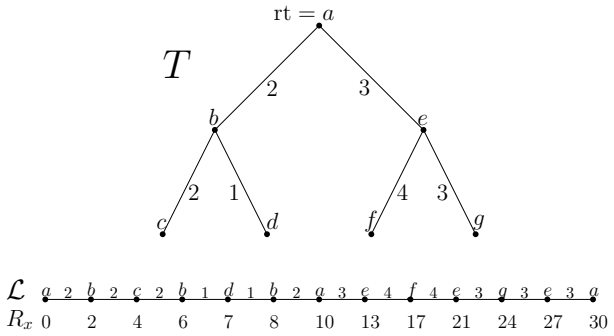

Figure 1: *Illustration of an Eulerian path of T.*

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 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., [52]).

Lemma 1. *Suppose every $v\in V$ holds m_v messages, each of $O(1)$ words[9], 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.

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 [29] 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

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

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 1 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 the following lemma we compute 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 $x \in \mathcal{L}(v)$. The proof is deferred to the full version.

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) $\forall\, v \in V,\ d_{T_{\mathrm{SLT}}}(rt, v) \leq \alpha \cdot d_G(rt, v)$, and 2) $w(T_{\mathrm{SLT}}) \leq \beta \cdot w(MST)$. 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 $\epsilon > 0$, constructs an $(1+\epsilon, 1+O(\frac{1}{\epsilon}))$-SLT in $\tilde{O}\left(\sqrt{n}+D\right) \cdot \mathrm{poly}(\epsilon^{-1})$ rounds.*

Initially we will assume that $\epsilon \in (0, 1)$. Afterwards, we will show how to generalize our result to $\epsilon \geq 1$. 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 [26, 39] require more than $\tilde{O}(\sqrt{n}+D)$ rounds. Instead, we will use an approximate shortest path tree of [13]. Specifically, they show that given a root vertex rt and a parameter $\epsilon \in (0, 1]$, one can compute an approximate shortest path tree T_{rt} in $\tilde{O}((\sqrt{n}+D)/\mathrm{poly}(\epsilon))$ rounds. The approximation here is in the sense that for every vertex v,

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

Our strategy to construct the SLT is similar to the framework of [8, 44]. 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 the paths in T_{rt} from rt to all the vertices in BP. We will argue that H has lightness $O(\frac{1}{\epsilon})$, and approximate distance to rt up to a $1+O(\epsilon)$ factor. Our final SLT will be constructed as yet another approximate shortest path tree in H (rooted at rt). The pseudo-code of the algorithm appears in the full version. 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 [44] 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 $\left(x_{i\alpha}, R_{x_{i\alpha}}\right)$. Generally, each vertex $x_j \in I_i$ will get at some point a message $\left(y, R_y\right)$ 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:

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

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 $\left(x_j, R_{x_j}\right)$. Otherwise the message will be $\left(y, R_y\right)$. After $\alpha - 1$ rounds this procedure ends, and each internal vertex $x \notin \mathrm{BP}'$ 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 $x_{i\alpha} \in \mathrm{BP}'$ will broadcast to rt the message $\left(x_{i\alpha}, R_{x_{i\alpha}}\right)$. 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 $\mathrm{BP}_2 \subseteq \mathrm{BP}'$ as follows. Initially $rt = x_0$ joins BP_2. Next, sequentially, for $x_{i\alpha}$ where $y \in \mathrm{BP}'$ was the last vertex to join BP_2, $x_{i\alpha}$ will join BP_2 if $d_{\mathcal{L}}(x_{i\alpha}, y) = R_{x_{i\alpha}} - R_y > \epsilon \cdot d_{T_{rt}}(rt, x_{i\alpha})$. After this local computation, rt will broadcast to the entire graph the set BP_2 in $O(\sqrt{n}+D)$ rounds (using Lemma 1). Define the final set of breakpoints as $\mathrm{BP} = \mathrm{BP}_1 \cup \mathrm{BP}_2$. (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 $b \in \mathrm{BP}$, let P_b be the unique path from rt to b in T_{rt}. Let $H = T \cup \bigcup_{b \in \mathrm{BP}} P_b$. 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 $v \in A_{\mathrm{BP}}$ 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 [46], more details appear in the full version). 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 $r_i \in A_{\mathrm{BP}}$ (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 $v \in A_{\mathrm{BP}}$.

4.3 Stretch and Lightness Analysis

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

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

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

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

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

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

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

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

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

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

(it might be that $x = y$ or $y = y'$). We first bound $d_{\mathcal{L}}(x, y')$, using that $d_G \le d_{\mathcal{L}}$ and the assumption $\epsilon \le 1$.

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

We conclude,

$$\begin{aligned} d_H(rt, x) &\le d_{T_{rt}}(rt, y') + d_{\mathcal{L}}(x, y') \\ &\le (1+\epsilon) \cdot \left(d_G(rt, x) + d_G(x, y')\right) + d_{\mathcal{L}}(x, y') \\ &\le (1+\epsilon) \cdot d_G(rt, x) + 3 \cdot d_{\mathcal{L}}(x, y') \le (1+25\epsilon) \cdot d_G(rt, x) . \end{aligned}$$

□

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

After creating the subgraph H, we create a $(1+\epsilon)$-shortest path tree T_{SLT} (using [13]) of H rooted at rt in $\tilde{O}(\sqrt{n}+D)/\mathrm{poly}(\epsilon)$ rounds. The tree T_{SLT} has weight at most $w(T_{\mathrm{SLT}}) \le w(H) \le (1+\frac{4}{\epsilon}) \cdot w(T)$ by Corollary 3. Moreover, by Lemma 4 for every vertex $v \in V$ it holds that

$$\begin{aligned} d_{T_{\mathrm{SLT}}}(rt, v) &\le (1+\epsilon) \cdot d_H(rt, v) \\ &\le (1+\epsilon) \cdot (1+25\epsilon) \cdot d_G(rt, v) \le (1+51\epsilon) \cdot d_G(rt, v) . \end{aligned}$$

By rescaling ϵ, we conclude that for every $\epsilon \in (0,1)$ we can construct an $(1+\epsilon, O(\frac{1}{\epsilon}))$-SLT. This is the right behavior (up to constant factors) when the distortion is small, as shown in [44]. 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 $1 < \epsilon = 1/\gamma$, 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 [11].

Lemma 5 ([11]). *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 a 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 $k \in \mathbb{N}$, $\epsilon \in (0,1)$, in $\tilde{O}_\epsilon\left(n^{\frac{1}{2}+\frac{1}{4k+2}} + D\right)$ rounds, w.h.p. returns a $(2k-1)(1+\epsilon)$ spanner H with $O_\epsilon(k \cdot n^{1+\frac{1}{k}})$ edges and lightness $O_\epsilon(k \cdot n^{\frac{1}{k}})$.*

Our algorithm is similar in spirit to the algorithms of [17, 31, 33]. The pseudo-code of the algorithm appears in the full version. 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 $E' = \{e \in E : w(e) \le L/n\}$, and for every $i \in \{0, 1, \ldots, \lceil \log_{1+\epsilon} n \rceil\}$ set $E_i = \{e \in E : \frac{L}{(1+\epsilon)^{i+1}} < w(e) \le \frac{L}{(1+\epsilon)^i}\}$. 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 [12]. Specifically we run [12] on the graph $G' = (V, E')$. In $O(k)$ rounds[10] 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 $i \in \{0, 1, \ldots, \lceil \log_{1+\epsilon} n \rceil\}$ 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 [31, 33] 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 [30] 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 [30] gave an efficient distributed implementation, the input graph $\mathcal{G}_i$ is not the communication graph G. Our 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 [30] 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 $u = \text{argmax}_{v \in N^+(x)}\{m(v)\}$, 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 $\{v \in N(x) : m(v) \ge m(x) - 1 \wedge s(v) = y\}$, 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[11] while the number of edges is bounded in expectation.

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

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 $m(A)$.

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

[11] For the stretch bound, the random samples $r(x)$ need to satisfy $r(x) < k$, which can be verified locally. The stretch analysis in [30] is conditioned on the event $\forall x \in V : r(x) < k$.

Therefore each vertex will forward only $|C_i|$ messages, and we can pipeline all the messages of the second phase in $O(|C_i|+D)$ rounds. (3) Broadcast phase: the root rt broadcasts all the new messages $(s(A), m(A))$ for all clusters $A \in C_i$ to all the graph in $O(|C_i| + D)$ 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 [30] it is shown that in expectation $|\mathcal{H}_i| = O(|C_i|^{1+\frac{1}{k}})$. 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 $O(|C_i|^{1+\frac{1}{k}} + D)$ 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 $O(|C_i|^{1+\frac{1}{k}} + D)$ rounds rt broadcasts all these edges and H_i is created accordingly. The total number of rounds to implement each iteration of [30] is

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

Case 2: $\log_{1+\epsilon}(\epsilon \cdot n^{\frac{k}{2k+1}}) < i \leq \log_{1+\epsilon}(n)$. Set $w_i = \frac{L}{(1+\epsilon)^i}$. Similarly to the previous regime, we will partition the graph into clusters C_i with weak diameter $\epsilon \cdot w_i$. 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 $R_{x_{j-1}} < s \cdot (\epsilon \cdot w_i) \leq R_{x_j}$.
(2) j is a multiple of $\left\lceil \frac{\epsilon \cdot n}{(1+\epsilon)^i} \right\rceil$ (that is, there is an integer q such that $j = q \cdot \left\lceil \frac{\epsilon \cdot n}{(1+\epsilon)^i} \right\rceil$).

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 $R_{x_b} - R_{x_a} < \epsilon \cdot w_i$ and $b - a < \frac{\epsilon \cdot n}{(1+\epsilon)^i}$. Moreover, the total number of centers is bounded by $\frac{L}{\epsilon \cdot w_i} + \frac{n}{\frac{\epsilon \cdot n}{(1+\epsilon)^i}} = \frac{2 \cdot (1+\epsilon)^i}{\epsilon}$. 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 $\frac{\epsilon \cdot n}{(1+\epsilon)^i} \leq n^{\frac{1}{2}+\frac{1}{4k+2}}$, and also for any $u, v \in C(x_a)$ we have $d_T(u, v) \leq \epsilon \cdot w_i$. 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 $\frac{\epsilon \cdot n}{(1+\epsilon)^i}$ 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 [30] 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 [30] it is shown that w.h.p. every vertex (cluster) adds at most $O(|C_i|^{1/k} \log n) = O(n^{1/k} \log n)$ 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 $0 \leq i \leq \lceil \log_{1+\epsilon} n \rceil$. As the there are $O(\log_{1+\epsilon} n)$ 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 $d_H(u, v) \leq (2k-1)(1+\epsilon)w(e)$. In fact, we will show a bound of $(2k-1)(1+O(\epsilon))$ 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 $\frac{w_i}{(1+\epsilon)} < w(e) \leq w_i$ for $w_i = \frac{L}{(1+\epsilon)^i}$. Let $A_u, A_v \in C_i$ be the clusters containing u, v respectively. If $A_u = A_v$, then $d_H(u, v) \leq d_T(u, v) \leq \epsilon \cdot w_i \leq w(e)$ (assuming $\epsilon < 1/2$, 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 $\epsilon \cdot w_i$ and the weight of all the edges in H_i is bounded by w_i we conclude

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

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 $O(|C_i|^{1+\frac{1}{k}}) = O\left(\left(\frac{(1+\epsilon)^i}{\epsilon}\right)^{1+\frac{1}{k}}\right)$ edges, all of weight bounded by $w_i = \frac{L}{(1+\epsilon)^i}$. So the expected weight of all these H_i together is

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

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

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

Sparsity. Following the analysis of the lightness, we have

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

We conclude,

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

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 [30] for weighted graphs in the CONGEST model, which provides a $(2k-1)\cdot(1+\epsilon)$-spanner with $O_\epsilon(\log k\cdot n^{1+1/k})$ 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 $O_\epsilon(\log k\cdot n^{1+1/k})$. 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.

REFERENCES

[1] Amir Abboud and Greg Bodwin. 2016. 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*. 351–361. https://doi.org/10.1145/2897518.2897555

[2] Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. 1999. Fast Estimation of Diameter and Shortest Paths (Without Matrix Multiplication). *SIAM J. Comput.* 28, 4 (1999), 1167–1181. https://doi.org/10.1137/S0097539796303421

[3] Stephen Alstrup, Søren Dahlgaard, Arnold Filtser, Morten Stöckel, and Christian Wulff-Nilsen. 2017. Constructing Light Spanners Deterministically in Near-Linear Time. *CoRR* abs/1709.01960 (2017). arXiv:1709.01960 http://arxiv.org/abs/1709.01960

[4] Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. 1993. On Sparse Spanners of Weighted Graphs. *Discrete & Computational Geometry* 9 (1993), 81–100. https://doi.org/10.1007/BF02189308

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

[6] Baruch Awerbuch. 1985. Complexity of network synchronization. *J. ACM* 32, 4 (Oct. 1985), 804–823. https://doi.org/10.1145/4221.4227

[7] Baruch Awerbuch, Alan E. Baratz, and David Peleg. 1990. 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*. 177–187. https://doi.org/10.1145/93385.93417

[8] Baruch Awerbuch, Alan E. Baratz, and David Peleg. 1992. *Efficient broadcast and light-weight spanners.* Technical Report CS92-22. The Weizmann Institute of Science, Rehovot, Israel.

[9] Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, and Serge A. Plotkin. 1989. 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*. 364–369. https://doi.org/10.1109/SFCS.1989.63504

[10] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. 2012. 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*. 321–330. https://doi.org/10.1109/FOCS.2012.60

[11] Yair Bartal, Arnold Filtser, and Ofer Neiman. 2019. On notions of distortion and an almost minimum spanning tree with constant average distortion. *J. Comput. System Sci.* (2019). https://doi.org/10.1016/j.jcss.2019.04.006 Preliminary version appeared in SODA16.

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

[13] Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. 2017. 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*. 7:1–7:16. https://doi.org/10.4230/LIPIcs.DISC.2017.7

[14] Yehuda Ben-Shimol, Amit Dvir, and Michael Segal. 2004. 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*. 1011–1015. https://doi.org/10.1109/PIMRC.2004.1373851

[15] Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. 2017. 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*. 767–778. https://doi.org/10.1109/FOCS.2017.76

[16] Glencora Borradaile, Hung Le, and Christian Wulff-Nilsen. 2019. 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*. 2371–2379. https://doi.org/10.1137/1.9781611975482.145

[17] Barun Chandra, Gautam Das, Giri Narasimhan, and José Soares. 1995. New sparseness results on graph spanners. *Int. J. Comput. Geometry Appl.* 5 (1995), 125–144. https://doi.org/10.1142/S0218195995000088

[18] Shiri Chechik and Christian Wulff-Nilsen. 2018. Near-Optimal Light Spanners. *ACM Trans. Algorithms* 14, 3 (2018), 33:1–33:15. https://doi.org/10.1145/3199607

[19] Edith Cohen. 1997. Size-Estimation Framework with Applications to Transitive Closure and Reachability. *J. Comput. Syst. Sci.* 55, 3 (1997), 441–453. https://doi.org/10.1006/jcss.1997.1534

[20] Edith Cohen. 1998. Fast Algorithms for Constructing t-Spanners and Paths with Stretch t. *SIAM J. Comput.* 28, 1 (1998), 210–236. https://doi.org/10.1137/S0097539794261295

[21] Mirela Damian, Saurav Pandit, and Sriram Pemmaraju. 2006. Distributed Spanner Construction in Doubling Metric Spaces. In *Proceedings of the 10th International Conference on Principles of Distributed Systems* (Bordeaux, France) *(OPODIS'06)*. Springer-Verlag, Berlin, Heidelberg, 157–171. https://doi.org/10.1007/11945529_12

[22] Gautam Das, Paul J. Heffernan, and Giri Narasimhan. 1993. 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*. 53–62. https://doi.org/10.1145/160985.160998

[23] Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. 2008. 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*. 273–282. https://doi.org/10.1145/1400751.1400788

[24] Michael Elkin. 2004. 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*. 331–340. https://doi.org/10.1145/1007352.1007407

[25] Michael Elkin. 2007. 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*. 185–194. https://doi.org/10.1145/1281100.1281128

[26] Michael Elkin. 2017. 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*. 757–770. https://doi.org/10.1145/3055399.3055452

[27] Michael Elkin and Ofer Neiman. 2016. 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*. 128–137. https://doi.org/10.1109/FOCS.2016.22

[28] Michael Elkin and Ofer Neiman. 2017. Linear-Size Hopsets with Small Hopbound, and Distributed Routing with Low Memory. *CoRR* abs/1704.08468 (2017). arXiv:1704.08468 http://arxiv.org/abs/1704.08468

[29] Michael Elkin and Ofer Neiman. 2018. Near-Optimal Distributed Routing with Low Memory. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing* (Egham, United Kingdom) *(PODC '18)*. ACM, New York, NY, USA, 207–216. https://doi.org/10.1145/3212734.3212761

[30] Michael Elkin and Ofer Neiman. 2019. Efficient Algorithms for Constructing Very Sparse Spanners and Emulators. *ACM Trans. Algorithms* 15, 1 (2019), 4:1–4:29. https://doi.org/10.1145/3274651 Preliminary version appeared in SODA17.

[31] Michael Elkin, Ofer Neiman, and Shay Solomon. 2015. Light Spanners. *SIAM J. Discrete Math.* 29, 3 (2015), 1312–1321. https://doi.org/10.1137/140979538

[32] Michael Elkin and David Peleg. 2001. (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*. 173–182. https://doi.org/10.1145/380752.380797

[33] Michael Elkin and Shay Solomon. 2016. Fast Constructions of Lightweight Spanners for General Graphs. *ACM Trans. Algorithms* 12, 3 (2016), 29:1–29:21. https://doi.org/10.1145/2836167 See also SODA'13.

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

[35] Arnold Filtser and Ofer Neiman. 2018. Light Spanners for High Dimensional Norms via Stochastic Decompositions. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*. 29:1–29:15. https://doi.org/10.4230/LIPIcs.ESA.2018.29

[36] Arnold Filtser and Shay Solomon. 2016. The Greedy Spanner is Existentially Optimal. In *PODC 2016*. 9–17. https://doi.org/10.1145/2933057.2933114

[37] Stephan Friedrichs and Christoph Lenzen. 2016. 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*. 455–466. https://doi.org/10.1145/2935764.2935777

[38] Mohsen Ghaffari. 2016. 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*. 270–277. https://doi.org/10.1137/1.9781611974331.ch20

[39] Mohsen Ghaffari and Jason Li. 2018. 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*. 431–444. https://doi.org/10.1145/3188745.3188948

[40] Lee-Ad Gottlieb. 2015. A Light Metric Spanner. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*. 759–772. https://doi.org/10.1109/FOCS.2015.52

[41] Anupam Gupta, Robert Krauthgamer, and James R. Lee. 2003. 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*. 534–543. https://doi.org/10.1109/SFCS.2003.1238226

[42] Sariel Har-Peled and Manor Mendel. 2006. Fast Construction of Nets in Low-Dimensional Metrics and Their Applications. *SIAM J. Comput.* 35, 5 (2006), 1148–1184. https://doi.org/10.1137/S0097539704446281

[43] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 2016. 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*. 489–498. https://doi.org/10.1145/2897518.2897638

[44] Samir Khuller, Balaji Raghavachari, and Neal E. Young. 1995. Balancing Minimum Spanning Trees and Shortest-Path Trees. *Algorithmica* 14, 4 (1995), 305–321. https://doi.org/10.1007/BF01294129

[45] Philip N. Klein. 2005. 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*. 647–657. https://doi.org/10.1109/SFCS.2005.7

[46] Shay Kutten and David Peleg. 1998. Fast Distributed Construction of Small k-Dominating Sets and Applications. *J. Algorithms* 28, 1 (1998), 40–66. https://doi.org/10.1006/jagm.1998.0929

[47] Michael Luby. 1986. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM J. Comput.* 15, 4 (Nov. 1986), 1036–1055. https://doi.org/10.1137/0215074

[48] Yishay Mansour and David Peleg. 1998. *An Approximation Algorithm for Minimum-Cost Network Design*. Technical Report. Weizmann Institute of Science, Rehovot.

[49] Yves Métivier, John Michael Robson, Nasser Saheb-Djahromi, and Akka Zemmari. 2011. An optimal bit complexity randomized distributed MIS algorithm. *Distributed Computing* 23, 5-6 (2011), 331–340. https://doi.org/10.1007/s00446-010-0121-5

[50] Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. 2015. Improved Parallel Algorithms for Spanners and Hopsets. In *Proc. of 27th SPAA*. 192–201. https://doi.org/10.1145/2755573.2755574

[51] T. S. Eugene Ng and Hui Zhang. 2002. Predicting Internet Network Distance with Coordinates-Based Approaches. In *21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*. 178–187. https://doi.org/10.1109/INFCOM.2002.1019258

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

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

[54] David Peleg and Jeffrey D. Ullman. 1989. An Optimal Synchronizer for the Hypercube. *SIAM J. Comput.* 18, 4 (1989), 740–747. https://doi.org/10.1137/0218050

[55] Paolo Penna and Carmine Ventre. 2004. 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*. 61–68. https://doi.org/10.1145/1023756.1023769

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

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

[58] Václav Rozhon and Mohsen Ghaffari. 2019. Polylogarithmic-Time Deterministic Network Decomposition and Distributed Derandomization. *CoRR* abs/1907.10937 (2019). arXiv:1907.10937 http://arxiv.org/abs/1907.10937

[59] F. Sibel Salman, Joseph Cheriyan, R. Ravi, and S. Subramanian. 2001. Approximating the Single-Sink Link-Installation Problem in Network Design. *SIAM Journal on Optimization* 11, 3 (2001), 595–610. https://doi.org/10.1137/S1052623497321432

[60] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. 2012. Distributed Verification and Hardness of Distributed Approximation. *SIAM J. Comput.* 41, 5 (2012), 1235–1265. https://doi.org/10.1137/11085178X

[61] Johannes Schneider, Michael Elkin, and Roger Wattenhofer. 2013. Symmetry breaking depending on the chromatic number or the neighborhood growth. *Theor. Comput. Sci.* 509 (2013), 40–50. https://doi.org/10.1016/j.tcs.2012.09.004

[62] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. 2000. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science* 290, 5500 (2000), 2319–2323. https://doi.org/10.1126/science.290.5500.2319

[63] Mikkel Thorup and Uri Zwick. 2006. 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*. 802–809. http://dl.acm.org/citation.cfm?id=1109557.1109645 http://dl.acm.org/citation.cfm?id=1109557.1109645.

[64] Bang Ye Wu, Kun-Mao Chao, and Chuan Yi Tang. 2002. Light graphs with small routing cost. *Networks* 39, 3 (2002), 130–138. https://doi.org/10.1002/net.10019

[65] Jun Yu, Ling Chen, and Gencai Chen. 2006. 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*. 127–134. https://doi.org/10.1109/DS-RT.2006.29