New Doubling Spanners: Better and Simpler

T-H. Hubert Chan[⋆], Mingfei Li[⋆], Li Ning[⋆], and Shay Solomon[⋆⋆]

Abstract. In a seminal STOC'95 paper, Arya et al. conjectured that spanners for low-dimensional Euclidean spaces with constant maximum degree, hop-diameter $O(\log n)$ and lightness $O(\log n)$ (i.e., weight $O(\log n) \cdot w(\mathsf{MST})$) can be constructed in $O(n \log n)$ time. This conjecture, which became a central open question in this area, was resolved in the affirmative by Elkin and Solomon in STOC'13 (even for doubling metrics).
In this work we present a simpler construction of spanners for doubling metrics with the above guarantees. Moreover, our construction extends in a simple and natural way to provide k-fault tolerant spanners with maximum degree $O(k^2)$, hop-diameter $O(\log n)$ and lightness $O(k^2 \log n)$.

1 Introduction

An n-point metric space (X, d) can be represented by a complete weighted graph $G = (X, E)$, where the weight $w(e)$ of an edge $e = (u, v)$ is given by $d(u, v)$. A *t-spanner* of X is a weighted subgraph $H = (X, E')$ of G (where $E' \subseteq E$ has the same weights) that preserves all pairwise distances to within a factor of t, i.e., $d_H(u, v) \leq t \cdot d(u, v)$ for all $u, v \in X$, where $d_H(u, v)$ is the distance between u and v in H. The parameter t is called the *stretch* of the spanner H. A path between u and v in H with weight at most $t \cdot d(u, v)$ is called a *t-spanner path*.

In this paper we focus on the regime of stretch $t = 1 + \epsilon$, for an arbitrarily small $0 < \epsilon < \frac{1}{2}$. In general, there are metric spaces (such as the one corresponding to uniformly weighted complete graph), where the only possible $(1 + \epsilon)$-spanner is the complete graph. A special class of metric spaces, which has been subject to intensive research in the last decade, is the class of *doubling metrics*. The *doubling dimension* of a metric space (X, d), denoted by $\dim(X)$ (or dim when the context is clear), is the smallest value ρ such that every ball in X can be covered by 2^ρ balls of half the radius [12]. A metric space is called *doubling* if its doubling dimension is bounded by some constant. (We will sometimes disregard dependencies on ϵ and dim to avoid cluttered expressions in the text, but we provide these dependencies in all formal statements.) The doubling dimension is a generalization of the Euclidean dimension for arbitrary metric spaces, as the Euclidean space $\mathbb{R}^D$ equipped with any of the ℓ_p-norms has doubling dimension $\Theta(D)$ [12]. Spanners for doubling metrics (hereafter, *doubling spanners*), and in

[⋆] Department of Computer Science, The University of Hong Kong. Email: {`hubert,mfli,lning`}`@cs.hku.hk`

[⋆⋆] Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel. E-mail: `shay.solomon@weizmann.ac.il`. This work is supported by the Koshland Center for basic Research.

particular for low-dimensional Euclidean spaces, have been studied extensively since the mid-eighties; see [6, 1, 9, 2, 13, 10, 3, 11, 15] and the references therein.

In addition to small stretch and small number of edges, it is often desirable to optimize other parameters depending on the application. First, it is often important for the spanner to achieve a small maximum degree (or shortly, degree), hence having a small number of edges. Second, it is sometimes required that the *hop-diameter* would be small, i.e., every pair of points should be connected by a t-spanner path with a small number of edges (or *hops*). Third, it is desirable that the weight of the spanner would be at most some small factor (called *lightness*) times the weight of a minimum spanning tree (MST) of the metric space.

A natural requirement for a spanner is to be robust against node failures, meaning that even when some of the nodes in the spanner fail, the remaining part still provides a t-spanner. Formally, given a parameter $1 \leq k \leq n-2$, a spanner H of X is called a k-vertex-fault-tolerant t-spanner ((k,t)-VFTS), if for any subset $F \subseteq X$ with $|F| \leq k$, $H \setminus F$ is a t-spanner for $X \setminus F$.

1.1 Our Contribution.

The following is the main result of this paper.

Theorem 1 ($(k, 1+\epsilon)$-VFTS with Degree $O(k^2)$, Hop-Diameter $O(\log n)$ and Lightness $O(k^2 \log n)$). *Let (X,d) be an n-point metric space, and let $0 < \epsilon < 1$. Given any parameter $1 \leq k \leq n-2$, there exists a $(k, 1+\epsilon)$-VFTS with degree $\epsilon^{-O(\mathrm{dim})} \cdot k^2$, hop-diameter $O(\log n)$, and lightness $\epsilon^{-O(\mathrm{dim})} \cdot k^2 \cdot \log n$. Such a spanner can be constructed in $\epsilon^{-O(\mathrm{dim})} \cdot n \log n + \epsilon^{-O(\mathrm{dim})} \cdot k^2 n$ time.*

Research Background. We review the most relevant related work; the readers can refer to [5, 8] for a more detailed survey. In a seminal STOC'95 paper, Arya et al. [1] gave several constructions of *Euclidean spanners* that trade between degree, hop-diameter and lightness. In particular, they showed that for any n-point low-dimensional Euclidean space, a $(1+\epsilon)$-spanner with constant degree, hop-diameter $O(\log n)$ and lightness $O(\log^2 n)$ can be built in $O(n \log n)$ time.

Arya et al. [1] conjectured that the lightness bound can be improved to $O(\log n)$, without increasing the stretch, the degree and the hop-diameter of the spanner, and within the same running time $O(n \log n)$. The bound $O(\log n)$ on the lightness is optimal due to a lower bound result by Dinitz et al. [7].

This conjecture was resolved in the affirmative by Elkin and Solomon [8] only recently. In fact, Elkin and Solomon showed a stronger result: their construction works for doubling metrics, and moreover, it provides a general tradeoff between the involved parameters that is tight up to constants in the entire range.

Chan et al. [4] showed that the standard net-tree with cross edge framework (used in [9, 2]) can be modified to give a simpler spanner construction with all the desired properties, except for running time $O(n \log n)$. Combining the techniques from a previous work on k-fault tolerant spanners [5], a precursor of this paper's result was achieved, but with a worse lightness of $O(k^3 \log n)$; moreover, the running time was not analyzed in [4]. Solomon [14] made further improvements to the construction in [4], and achieved improved lightness $O(k^2 \log n)$ and running time $O(n \log n)$. This paper is the result of a collaboration between the authors of the (unpublished) manuscripts [4, 14].

1.2 Our Techniques. We first give the main ideas for constructing a spanner when all nodes are functioning. The treatment for fault-tolerance is given in Section 5. We use the *standard net-tree with cross edge framework* from [9, 2]: given a metric space (X, d), construct a hierarchical sequence $\{N_i\}$ of nets with geometrically increasing distance scales. For each $x \in N_i \subseteq X$ in level i (hereafter, *level-i net-point*), we have a node (x, i) called *incubator*. The hierarchical net structure induces an *incubator tree* ($\mathsf{IncTree}$) with the incubators as nodes. At each level, if two net-points are close together with respect to the distance scale at that level, we add a *cross edge* between their corresponding incubators.

A basic spanner [9, 2] consisting of the tree edges and the cross edges can be shown to have a low stretch. The basic idea is that for any two points u and v, we can start at the corresponding leaf nodes and climb to an appropriate level (depending on $d(u, v)$) to reach net-points u' and v' that are close to u and v, respectively, such that the cross edge $\{u', v'\}$ is guaranteed to exist. Observe that there is a 1-1 correspondence between the leaf nodes and the original points of X (i.e., each leaf corresponds to a unique point), and we later show how to label each internal node with a unique point in X using the incubator-zombie terminology.[1] Next, we analyze each of the involved parameters (degree, hop-diameter, and lightness), and explain how issues that arise can be resolved.

Degree. Since the doubling dimension is constant, each node in the net-tree has a constant number of children and a constant number of incident cross edges. However, in many net-based spanner constructions, each chain of *lonely* nodes (i.e., a chain of nodes each of which has only one child) will be *contracted*; this may increase the degree of the contracted nodes due to cross edges. The idea of constant degree single-sink spanners (used in [2, 5]) can be applied to resolve this issue. However, a simpler method is *parent replacement*, used by Gottlieb and Roditty [11] to build a *routing tree* ($\mathsf{RouTree}$) and reroute spanner paths, thus pruning unnecessary cross edges. In Section 3 we use Gottlieb and Roditty's construction to bound the degree of our spanner construction.

Hop-Diameter. Observe that there may be many levels in the net-tree. In this case, in the aforementioned spanner path between u and v, it will take many hops to go from u (respectively, v) to an appropriate ancestor u' (resp., v'). However, this can be easily fixed by adding shortcut edges to subtrees of the contracted routing tree ($\mathsf{ConRouTree}$) at "small" distance scales via the 1-spanner construction with hop-diameter $O(\log n)$ for tree metrics by Solomon and Elkin [15]; this "shortcut spanner" increases both the degree and the lightness by a constant. We shall see in Section 4 that there are only $O(\log n)$ levels with "large" distance scales, and hence hop-diameter $O(\log n)$ can achieved.

Lightness. For doubling metrics, lightness comes almost for free. We will show that the total weight of small-scale edges is $O(w(\mathsf{MST}))$. For each of the $O(\log n)$ large-scale levels, the standard analysis in [9, 2] uses the fact that for doubling metrics each net-point has a constant number of neighbors at that level. Consequently, the weight of edges from each large-scale level is only a constant times that of an MST, thereby giving lightness $O(\log n)$. See Section 4 for the details.

[1] The terminology is borrowed from [8], but these terms have different meaning there.

To summarize, we obtained a spanner on the incubators that consists of (1) edges in ConRouTree, where each chain of lonely nodes is contracted into a single *super incubator*, (2) useful cross edges (which are not pruned after rerouting), and (3) edges in the shortcut spanner for ConRouTree. The resulting *incubator graph* $\mathcal{H}$ has constant degree, hop-diameter $O(\log n)$, lightness $O(\log n)$, and low stretch with respect to the leaf nodes.

The final step is to convert the incubator graph $\mathcal{H}$ (which contains more than n nodes) to a spanner H for the original n-point metric space. One way is to label each node in the net-tree with the corresponding net-point in X. Then, an edge between two nodes induces an edge between their labels. However, this labeling is problematic, due to the hierarchical property of the nets. Although $\mathcal{H}$ has constant degree, if a point in X is used as a label for many nodes, that point will accumulate a large degree. In particular, the point associated with the root node is a net-point at every level, hence this gives rise to a large degree.

The key idea is simple and has been used in [1, 11]: we label each node with a point that is nearby with respect to the relevant distance scale (which means that small stretch will still be preserved), such that each label is used only a constant number of times. This guarantees that the degree of the resulting spanner will be constant. In order to describe the labeling process clearly, we find it convenient to use the incubator-zombie terminology.

Incubators Working with Zombies. Each incubator will receive a label that we refer to as a *zombie*, which is identified with a point in X. The incubator graph $\mathcal{H}$ and the zombies naturally induce a spanner on X: if there is an edge between two incubators, then there is an induced edge between the corresponding zombies. It is left to show how we assign zombies to incubators. Each leaf incubator can be simply assigned its original net-point, as there is a 1-1 correspondence between leaf incubators and points in X. Also, since each internal incubator has at least two children in the contracted incubator tree (ConIncTree), each internal incubator can be assigned a unique nearby zombie from its descendant leaves. (To achieve fault-tolerance, we use a *zombie-climbing* process that involves using both ConIncTree and ConRouTree; we will guarantee that each internal incubator holds up to $k + 1$ nearby zombies, and each point appears as a zombie in $O(k)$ incubators. We provide the details in Section 5.)

Sketch Analysis. Since each incubator contains a nearby zombie, small stretch and lightness can still be preserved. Since the incubator graph $\mathcal{H}$ has constant degree and each point in X can be the identity of at most two zombies, the degree of the resulting spanner H is constant too. The hop-diameter of H is $O(\log n)$, as any spanner path (between leaf nodes) in $\mathcal{H}$ with l hops gives rise to a spanner path in H with at most l hops. Finally, the running time is dominated by the subroutines that our construction uses, which have been shown (in the relevant references) to take $O(n \log n)$ time.

2 Preliminaries

Let (X, d) be an n-point doubling metric, and let $1 \leq k \leq n-2$ be the maximum number of failed nodes allowed. We consider the regime of stretch $1 + \epsilon$, for an arbitrarily small $0 < \epsilon < \frac{1}{2}$. We assume that the minimum inter-point distance of X is 1, and let $\Delta := \max_{u,v \in X} d(u,v)$ be the *diameter* of X.

The ball of radius $r > 0$ centered at x is $B(x,r) := \{u \in X : d(x,u) \leq r\}$. A set $Y \subseteq X$ is called an *r-cover* of X if for any point $x \in X$ there is a point $y \in Y$, with $d(x,y) \leq r$. A set Y is an *r-packing* if for any pair of distinct points $y, y' \in Y$, it holds that $d(y,y') > r$. For $r_1 \geq r_2 > 0$, we say that a set $Y \subseteq X$ is an *(r_1, r_2)-net* for X if Y is both an r_1-cover of X and an r_2-packing. Note that such a net can be constructed by a greedy algorithm. By recursively applying the definition of doubling dimension, we can get the following key fact [12].

Fact 1 (Nets Have Small Size [12]) *Let $R \geq 2r > 0$ and let $Y \subseteq X$ be an r-packing contained in a ball of radius R. Then, $|Y| \leq (\frac{R}{r})^{2\dim}$.*

The minimum spanning tree of a metric space (X,d) is denoted by $\mathsf{MST}(X)$ (or simply MST if (X,d) is clear from the context). Also, we denote by $w(\mathsf{MST})$ the weight of MST. Given a spanner H for (X,d), the *lightness* of H is defined as the ratio of the weight of H to the weight of MST.

Fact 2 (Two Lower Bounds for $w(\mathsf{MST})$) *1. $w(\mathsf{MST}) \geq \Delta$.*
2. Let $S \subseteq X$ be an r-packing, with $r \leq \Delta$. Then, $w(\mathsf{MST}) \geq \frac{1}{2} r \cdot |S|$.

Hierarchical Nets. We consider the hierarchical nets that are used by Gottlieb and Roditty [11]. Let $r_i := 5^i$ and $\ell := \lceil \log_5 \Delta \rceil$. Also, let $\{N_i\}_{i \geq 0}^{\ell}$ be a sequence of hierarchical nets, where $N_0 := X$ and for each $i \geq 1$, N_i is a $(3r_i, r_i)$-net for N_{i-1}. (Observe that N_ℓ contains one point.) As mentioned in [11], this choice of parameters is needed to achieve running time $O(n \log n)$.

Net-tree with Cross Edge Framework. We recap the basic spanner construction [2, 11] using the incubator-zombie terminology.

Incubators. For each level i and each $x \in N_i$, there is a corresponding *incubator* $C = (x, i)$, where x is the *identity* of the incubator and i is its level. For $C_1 = (x_1, i_1)$ and $C_2 = (x_2, i_2)$, define $d(C_1, C_2) := d(x_1, x_2)$; for $C_1 = (x_1, i_1)$ and $x_2 \in X$, define $d(C_1, x_2) := d(x_1, x_2)$.

Incubator Tree. The hierarchical nets induce a tree structure (hereafter, the *incubator tree* $\mathsf{IncTree}$) on the incubators as follows. The only incubator at level ℓ is the root, and for each level $0 \leq i < \ell$, each incubator at level i (hereafter, *level-i incubator*) has a parent at level $i+1$ within distance $3r_{i+1}$. (Recall that N_{i+1} is a $3r_{i+1}$-cover for N_i.) Hence, every descendant of a level-i incubator can reach it by climbing a path of weight at most $\sum_{j \leq i} 3r_j \leq 4r_i$.

Cross Edges. In order to achieve stretch $(1+\epsilon)$, in each level cross edges are added between incubators that are close together with respect to the distance scale at that level. Specifically, for each level $0 \leq i < \ell$, for all $u, v \in N_i$ such that $u \neq v$ and $d(u,v) \leq \gamma r_i$, for some appropriate parameter $\gamma = O(\frac{1}{\epsilon})$, we add a cross edge between the corresponding incubators (u, i) and (v, i) (with weight $d(u,v)$). The basic spanner construction is obtained as the union of the tree ($\mathsf{IncTree}$) edges and the cross edges. The following lemma gives the essence of the cross edge framework; a variant of this lemma appears in [2, Lemma 5.1] and [9]. Since we shall later reroute spanner paths and assign internal incubators with labels (which we refer to as *zombies*), we also give an extended version here.

Lemma 1 (Cross Edge Framework Guarantees Low Stretch). *Consider the cross edge framework as described above.*
(a) *Let $\mu > 0$ be an arbitrary constant. Suppose a graph $\mathcal{H}$ on the incubators contains all cross edges (defined with some appropriate parameter γ depending on μ and ϵ), and for each level $i \geq 1$, each level-$(i-1)$ incubator is connected via a tree edge to some level-i incubator within distance μr_i. Then, $\mathcal{H}$ contains a $(1+\epsilon)$-spanner path $P_{u,v}$ for each $u, v \in X$, obtained by climbing up from the leaf incubators corresponding to u and v to some level-j ancestors u' and v', respectively, where u' and v' are connected by a cross edge and $r_j = O(\epsilon) \cdot d(u,v)$.*
(b) *In the above graph $\mathcal{H}$, if for each level $i \geq 1$, each level-i incubator (u,i) is labeled with a point $\widehat{u}$ such that $d(u, \widehat{u}) = O(r_i)$, then the path $\widehat{P_{u,v}}$ induced by the above path $P_{u,v}$ and the labels is a $(1+O(\epsilon))$-spanner path.*

Lonely Incubators. An incubator is called *lonely* if it has exactly one child incubator (which has the same identity as the parent); otherwise it is *non-lonely.* Observe that the leaf incubators have no children and are non-lonely. For efficiency reasons, a long chain of lonely incubators will be represented implicitly; implementation details can be found in [11]. As we shall later see, for the *zombie-climbing* process (described in Section 5) to succeed we need the property that each internal incubator has at least two children. Observe that at the bottom of a chain $\mathcal{C}$ of lonely incubators is a non-lonely incubator C with the same identity. We shall later contract a chain $\mathcal{C}$ of lonely incubators (together with the non-lonely incubator C at the bottom) into a *super incubator.*
Running Time. All the subroutines that our construction uses have been shown (in the relevant references) to take $O(n \log n)$ running time. Hence, we disregard the running time analysis, except for places which require clarification.

Challenges Ahead. Lemma 1 can be used to achieve low stretch. As lonely incubators need to be contracted, the next issue is that many cross edges will be inherited by the super incubator, which may explode the degree. To overcome this obstacle, in Section 3 we use Gottlieb and Roditty's technique [11] to reroute spanner paths and prune redundant cross edges. In Section 4 we show that hop-diameter $O(\log n)$ (and lightness $O(\log n)$ too) can be achieved by applying Solomon and Elkin's shortcut spanner [15] to all subtrees at sufficiently small distance scales (less than $\frac{\Delta}{n}$). In Section 5 we describe a zombie-climbing process, which converts a graph on the incubators to a k-fault tolerant spanner on X with all the desired properties, thereby completing the proof of Theorem 1.

3 Reducing Degree via Gottlieb-Roditty's Spanner

By Fact 1, each internal incubator has at most $O(1)^{O(\dim)}$ children in $\mathsf{IncTree}$, and each incubator is incident on at most $\epsilon^{-O(\dim)}$ cross edges. However, the problem that arises is that when a chain of lonely incubators is contracted, the cross edges that are incident on the corresponding super incubator may explode its degree. We employ the *parent replacement* technique due to Gottlieb and Roditty [11] to reroute spanner paths and make some cross edges *redundant*. Our procedure below is a simple modification of subroutines that appear in [11], and hence can be implemented within the same running time $O(n \log n)$.

Routing Tree. We carry out the parent replacement procedure by constructing a *routing tree* ($\mathsf{RouTree}$), which has the same leaf incubators as $\mathsf{IncTree}$; moreover, each level-$(i-1)$ child incubator is connected to a level-i parent incubator with an edge of possibly heavier weight at most $5r_i$ (as opposed to weight at most $3r_i$ as in $\mathsf{IncTree}$). Consider a non-root incubator C in the (uncontracted) $\mathsf{IncTree}$. The parent incubator of C in $\mathsf{RouTree}$ is determined by the following rules.
(1) If either the parent C' of C in $\mathsf{IncTree}$ or the parent of C' is non-lonely, then C will have the same parent C' in $\mathsf{RouTree}$.
(2) For a chain of at least two lonely incubators, we start from the bottom incubator $C_i = (x, i)$ (which is non-lonely) at some level i. If the parent C_{i+1} of $C_i = (x, i)$ in $\mathsf{IncTree}$ is lonely and the parent of C_{i+1} is also lonely, then we try to find a new parent for C_i. Specifically, if there is some point $w \in N_{i+1} \setminus \{x\}$ such that $d(x, w) \leq 5r_{i+1}$ (if there is more than one such point w, we can pick one arbitrarily), then a non-lonely *adopting parent* is found as follows.
(a) If the incubator $\widehat{C}_{i+1} = (w, i+1)$ is non-lonely, then $\widehat{C}_{i+1}$ is designated as the adopting parent of C_i (which means that $\widehat{C}_{i+1}$ will be C_i's parent in $\mathsf{RouTree}$). Similarly, C_i is designated as an *adopted child* of $\widehat{C}_{i+1}$.
(b) If the incubator $\widehat{C}_{i+1}$ is lonely, then it does not adopt C_i. However, we shall see in Lemma 2 that the parent $\widehat{C}_{i+2}$ of $\widehat{C}_{i+1}$ cannot be lonely. Moreover, it is close enough to C_{i+1}, namely, $d(C_{i+1}, \widehat{C}_{i+2}) \leq 5r_{i+2}$. In this case $\widehat{C}_{i+2}$ will adopt C_{i+1} (and will be its parent in $\mathsf{RouTree}$), and C_{i+1} will remain C_i's parent.

Observe that once an adopting parent is found, there is no need to find adopting parents for the rest of the lonely ancestors in the chain, because these lonely ancestors will not adopt, and so they will not be used for routing.
If there is no such nearby point $w \in N_{i+1} \setminus \{x\}$ for $C_i = (x, i)$, then C_i's parent in $\mathsf{RouTree}$ remains C_{i+1}, and we continue to climb up the chain.

Lemma 2 (Lonely Incubators Need Not Adopt). *[Proof in full version] Suppose that an incubator $C_i = (x, i)$ has at least two lonely ancestors (excluding C_i) and there exists a point $w \in N_{i+1} \setminus \{x\}$, such that $d(x, w) \leq 5r_{i+1}$ and the incubator $\widehat{C}_{i+1} = (w, i+1)$ is lonely. Then, the parent $\widehat{C}_{i+2} = (u, i+2)$ of $\widehat{C}_{i+1}$ in* $\mathsf{IncTree}$ *is non-lonely; moreover, $d(x, u) \leq 5r_{i+2}$.*

Rerouting Spanner Paths. When we wish to find a spanner path between u and v, we use $\mathsf{RouTree}$ to climb to the corresponding ancestor incubators (from an appropriate level) which are connected by a cross edge. Observe that using $\mathsf{RouTree}$, it is still possible to climb from a level-i incubator to a level-$(i+1)$ incubator that is within distance $O(r_{i+1})$ from it. Hence, by Lemma 1, stretch $1 + \epsilon$ can still be preserved.
Degree Analysis. Notice that only non-lonely incubators can adopt, and by Fact 1, each incubator can have only $O(1)^{O(\dim)}$ adopted child incubators. Hence,

the degree of RouTree is $O(1)^{O(\dim)}$. Consider a chain of lonely incubators with identity x. If some incubator $C = (x, i)$ has found an adopting parent, then all lonely ancestors of C in the chain will not be used for routing (since a lonely incubator cannot adopt). Thus the cross edges incident on those unused lonely ancestors are redundant. Next, we show that the number of useful cross edges accumulated by a chain of lonely incubators (until an adoption occurs) is small.

Lemma 3 (No Nearby Net-Points Implies Few Cross Edges). *[Proof in full version] Suppose that $x \in N_i$ and all other points in N_i are at distance more than $5r_i$ away (i.e., no adopting parent is found for any child of (x, i)). Then, there are at most $O(\gamma)^{O(\dim)} = \epsilon^{-O(\dim)}$ cross edges with level at most i connecting incubators with identity x and non-descendants of (x, i).*

Pruning Cross Edges and Routing Tree. After the lowest level incubator in a chain of lonely incubators finds an adopting parent, redundant cross edges incident on the ancestors of the adopted incubator can be pruned. By Lemma 3, no matter if an adopting parent is found for a chain, at most $\epsilon^{-O(\dim)}$ remaining cross edges are incident on incubators in the chain for the following reason.
• If no adopting parent is found for a chain, Lemma 3 can be applied to the second lonely incubator (if any) from the top of the chain. Since the top incubator has only $\epsilon^{-O(\dim)}$ cross edges, the entire chain has $\epsilon^{-O(\dim)}$ incident cross edges.
• If some point $w \in N_{i+1} \setminus \{x\}$ is found for an incubator $C_i = (x, i)$ in the parent replacement process described above, Lemma 3 can be applied to C_i (unless C_i is the bottom incubator in the chain, and then we do not need to apply the lemma). Since either C_i or its parent will be adopted (and the cross edges incident on the ancestors of the adopted child are redundant, and will be pruned), it follows that the entire chain has $\epsilon^{-O(\dim)}$ remaining incident cross edges.

For efficiency reasons, observe that we can first build RouTree and add cross edges only for incubators that are actually used for routing. In other words, we do not have to add redundant cross edges that will be pruned later.

Contraction Phase. After finishing the construction of RouTree, we start the contraction phase, which involves contracting all chains of lonely incubators. The above argument implies that the number of cross edges incident on a super incubator is at most $\epsilon^{-O(\dim)}$. Also, since lonely incubators cannot adopt, the degree of the routing tree cannot increase.
• If no adopting parent is found for a chain, the chain will be contracted to a super incubator C, which has the same parent $\overline{C}$ (that itself may be a super incubator corresponding to a contracted chain) in the *contracted incubator tree* (ConIncTree) and the *contracted routing tree* (ConRouTree).
• If an adopting parent $\widehat{C}$ is found for a chain, the chain will be contracted to a super incubator C, whose parent $\overline{C}$ in ConRouTree is different from its parent $\tilde{C}$ in ConIncTree; either one among $\overline{C}$ and $\tilde{C}$ can be a super incubator.

Multiple edges are removed from the resulting multi-graph, keeping just the edge of minimum level between any pair of incident (super) incubators.

Corollary 1 (Constant Degree). ConRouTree *has degree $O(1)^{O(\dim)}$, and each incubator (and also super incubator) has $\epsilon^{-O(\dim)}$ cross edges.*

4 Achieving Small Hop-Diameter and Lightness

Consider $\mathsf{ConRouTree}$ constructed in Section 3. Observe that if the maximum inter-point distance Δ is large enough (exponential in n), the hop-diameter will be as large as $\Theta(n)$. In this section we add edges to shortcut $\mathsf{ConRouTree}$, such that for each level i, any leaf incubator can reach some level-i incubator in $O(\log n)$ hops and within distance $O(r_i)$; this guarantees that the hop-diameter is $O(\log n)$. We also make sure that the lightness will be in check.

Theorem 2 (Spanner Shortcut [15]). *Let T be a tree (whose edges have positive weights) with n nodes and degree $\mathsf{deg}(T)$. For the tree metric induced by the shortest-path distances in T, a 1-spanner J with $O(n)$ edges, degree at most $\mathsf{deg}(T)+4$, and hop-diameter $O(\log n)$ can be constructed in $O(n \log n)$ time.*

Levels of Super Incubators and Edges. Technically, a super incubator is of the same level as the non-lonely incubator at the bottom of the corresponding chain. The level of an edge before the contractions is defined as the maximum level of its endpoint incubators, and its level after the contractions remains the same.
Shortcut the Low Levels of $\mathsf{ConRouTree}$. Let $\widehat{r} = \frac{\Delta}{n}$, and define $\sigma := \lfloor \log_5 \widehat{r} \rfloor$. Observe that the number of levels above σ is $O(\log n)$. We shortcut all maximal subtrees rooted at level at most σ in $\mathsf{ConRouTree}$, i.e., the root of each such subtree is at level at most σ, but its parent is at level greater than σ. Each such subtree is shortcut via the 1-spanner that is given by Theorem 2. Notice that this shortcut procedure adds edges that enable going from each leaf incubator to any of its ancestor incubators in $\mathsf{ConRouTree}$ tree in $O(\log n)$ hops. This implies that for any $u, v \in X$, there is a spanner path between the corresponding leaf incubators with $O(\log n)$ hops and weight at most $(1+\epsilon) \cdot d(u,v)$. Moreover, the shortcut procedure increases the degree of each incubator by at most four.
Large vs Small Scales. We analyze the weight of the spanner by considering the weight contribution from large-scale edges and small-scale edges separately. An edge has a *small scale* if its level is at most σ (we use the convention that a shortcut edge has a small scale); otherwise, it has a *large scale*. We remark that the following lemma remains valid if we increase the weight of each level-i edge by $O(r_i)$; this observation will be used later (in Section 5) when we apply our labeling procedure.

Lemma 4 (Edges are Light). *[Proof in full version] The total weight of all large-scale edges is $\epsilon^{-O(\mathrm{dim})} \cdot \log n \cdot w(\mathsf{MST})$, and the total weight of all small-scale edges is $\epsilon^{-O(\mathrm{dim})} \cdot w(\mathsf{MST})$.*

Incubator Graph $\mathcal{H}$. To summarize, we build an *incubator graph $\mathcal{H}$* on the incubators (and super incubators) that consists of (1) the edges in $\mathsf{ConRouTree}$, (2) useful cross edges that remain after pruning, and (3) edges used to shortcut the low levels (at most σ) of $\mathsf{ConRouTree}$. The incubator graph $\mathcal{H}$ has degree $\epsilon^{-O(\mathrm{dim})}$. Moreover, for any $u, v \in X$, there is a path in $\mathcal{H}$ between the corresponding leaf incubators with $O(\log n)$ hops and weight at most $(1+\epsilon) \cdot d(u,v)$.

Also, Lemma 4 implies that it has weight $\epsilon^{-O(\dim)} \cdot \log n \cdot w(\mathsf{MST})$. Finally, it is easy to see that $\mathcal{H}$ can be implemented within $\epsilon^{-O(\dim)} \cdot n \log n$ time.

In other words, the incubator graph $\mathcal{H}$ has almost all the desired properties, except that each point in X may be the identity of many incubators (and super incubators). Moreover, we have not considered fault tolerance so far. We will address these issues in Section 5.

5 Incubators Working with Zombies: Fault Tolerance

The incubator graph $\mathcal{H}$ defined at the end of Section 4 achieves all the desired properties, except that the same point may be the identity of many incubators. Moreover, there is a 1-1 correspondence between the leaf incubators and the points in X. In this section we show how to convert $\mathcal{H}$ into a k-fault tolerant spanner H for X that satisfies all the desired properties. To this end we devise a simple labeling procedure (that we refer to below as the *zombie-climbing procedure*), which is convenient to describe via the incubator-zombie terminology.

Zombies. A *zombie* is identified by a point $x \in X$, which is the *identity* of the zombie. When the context is clear, we do not distinguish between a zombie and its identity. After the zombie-climbing procedure is finished, each leaf incubator will contain a zombie whose identity is the same as the leaf's identity, and each internal incubator will contain up to $k + 1$ zombies with distinct identities.

Induced Spanner on X. The incubator graph $\mathcal{H}$ together with the zombies induce a spanner H on X in a natural way: two points u and v are neighbors in H if there are zombies with identities u and v residing in neighboring incubators in $\mathcal{H}$. Hence, it suffices to describe how zombies are assigned to incubators.

The zombies are climbing. . . We assign zombies to incubators in two stages. In the first stage, we use $\mathsf{ConIncTree}$ to assign a single *host* zombie to each incubator; in the second stage, for each internal incubator, we use $\mathsf{ConRouTree}$ to collect up to k additional *guest* zombies with distinct identities, which are also different from the identity of the host zombie.

First Stage. Consider $\mathsf{ConIncTree}$, and note that each internal incubator has at least two children. Each incubator is assigned a host zombie as follows. A leaf incubator C creates two zombies with the same identity as itself; one stays in C as its host zombie and the other climbs to C's parent. An internal incubator $\widehat{C}$ receives exactly one zombie from each of its (at least two) children. One of these zombies stays in $\widehat{C}$ as its host zombie, and another one (chosen arbitrarily) climbs to $\widehat{C}$'s parent incubator (if any); extra zombies (which do not become host zombies) are discarded. Since there are $O(n)$ incubators, this procedure takes $O(n)$ time. Observe that each point can appear as the identity of at most two host zombies: once in a leaf incubator, and at most once in an internal incubator.

Second Stage. We use $\mathsf{ConRouTree}$ in this step. Each internal incubator C collects up to k guest zombies with distinct identities from the host zombies of C's descendants (in $\mathsf{ConRouTree}$) using a bounded breadth-first search. Specifically, when a descendant incubator $\widetilde{C}$ is visited, if its host zombie $\tilde{z}$ is different from the host zombie of C and from all the guest zombies already collected by C, then $\tilde{z}$ will be collected as one of C's guest zombies. The breadth-first search terminates once C has collected k distinct guest zombies or when all C's descendants in $\mathsf{ConRouTree}$ have been visited. Since for each breadth-first search, $O(k)$ incubators are visited (as each point can appear as the identity of at most two host zombies), the second stage can be implemented within $O(kn)$ time.

Lemma 5 (Each Point Appears as $O(k)$ Zombies). *[Proof in full version] Each point can be the identity of a zombie in at most $2k+2$ incubators (as either a host or a guest).*

Fault tolerance. Recall that in the cross edge framework, the low-stretch spanner path between any two points u and v is obtained as follows. We start from the two leaf incubators corresponding to u and v, and climb to the ancestor incubators (according to $\mathsf{ConRouTree}$ in our case) in some appropriate level, where a cross edge is guaranteed to exist. Observe that only incubators with the same identity will be contracted, and so this does not change the stretch of the spanner path. The two following lemmas show that for any functioning point u, each ancestor of the leaf incubator corresponding to u (in $\mathsf{ConRouTree}$) contains at least one nearby functioning zombie. Consequently, fault-tolerance is achieved.

Lemma 6 (Every Ancestor in $\mathsf{ConRouTree}$ Has a Functioning Zombie). *[Proof in full version] Suppose that at most k points fail. Let u be an arbitrary functioning point, and let C_u be the leaf incubator corresponding to u. Then, every ancestor of C_u in $\mathsf{ConRouTree}$ contains at least one functioning zombie.*

Lemma 7 (Incubators Contain Nearby Zombies). *[Proof in full version] If an incubator $C = (x, i)$ contains a zombie z, then $d(x, z) = O(r_i)$.*

Tying Up Everything Together – Completing the Proof of Theorem 1
Stretch and Fault-Tolerance. Lemmas 6 and 7 state that each incubator contains a functioning zombie that is nearby, which implies that a level-i incubator edge will induce a functioning zombie edge with weight that is greater by at most an additive factor of $O(r_i)$. The second assertion of Lemma 1 implies that the stretch of the resulting spanner is $1 + O(\epsilon)$; we can achieve stretch $(1 + \epsilon)$ by rescaling γ (and other parameters) by an appropriate constant.
Degree. Since the incubator graph $\mathcal{H}$ has degree $\epsilon^{-O(\dim)}$ and each incubator contains at most $k + 1$ zombies, it follows that each occurrence of a point as the identity of some zombie will incur a degree of at most $\epsilon^{-O(\dim)} \cdot k$. By Lemma 5, each point can be the identity of at most $2k + 2$ zombies, which implies that the degree of H is at most $\epsilon^{-O(\dim)} \cdot k^2$.
Hop-Diameter. Observe that any spanner path in $\mathcal{H}$ (between leaf nodes) with l hops induces a functioning spanner path in H with at most l hops. Hence the hop-diameter of H is $O(\log n)$.
Lightness. As already mentioned, in the proof of Lemma 4, the upper bound still holds if we add $O(r_i)$ to the weight of each level-i incubator edge. Moreover, as only zombies within distance $O(r_i)$ are assigned to a level-i incubator, it follows that each level-i zombie edge has weight at most an additive factor $O(r_i)$

greater than that of the inducing incubator edge. Since every incubator edge induces at most $O(k^2)$ zombie edges, we conclude that the lightness of H is $\epsilon^{-O(\dim)} \cdot k^2 \log n$.

Running Time. As mentioned in Sections 3 and 4, our construction uses subroutines from Gottlieb-Roditty's spanner [11] and Elkin-Solomon's shortcut spanner [15], which have running time at most $\epsilon^{-O(\dim)} \cdot n \log n$. Also, the zombie-climbing procedure takes time $O(kn)$. Finally, observe that there are $\epsilon^{-O(\dim)} \cdot n$ edges in $\mathcal{H}$, each of which induces $O(k^2)$ zombie edges; hence, transforming the incubator graph $\mathcal{H}$ into the ultimate spanner H takes $\epsilon^{-O(\dim)} \cdot k^2 n$ time.

Acknowledgments. The fourth-named author is grateful to Michael Elkin and Michiel Smid for helpful discussions.

References

1. S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. H. M. Smid. Euclidean spanners: short, thin, and lanky. In *STOC*, pages 489–498, 1995.
2. H. T.-H. Chan, A. Gupta, B. M. Maggs, and S. Zhou. On hierarchical routing in doubling metrics. In *SODA*, pages 762–771, 2005.
3. T.-H. H. Chan and A. Gupta. Small hop-diameter sparse spanners for doubling metrics. *Discrete & Computational Geometry*, 41(1):28–44, 2009.
4. T.-H. H. Chan, M. Li, and L. Ning. Incubators vs zombies: Fault-tolerant, short, thin and lanky spanners for doubling metrics. *CoRR*, abs/1207.0892, 2012.
5. T.-H. H. Chan, M. Li, and L. Ning. Sparse fault-tolerant spanners for doubling metrics with bounded hop-diameter or degree. In *ICALP*, pages 182–193, 2012.
6. P. Chew. There is a planar graph almost as good as the complete graph. In *SoCG*, pages 169–177, 1986.
7. Y. Dinitz, M. Elkin, and S. Solomon. Shallow-low-light trees, and tight lower bounds for Euclidean spanners. In *FOCS*, pages 519–528, 2008.
8. M. Elkin and S. Solomon. Optimal Euclidean spanners: really short, thin and lanky. In *STOC*, 2013 (to appear).
9. J. Gao, L. J. Guibas, and A. Nguyen. Deformable spanners and applications. In *SoCG*, pages 190–199, 2004.
10. L.-A. Gottlieb and L. Roditty. Improved algorithms for fully dynamic geometric spanners and geometric routing. In *SODA*, pages 591–600, 2008.
11. L.-A. Gottlieb and L. Roditty. An optimal dynamic spanner for doubling metric spaces. In *ESA*, pages 478–489, 2008.
12. A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS*, pages 534–543, 2003.
13. L. Roditty. Fully dynamic geometric spanners. In *SoCG*, pages 373–380, 2007.
14. S. Solomon. Fault-tolerant spanners for doubling metrics: Better and simpler. *CoRR*, abs/1207.7040, 2012.
15. S. Solomon and M. Elkin. Balancing degree, diameter and weight in Euclidean spanners. In *ESA*, pages 48–59, 2010.