Fully Dynamic Algorithms for Graph Spanners via Low-Diameter Router Decomposition*

Julia Chuzhoy† Merav Parter ‡

February 16, 2026

Abstract

A t-spanner of an undirected n-vertex graph G is a sparse subgraph H of G that preserves all pairwise distances between its vertices to within multiplicative factor t, also called the *stretch*. Spanners play an important role in the design of efficient algorithms for distance-based graph optimization problems, as they allow one to sparsify the graph, while approximately preserving all distances. It is well known that any n-vertex graph admits a $(2k-1)$-spanner with $O(n^{1+1/k})$ edges, and that this stretch-size tradeoff is optimal assuming the Erdös Girth Conjecture. In this paper we investigate the problem of efficiently maintaining spanners in the fully dynamic setting with an adaptive adversary. Despite a long and intensive line of research, this problem is still poorly understood: for example, no algorithm achieving a sublogarithmic stretch, with a sublinear in n update time, and a strongly subquadratic in n bound on the size of the spanner is currently known in this setting. One of our main results is a *deterministic* (and therefore, adaptive-adversary) algorithm, that, for any $512 \leq k \leq (\log n)^{1/49}$ and $1/k \leq \delta \leq 1/400$, maintains a spanner H of a fully dynamic graph with stretch $\mathrm{poly}(k) \cdot 2^{O(1/\delta^6)}$ and size $|E(H)| \leq O(n^{1+O(1/k)})$, with worst-case update time $n^{O(\delta)}$ and recourse $n^{O(1/k)}$.

Our algorithm relies on a new technical tool that we develop, and that we believe to be of independent interest, called *low-diameter router decomposition*. Specifically, we design a deterministic algorithm that maintains a decomposition of a fully dynamic graph into edge-disjoint clusters with bounded vertex overlap, where each cluster C is guaranteed to be a bounded-diameter router, meaning that any reasonable multicommodity demand over the vertices of C can be routed along short paths and with low congestion inside C. A similar graph decomposition notion was introduced by [Haeupler et al., STOC 2022] and recently strengthened by [Haeupler et al., FOCS 2024]; the latter result was already used to obtain fast algorithms for multicommodity flows [Haeupler et al., STOC 2024] and dynamic distance oracles [Haeupler et al., FOCS 2024]. However, in contrast to these and other prior works, the decomposition that our algorithm maintains is guaranteed to be *proper*, in the sense that the routing paths between the pairs of vertices of each cluster C are contained inside C (rather than in the entire graph G). Additionally, our algorithm maintains, for each cluster C of the decomposition, a subgraph $C' \subseteq C$ of a prescribed density, that also has strong routing properties.

We show additional applications of our low-diameter router decomposition, by obtaining new deterministic dynamic algorithms for fault-tolerant spanners and low-congestion spanners. Several of these applications crucially rely on the fact that our router decomposition is proper.

*Preliminary version appeared in SODA 2025

†Toyota Technological Institute at Chicago. Email: cjulia@ttic.edu. Supported in part by NSF grant CCF-2402283 and NSF HDR TRIPODS award 2216899.

‡Weizmann Institute of Science, Israel. Email: merav.parter@weizmann.ac.il. Supported partially by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 949083.

Contents

1 Introduction

A spanner of a graph G is a sparse subgraph of G, that approximately preserves all pairwise distances between its vertices. The notion of spanners naturally falls within the broader class of graph-theoretic objects called *graph sparsifiers*. A sparsifier of a given graph G is a sparse subgraph of G, that approximately preserves some central properties of G, such as, for example, cut values, flows, or, in this case, distances. A natural and powerful paradigm for designing fast algorithms for graph optimization problems is to first compute a sparsifier H of the input graph G, that preserves properties of G relevant to the problem, and then solve the problem on the much sparser graph H. It is then not surprising that various types of sparsifiers increasingly play a central role in the design of fast algorithms for various combinatorial optimization problems, in static, dynamic, and other settings. Since their introduction by Peleg and Schaffer [PS89], spanners have been studied extensively in various settings (see, e.g. [PU87, EP01, EZ04, Bas06, BS07, Bas08, DGPV08, BKS12, GK18, DFKL21]), and have been used in a wide range of applications: from distance oracles [TZ01], shortest path computation [BK06] and routing schemes [PU89] to spectral sparsification [KP12].

Formally, given an n-vertex graph G with non-negative edge lengths, a k-spanner for G is a subgraph $H \subseteq G$, such that, for every pair u, v of vertices of G, $\mathsf{dist}_H(u,v) \leq k \cdot \mathsf{dist}_G(u,v)$. In addition to the stretch parameter k, a central measure of the quality of the spanner H is its size $|E(H)|$. It is well known that any n-vertex graph admits a $(2k-1)$-spanner with $O(n^{1+1/k})$ edges, and this tradeoff is tight under the Erdös Girth Conjecture [Erd64]. In their influential paper, Baswana and Sen [BS07] presented a randomized algorithm for computing a $(2k-1)$-spanner of a static n-vertex and m-edge graph G, that contains $O(k \cdot n^{1+1/k})$ edges, in expected time $O(km)$. Roditty, Thorup and Zwick [RTZ05] derandomized this construction, while achieving similar parameters.

In this paper we focus on the problem of efficiently maintaining a spanner of a fully dynamic graph G, that is, a graph undergoing an online sequence of edge insertions and deletions. As usual in the area of dynamic algorithms, we distinguish between the *oblivious-adversary* and the *adaptive-adversary* settings. In the case of oblivious adversary, it is assumed that the sequence of updates to the input graph G is fixed in advance, and may not depend on the algorithm's behavior and random choices. In contrast, in the *adaptive-adversary* setting, each update to the graph may depend on the algorithm's past behavior and inner state arbitrarily. It is often the case that the oblivious adversary assumption considerably limits the applicability of a dynamic algorithm. As an example, one common use of such algorithms is in speeding up algorithms for static problems, which typically requires that the dynamic algorithm can withstand an adaptive adversary. A large number of breakthrough result in recent years, that obtained fast algorithms for central graph optimization problems in the static setting, crucially rely on dynamic algorithms that can withstand an adaptive adversary. With this motivation in mind, we focus in this paper on the adaptive-adversary setting. We note that deterministic algorithms are always guaranteed to work against an adaptive adversary, and as such they are especially desirable. For brevity, we will refer to algorithms that can withstand oblivious or adaptive adversaries as *oblivious-update* or *adaptive-update* algorithms, respectively.

Dynamic Spanners

Dynamic algorithms for graph spanners have been studied extensively, since the pioneering work of Ausiello et al. [ADF$^+$07]. This line of research can be partitioned into three main eras.

Era 1: Optimal Spanners with Low Amortized Update Time. Earlier work studied the problem of maintaining spanners of fully dynamic graphs with near-optimal tradeoff between the stretch and the size of the spanner, with the focus on optimizing the *amortized* update time, that

is, the **average** amount of time that the algorithm spends in order to process each update to the input graph G. Ausiello, Fraciosa and Italiano [ADF$^+$07] provided deterministic dynamic algorithms for spanners with stretch 3 and 5, with amortized update time proportional to the maximum vertex degree in the graph. Elkin [Elk07] presented a randomized oblivious-update algorithm for maintaining $(2k-1)$-spanners for arbitrary values of k, with amortized update time $O(mn^{-1/k})$, where m is the number of edges in the initial graph; note that, for sufficiently dense graphs, this amortized update time may be superlinear in n. This line of work culminated with the result of Baswana, Khurana, and Sarkar [BKS12], that achieved expected amortized update time $\min\left\{O(k^2\log^2 n), 2^{O(k)}\right\}$ against an oblivious adversary with spanner size $O(kn^{1+1/k}\log n)$. Forster and Goranci [FG19] later improved the size and the update time by factor k. The main drawbacks of these algorithms are that (i) their worst-case update time may be as high as $\Omega(n)$; and (ii) except for the work of [ADF$^+$07], all results cited above work in the oblivious-adversary setting only.

Era 2: Low Worst-Case Update Time against an Oblivious Adversary. Bodwin and Krinninger [BK16] designed the first algorithms for maintaining spanners of dynamic graphs with worst-case update time that is sublinear in n, for the special case of stretch values 3 and 5. Specifically, they design randomized oblivious-update algorithms that maintain 3-spanners with $\widetilde{O}(n^{3/2})$ edges and worst-case update time $\widetilde{O}(n^{3/4})$, and 5-spanners with $\widetilde{O}(n^{4/3})$ edges and worst-case update time $\widetilde{O}(n^{5/9})$. Later, Bernstein, Forster and Henzinger [BFH19] significantly narrowed the gap between amortized and worst-case update time guarantees by "deamortizing" the result of Baswana et al. [BKS12]. Their algorithms are based on a general technique for randomized data structures, that converts amortized update time guarantees to worst-case ones. This approach yields a randomized oblivious-update algorithm for dynamic k-spanners with $k = O(1)$, with spanner size $O(nk^{1+1/k})$ and worst-case expected update time $2^{O(k)}\log^3 n$, nearly matching the amortized update time bound of [BKS12].

A key limitation of the algorithm of [BFH19], as well as all of the above-mentioned previous dynamic algorithms for stretch values $k > 5$, is that they are randomized, and more crucially, they only work against an *oblivious adversary*. As mentioned above, this significantly limits the applicability of such algorithms, and addressing this drawback became the next major goal in this area.

Era 3: The Quest for Adaptive-Update Algorithms. Bernstein et al. [BvdBG$^+$22] provided the first dynamic algorithms for maintaining spanners that can withstand an adaptive adversary for stretch values $k > 5$. Their algorithm is randomized, and with high probability maintains a spanner with stretch $O(\text{poly}\log n)$ and size $\widetilde{O}(n)$, with $O(\text{poly}\log n)$ *amortized* update time. Their algorithm can also be adapted to obtain worst-case update time at most $n^{o(1)}$, at the cost of increasing the stretch bound to $n^{o(1)}$. Their approach relies on expander decomposition, and as such is unlikely to lead to the construction of spanners with sublogarithmc stretch, as was also noted in [BSS22]. Lastly, the recent work of [CKL$^+$22, vdBCK$^+$23] provided a deterministic algorithm for maintaining an $n^{o(1)}$-spanner of a fully dynamic graph, whose size is $n^{1+o(1)}$, and worst-case update time bound is at least as high as $n^{o(1)}$ times the maximum vertex degree in the graph. Their algorithm has several other important properties, such as, for example, it has a low recourse, and it maintains a low-congestion embedding of G into the sparsifier, which we discuss below. This algorithm, in turn, serves as one of the main subroutines used in their breakthrough almost-linear time algorithm for min-cost flow, and it was also used in a recent work of [KMPG24] for obtaining low-congestion vertex sparsifiers and algorithms for dynamic All-Pairs Shortest Paths. We provide a summary of known results for fully-dynamic spanners in Table 1. To summarize, to the best of our knowledge, the following fundamental question still remains open:

Question 1 *Is there an adaptive-update algorithm for maintaining a spanner of an n-vertex dynamic graph with any stretch value $k < o(\log n)$, whose size is at most $n^{2-\epsilon}$ for any constant ϵ, and worst-case (or even amortized) update time is $o(n)$?*

Dynamic Spanners with Small Recourse. An additional desired property of dynamic algorithms that maintain spanners (as well as other types of sparsifiers) is to ensure that the resulting spanner changes slowly, as the input graph G undergoes updates. The reason is that, as mentioned already, a commonly used paradigm is to apply algorithms for various distance-based problems to the spanner rather than the original graph, in order to obtain faster running time guarantees. If, however, following an update to the input graph G, the resulting spanner H may undergo a large number of changes, then it is unlikely that such an algorithm may provide low worst-case update time guarantees. This naturally leads to the notion of a *recourse*: the largest number of edge insertions and deletions that the spanner H may undergo after each update to the input graph G. This measure naturally plays an important role in dynamic algorithms, see e.g. [CHK16, BC18, AOSS19, CZ19]. Bhattacharya, Saranurak and Sukprasert [BSS22] presented a deterministic (and therefore, adaptive-update) algorithm for maintaining $(2k-1)$ spanners with near-optimal size, $O(\log n)$ amortized recourse and $\mathrm{poly}(n)$ worst-case update time. The algorithm of [CKL$^+$22, vdBCK$^+$23] mentioned above also has recourse at most $n^{o(1)}$.

Our first main result answers Question 1 in the affirmative, by providing an algorithm for maintaining dynamic spanners that also has a low recourse:

Theorem 1.1 (Dynamic Deterministic Spanners, Informal) *There is a deterministic algorithm, whose input is an n-vertex simple undirected graph G with integral poly-bounded edge lengths, a stretch parameter $512 \leq k \leq (\log n)^{1/49}$, and an additional parameter $1/k \leq \delta \leq 1/400$. Graph G initially has no edges, and it undergoes an online sequence of edge insertions and deletions. The algorithm maintains a spanner H of G with stretch $k^{O(1)} \cdot 2^{O(1/\delta^6)}$ and $|E(H)| \leq O(n^{1+O(1/k)})$. The worst-case update time of the algorithm is $n^{O(\delta)}$, and its recourse is $n^{O(1/k)}$. The algorithm can also be extended to maintain fault-tolerant spanners, connectivity certificates, and low-congestion spanners.*

At the heart of our approach is a new algorithmic tool that we believe is of independent interest, that we call a *low-diameter router decomposition*, or just a *router decomposition* for brevity. Given an n-vertex graph G and a parameter k, a router decomposition of G consists of a collection $\mathcal{C}$ of edge-disjoint *router* subgraphs of G, that we call clusters, each of which has a small diameter, such that $\sum_{C \in \mathcal{C}} |V(C)| \leq n^{1+O(1/k)}$ holds, and the vast majority of the edges of G lie in some cluster of $\mathcal{C}$. The notion of routers was introduced in a very recent work of [HHT24, HHL$^+$24], as an object that is closely related to length-constrained expanders of [HRG22, HHG22]. Routers are also closely related to well-connected graphs introduced in [Chu23]; we discuss this connection in more detail in Section 2.4. Intuitively, a graph G is a router, if any "reasonable" multicommodity demand can be routed in G via short paths (the length is typically sublogarithmic in n), and with low congestion. One example of a reasonable demand is where the total demand on every vertex is bounded by its degree.

Before we continue, we need to define the notion of *demands* and their *routing*. A *demand* $\mathcal{D}$ in a graph G consists of a collection Π of pairs of vertices of G, and, for every pair $(a,b) \in \Pi$, a demand value $D(a,b) > 0$. For a vertex $v \in V(G)$, the *demand on* v is the sum of demands $D(u,v)$ over all pairs $(u,v) \in \Pi$ containing v. We say that the demand $\mathcal{D}$ is *unit*, if the demand on every vertex v is bounded by its degree, and we say that it is Δ-bounded, for some value $\Delta > 0$, if the demand on

Stretch	Spanner Size	Update Time	[W/A]	Deterministic?	Citation
$(2k-1)$	$O\left(k \cdot n^{1+1/k}\right)$	$O(m \cdot n^{-1/k})$	[A]	Oblivious	[Elk07]
$(2k-1)$	$O\left(kn^{1+1/k} \log n\right)$	$\min\{O(k^2 \log^2 n), 2^{O(k)}\}$	[A]	Oblivious	[BKS12]
$(2k-1)$	$O\left(n^{1+1/k} \log n\right)$	$O(k \log^2 n)$	[A]	Oblivious	[FG19]
$(2k-1)$	$O\left(k \cdot n^{1+1/k}\right)$	$2^{O(k)} \cdot \log^3 n$	[W]	Oblivious	[BFH19]
$\text{poly} \log n$	$\widetilde{O}(n)$	$\text{poly} \log n$	[A]	Adaptive	[BvdBG$^+$22]
$n^{o(1)}$	$O\left(n^{1+o(1)}\right)$	$n^{o(1)}$	[W]	Adaptive	[BvdBG$^+$22]
$n^{o(1)}$	$O\left(n^{1+o(1)}\right)$	$n^{o(1)}$	[W]	Deterministic	[vdBCK$^+$23]
$\text{poly}(k) \cdot 2^{O(1/\delta^6)}$	$O\left(n^{1+O(1/k)}\right)$	$n^{O(\delta)}$	[W]	Deterministic	**(this paper)**

Table 1: Summary of known algorithms for fully dynamic spanners. Amortized update time is marked by [A] and worst-case update time by [W]. In the penultimate column we mark wherther the algorithm is deterministic, and, if it is randomized, whether it works against an adaptive or an oblivious adversary. Our result holds for any $512 \leq k \leq (\log n)^{1/49}$ and $1/k \leq \delta \leq 1/400$ where $1/\delta$ is an integer2.

every vertex v is at most Δ. Lastly, we say that demand $\mathcal{D}$ is *h-length* if, for every pair $(a,b) \in \Pi$ of vertices, $\mathsf{dist}_G(u,v) \leq h$. The *routing* of a demand $\mathcal{D}$ is a flow f, where, for every pair $(a,b) \in \Pi$, $D(a,b)$ flow units are sent from a to b. The *congestion* of the routing is the maximum amount of flow through any edge of G. We say that the flow is over paths of length at most d, if every flow-path P with $f(P) > 0$ has length at most d. We say that an algorithm for a static problem has *almost linear* running time, if its running time is $m^{1+o(1)}$, where m is the number of edges in the input graph, and we say that its running time is *near linear*, if it is bounded by $\tilde{O}(m)$; throughout, $\tilde{O}$ is used to hide $\text{poly} \log n$ factors. We now provide an overview of recent work on expander decompositions, routers and related graph notions.

Expanders, Routers and Graph Decompositions.

Expanders are a fundamental notion in both graph theory and algorithms. Thanks to a number of powerful algorithmic techniques that were developed for expanders over the years, they now play a central role in the design of efficient algorithms for a wide variety of graph problems, in static, dynamic, and other settings. One of the central properties that make the expanders useful in many applications is that, as shown in the seminal work of Leighton and Rao [LR99], they have strong routing properties, in the sense that any unit demand can be routed across an expander with polylogarithmic congestion, via paths of polylogarithmic length. An almost-linear time algorithm for computing such routing was shown in [GKS17, GL18].

Another central tool in this area is *expander decomposition*, that decomposes the input graph G into vertex-disjoint subgraphs called clusters, such that each cluster is a strong enough expander, and only a small number of edges of G connect vertices that lie in different clusters. Expander decompositions are often used in order to reduce a given optimization problem in general graphs to that on expanders, which, due to the many useful properties of expanders, is often more tractable. Some examples of algorithmic results relying on expander decomposition include graph sketching and sparsification [ACK$^+$16, JS18, CGP$^+$20], dynamic algorithms for minimum spanning forest [NS17, WN17], and recent breakthrough almost-linear time algorithms for Min-Cost Flow in directed graphs [CKL$^+$22, vdBCK$^+$23], to name just a few.

In their seminal work, Spielman and Teng [ST04] provided the first almost-linear time algorithm that computes a decomposition of a graph into clusters, with somewhat weaker guarantees than the expander decomposition: their decomposition only guarantees that every cluster is contained

inside some expander. In other words, if we are given, for every cluster C in the decomposition $\mathcal{C}$, a unit demand $\mathcal{D}_C$, then all these demands can be routed in G simultaneously via short paths and with low congestion, but the routing of the demand $\mathcal{D}_C$ for a cluster C may not be confined to C, and may use edges and vertices that lie outside of C. While their decomposition algorithm found many applications, the use of this weakened notion of expander decomposition makes some of the resulting algorithms quite complex. Moreover, in some applications, this weakened notion of expander decomposition is not sufficient. The work of Saranurak and Wang [SW19] overcomes this shortcoming, by providing a near-linear time randomized algorithm for computing a proper (or strong) expander decomposition. This so-called "user-friendly" decomposition led to simplifying the analysis of existing algorithms, e.g., [ST04, KLOS14, CKP$^+$17, JS18], and to new applications that crucially require the stronger guarantee of the proper decomposition, that include algorithms for dynamic minimum spanning forest [NSWN17, Wul17, NSW17] and short-cycle decomposition [CGP$^+$20]. Deterministic almost-linear time algorithms for computing proper expander decompositions were later developed by [GLN$^+$19, CGL$^+$20, Chu23].

While expanders provide a valuable and powerful tool for the design of algorithms, they have some shortcomings. One such central shortcoming is that the diameter of an n-vertex expander may be as large as logarithmic in n, and moreover, given a unit demand $\mathcal{D}$, one can only guarantee its routing with low congestion via paths of polylogarithmic length, even if all pairs of vertices with non-negative demand are very close to each other. One implication of this shortcoming is that, when relying on expanders for distance-based problems, such as, for example, dynamic All-Pairs Shortest Paths (APSP), distance oracles, and graph spanners, it seems inevitable that one has to settle for super-logarithmic approximation guarantees. To address this challenge, and in order to generalize the successful paradigm of using expanders in distance-based problems to shorter distances, Ghaffari, Haeupler and Räcke [HRG22] introduced the notion *length-constrained (LC) expanders*. We provide here a *routing characterization* of length-constrained expanders, that was shown by [HHT24] to be equivalent, up to constant factors, to the original definition of [HRG22]. Under this definition, a graph G is an (h,s)-length φ-expander, if any h-length unit demand $\mathcal{D}$ can routed in G via paths of length $O(h \cdot s)$, with congestion at most $\widetilde{O}(1/\varphi)$. A somewhat similar notion of well-connected graphs was introduced in [Chu23], with the same motivation. They also designed a number of algorithmic tools for well-connected graphs, that may be thought of as analogues of similar tools for expanders, and that were recently used in [CZ23] to obtain the first adaptive-update algorithm for dynamic APSP with sublogaritmic approximation and amortized update time $n^{o(1)}$. We provide a formal definition of well-connected graphs and discuss their relationship with LC-expanders in Section 2.4.

Due to the success of the expander decompositions as a powerful tool in algorithm design, it is natural to ask whether one can also decompose a given graph into LC-expanders. Ghaffari, Haeupler and Räcke [HRG22] introduced the notion of LC-expander decomposition, that, in turn, relies on the notion of a *moving cut*. Given a graph G with lengths $\ell(e) \geq 0$ on its edges $e \in E(G)$ and integers h and s, an (hs)-length moving cut L assigns an integral *length increase* value $L(e) \in \{1, \ldots, (hs)\}$ to each edge $e \in E(G)$. A graph that is obtained from G by setting the length of every edge e to $\ell(e) + L(e)$ is denoted by $G - L$. The *size* of the (hs)-length moving cut L is $|L| = \frac{1}{hs} \cdot \sum_{e \in E(G)} L(e)$. An (h,s)-length φ-expander decomposition is simply an (hs)-length moving cut L, such that graph $G - L$ is an (h,s)-length φ-expander.

The work of [HRG22] provided a proof that there exists an (h,s)-length φ-expander decomposition of any n-vertex m-edge graph, for $s = O(\log n)$, such that the resulting (hs)-length moving cut has size $\tilde{O}(\varphi m)$. Additionally, they provide an algorithm for computing a weaker notion of an (h,s)-length φ-expander decomposition.

A very recent work by Haeupler, Hershkowitz and Tan [HHT24] provided an algorithm that, given an

n-vertex m-edge graph G, with parameters $0 < \epsilon < 1$, h and φ, computes an (h, s)-length φ-expander decomposition of G for $s = \exp(\text{poly}(1/\epsilon))$, where the size of the resulting moving cut is at most $n^{\epsilon} \cdot m \cdot \varphi$, in time $O(m \cdot n^{\text{poly}(\epsilon)} \cdot \text{poly}(h))$. Their algorithm was recently extended to the dynamic setting by [HLS24]. A key notion underlying these results is that of *routers*. As mentioned already, a graph G is a router if any unit demand can be routed in G via short paths with low congestion. Equivalently, a router can be thought of as an (h, s)-length φ-expander, whose diameter is bounded by h. In order certify that a graph H is an (hs)-length φ-expander, [HHT24] use a variation of the notion of *neighborhood covers*. In this variation, a neighborhood cover is a collection $\{\mathcal{C}_1, \ldots, \mathcal{C}_r\}$ of clusterings, where, for all $1 \leq i \leq r$, clustering $\mathcal{C}_i$ is a collection of subgraphs of H called clusters, of radius at most h each. Additionally, every pair of clusters in $\mathcal{C}_i$ must be at distance at least hs from each other. Lastly, it is required that every vertex of H belongs to a relatively small number of clusters in $\bigcup_i \mathcal{C}_i$, and, for every vertex v of G, its h-neighborhood $B_H(v, h)$ must be contained in some cluster in $\bigcup_i \mathcal{C}_i$. As shown in [HHT24], in order to certify that a graph H is an (h, s)-length φ-expander, it is sufficient to compute, for every cluster $C \in \bigcup_i \mathcal{C}_i$, a router graph R_C with $V(R_C) = V(C)$, together with an embedding of all such resulting routers R_C into G via short paths, that cause low congestion. Their algorithm then computes a moving cut L for graph G, and certifies that the resulting graph $G' = G - L$ is an (h, s)-length φ-expander by computing the neighborhood cover $\{\mathcal{C}_1, \ldots, \mathcal{C}_r\}$ of G', and then embedding, for every cluster $C \in \bigcup_i \mathcal{C}_i$, the corresponding router R_C into G'. Therefore, the algorithms of [HHT24, HLS24] can also be thought of as computing and dynamically maintaining a weaker notion of router decomposition of the input graph G, similar to the weaker notion of expander decomposition of [ST04]. Specifically, every cluster C in the neighborhood cover is provided with a certificate router graph R_C, with $V(C) = V(R_C)$, and an embedding of R_C into G via short paths. However, this embedding may use edges and vertices that lie outside of C. Our main technical result is an algorithm for maintaining a *proper* router decomposition of a dynamic graph:

Theorem 1.2 (Dynamic Router Decomposition, Informal) *There is a deterministic algorithm, whose input is an n-vertex undirected graph G, a stretch parameter $512 \leq k \leq (\log n)^{1/49}$, and two additional parameters $1/k \leq \delta \leq 1/400$, and $\Delta \geq n^{20/k}$. Graph G initially has no edges, and undergoes an online sequence of edge insertions and deletions. The algorithm maintains a collection $\mathcal{C}$ of edge-disjoint subgraphs (clusters) of G, and, for every cluster $C \in \mathcal{C}$, a subgraph $C' \subseteq C$ with $V(C') = V(C)$, such that the following hold at all times: (i) for every cluster $C \in \mathcal{C}$, any unit demand $\mathcal{D}_C$ for C can be routed inside C via paths of length at most $d \leq \text{poly}(k) \cdot 2^{O(1/\delta^6)}$, with congestion at most $\eta = n^{O(1/k)}$; (ii) $\sum_{C \in \mathcal{C}} |V(C)| \leq n^{1+O(1/k)}$; (iii) for every cluster $C \in \mathcal{C}$, any Δ-restricted demand on $V(C)$ can be routed in C' via paths of length at most d, with congestion at most η; (iv) $\sum_{C \in \mathcal{C}} |E(C')| \leq \Delta \cdot n^{1+O(1/k)}$; and (v) the total number of edges $e \in E(G)$ that do not lie in any cluster of $\mathcal{C}$ is at most $\Delta \cdot n^{1+O(1/k)}$. The worst-case update time of the algorithm is $n^{O(\delta)}$.*

In a sense, our decomposition result can be thought of as strengthening the results of [HHT24, HLS24] to a proper router decomposition, in the same manner as the result of [SW19] strengthened the fast algorithm of [ST04] for weak expander decomposition to obtain a proper expander decomposition. While this may appear as a technicality, just as with expander decompositions, a proper router decomposition provides a cleaner structure that is easier to use in applications. In fact some of the applications of our router decomposition described below crucially rely on the fact that the decomposition is proper. Additionally, for every cluster C in the decomposition, we maintain a sparsified subgraph $C' \subseteq C$ of a prescribed average density that remains a sufficiently strong router. These subgraphs are useful for computing various sparsifiers of G. We note that, unlike the decompositions of [HHT24, HLS24], our decomposition does not provide an LC-expander decomposition of G. Specifically, if we denote by E^{del}

the set of edges that do not lie in any cluster, then our decomposition does not guarantee that $G \setminus E^{\mathrm{del}}$ is a length-constrained expander. This is due to the fact that the clusters in our decomposition do not necessarily define a neighborhood cover in $G \setminus E^{\mathrm{del}}$.

We obtain the result of Theorem 1.1 from Theorem 1.2 as follows: we use the parameter $\Delta = n^{O(1/k)}$; the spanner H is obtained by taking the union of the subgraphs C' for all $C \in \mathcal{C}$, together with all edges of G that do not lie in the clusters of $\mathcal{C}$. We show that H can be maintained with worst-case update time $n^{O(\delta)}$ and recourse $n^{O(1/k)}$. Our results also extend to low-congestion and fault-tolerant spanners, that we discuss next.

Low-Congestion and Fault-Tolerant Spanners.

The following general recipe, pioneered by [BvdBG+22], can be used in order to compute spanners via expander decomposition and routing. Initially, the spanner H contains n vertices and no edges. Next, we compute an expander decomposition $\mathcal{C}$ of G, and, for every cluster $C \in \mathcal{C}$, we compute a sparse subgraph $C' \subseteq C$ that is an expander. For every cluster $C \in \mathcal{C}$ of the decomposition, we then add the edges of C' to H, and we consider the edges of C as "settled". The algorithm then continues recursively with edges that are not settled yet: namely, edges whose endpoints lie in different clusters of $\mathcal{C}$. It is easy to verify that the resulting spanner H has polylogarithmic stretch and that it is sparse. The recent breakthrough almost-linear algorithm of [CKL+22, vdBCK+23] for Min Cost Flow exploits the fact that the spanner H obtained via this approach has additional useful properties. Specifically, they show that one can embed all edges of G into H, such that all the embedding paths are short, and cause low vertex-congestion. This is in contrast to the standard notion of spanners, that only guarantees that, for each edge (u, v) of G, there is a short u-v path in H, though the resulting paths may cause arbitrarily high congestion.

In this work, we formalize this notion by introducing *low-congestion spanners*, and show an algorithm that dynamically maintains such a spanner, by exploiting our result for router decomposition. Formally, let G be a graph, and let H be a subgraph of G. We say that H is a (d, η)-low congestion spanner for G, if there is an embedding of the edges of G into H, such that all embedding paths have length at most d, and cause edge-congestion at most η. We use our router decomposition to show that for any stretch parameter $512 \leq k \leq (\log n)^{1/49}$ and degree threshold Δ, any n-vertex graph G admits a (d, η)-low congestion spanner with $\widetilde{O}(\Delta \cdot n^{1+O(1/k)})$ edges for $d = \mathrm{poly}(k)$ and $\eta = O\left(n^{O(1/k)} \cdot \max\left\{\frac{\Delta_{\max}(G)}{\Delta}, 1\right\}\right)$, where $\Delta_{\max}(G)$ is the maximum vertex degree in G. In fact we show that the graph H that is maintained by the algorithm from Theorem 1.1, and which, in turn, relies on the router decomposition from Theorem 1.2, has this property. We believe that low-congestion spanner is an interesting object that may find other applications in the future.

Fault-Tolerant Spanners. An additional useful property satisfied by expander-based spanners is their resilience to edge faults. As many applications of spanners arise in the context of distributed networks, which are inherently prone to failures of edges and vertices, it is especially desirable to obtain robust spanners, that maintain their functionality in the presence of such faults. *Fault-tolerant (FT) spanners* are specifically designed to provide such guarantees: given a graph G and parameters f and t, a subgraph $H \subseteq G$ is an f-FT t-spanner if, for every collection $F \subseteq E(G)$ of at most t failed edges, $H \setminus F$ is a t-spanner for $G \setminus F$. FT-spanners were first introduced in the context of geometric graphs by Levcopoulos et al. [LNS98], and were later considered for general graphs by Chechik et al. [CLPR10]. Recently, Parter [Par22] presented a near-linear time randomized algorithm for constructing an f-FT $(2k-1)$-spanner with $\widetilde{O}(f^{1-1/k} \cdot n^{1+1/k})$ edges, which is also resilient to up to f vertex faults. Bodwin, Dinitz and Robelle [BDR22] provided an existential result for f-FT $(2k-1)$-spanners: for odd values of k, they show that there exist spanners of size $O(k^2 f^{1/2-1/(2k)} n^{1+1/k} + kfn)$, and for even k, the size

is $O(k^2 f^{1/2} n^{1+1/k} + kfn)$. These bounds almost match the $\Omega\left(f^{1/2-1/(2k)} \cdot n^{1+1/k}\right)$ size lower bound that is based on the Erdös Girth Conjecture [BDPW18].

Notice that, if we use the notion of f-FT spanners, that can only tolerate at most f edge failures, the spanner size must be at least fn, even when allowing a large stretch, since it is possible that all of the edge faults are incident to a single vertex. It is possible, however, that one can obtain sparse spanners that can withstand a significantly higher number of edge faults, as long as these faults are distributed evenly in the graph, and are not concentrated around a small number of vertices. With this motivation in mind, a very recent work of Bodwin, Haeupler and Parter [BHP24] introduced a stronger notion of fault-tolerant spanners, called *Faulty-Degree (FD) spanners*. Specifically, suppose we are given an n-vertex graph G, and parameters t and f. An f-FD t-spanner for G is a graph $H \subseteq G$, with the following property: if F is any subset of edges of G, such that every vertex of G is incident to at most f edges of F, then $H \setminus F$ is a t-spanner for $G \setminus F$. Notice that, for $f = 1$, the set F of edges can be any matching, whose size may be as high as linear in n. Bodwin et al. [BHP24] provided two approaches for computing sparse f-FD spanners. The first approach uses the LC-expander decomposition, and obtains f-FD spanners with stretch $k^{O(k)}$ and size $\widetilde{O}(fn^{1+1/k})$ via an algorithm with a high polynomial running time. Combining this approach with the recent time-efficient algorithm for LC-expander decomposition and routing of [HHT24, HLS24] yields an algorithm with running time $O(m \cdot n^{O(1/k)})$, albeit with a slightly higher stretch $2^{O(\text{poly}(k))}$. Their second approach, based on a greedy algorithm, provides an f-FD spanner with stretch $(2k-1)$ and size $\widetilde{O}\left(f^{1-1/k} \cdot n^{1+1/k} \cdot k^{O(k)}\right)$, which is nearly optimal for fixed values of k under the Erdös Girth Conjecture. The main drawback of the latter approach is the large polynomial running time of their algorithm. In their paper, [BHP24] noted that improving the $k^{O(k)}$ stretch bound obtained via the LC-expanders is closely related to the question of the resilience of LC-expanders to bounded degree faults, which was left open therein (see Theorem 1.14 in [BHP24]). In this work, we build on our results for router decomposition to obtain f-FD spanners with stretch $\text{poly}(k)$, that contain $\tilde{O}(fn^{1+O(1/k)})$ edges. This allows us to address the open problem of [BHP24], only on router graphs (which can viewed as LC-expanders with small diameter), and improve the stretch bound from $k^{O(k)}$ to $\text{poly}(k)$. We also obtain an algorithm for maintainning such a spanner for a fully dynamic graph G: given a parameter $\delta \geq 1/k$, the stretch of the spanner is bounded by $\text{poly}(k) \cdot 2^{O(1/\delta^6)}$, and the worst-case update time of the algorithm is $n^{O(\delta)}$; the algorithm also guarantees that the recourse is bounded by $n^{O(1/k)}$. In particular, this yields an algorithm for constructing a spanner for a static m-edge graph with stretch $\text{poly}(k) \cdot 2^{O(1/\delta^6)}$ in time $m^{1+O(\delta)}$, which outperforms the (at least quadratic) runtime of the greedy-based approach of [BHP24].

Connectivity Certificates. Finally, a graph structure closely related to FT-spanners is the f-edge connectivity certificates, introduced by [NI92]. For an integer $f \geq 1$, we say that a graph G is f-edge connected, if, for any subset F of $f-1$ edges of G, graph $G \setminus F$ is connected. An f-edge connectivity certificate for G is a subgraph $H \subseteq G$, that has the following property: H is f-edge connected if and only if G is. Connectivity certificates play a key role in fast algorithms for minimum cuts; see, e.g. [Mat93, CKT93, KM97, GK13, DHNS19, FNY+20, LNP+21, GHN+23]. Nagamochi and Ibaraki [NI92] presented a linear-time static algorithm for computing f-edge connectivity certificates of n-vertex graphs G, with $O(fn)$ edges. In the dynamic setting, Thorup and Karger [TK00] presented a deterministic algorithm for maintaining an f-edge connectivity certificate of a dynamic n-vertex graph, of size $O(fn)$, with $\widetilde{O}(f^2)$ amortized update time. Combining the latter with the deterministic algorithm of [CGL+20] for dynamic minimum spanning forest yields $O(f^2 \cdot n^{o(1)})$ worst-case update time. The dependency of the update time on f can be avoided using randomness: the recent work of Assadi and Shah [AS23] implies a *randomized* algorithm for maintaining an f-connectivity certificate with $O(f \cdot n \log n)$ edges in a fully dynamic n-vertex graph, with $O(\text{poly} \log n)$ worst-case update time. To the best of our knowledge, there is no deterministic dynamic algorithm for maintaining sparse

f-connectivity certificate with worst-case update time bound that is independent of f. We show that our algorithm for maintaining a router decomposition can also be used to maintain an f-connectivity certificate with $f \cdot n^{1+o(1)}$ edges (and therefore of almost optimal size) for a fully dynamic graph, with worst-case update time and recourse bounded by $n^{o(1)}$. In fact, we maintain an even stronger version of an f-connectivity certificate recently introduced in [BHP24], called f-FD connectivity certificate, that can withstand the failure of any edge subset F, as long as every vertex of G is incident to at most f edges of F.

1.1 Our Results and Techniques

Our main technical result is an algorithm that maintains a low-diameter router decomposition of a fully dynamic graph with low worst-case update time. Before we state the result, we need to define the notions of demands and routers; formal definitions can be found in Sections 2.2 and 2.3. Consider a graph $G = (V, E)$. A *weighting* of the vertices of G is an assignment $w(v) \geq 0$ of a non-negative weight to every vertex $v \in V$. We also denote such a weighting by an n-dimensional vector $\overline{w} \in \mathbb{R}^n$. We sometimes use two special vertex weightings: in the first weighting, denoted by $\overline{\deg}_G$, the weight of every vertex $v \in V$ is its degree in G. In the second weighting, we are given some value $\Delta > 0$, and the weight of every vertex is set to be Δ; we denote such a weighting by $\overline{\Delta}$. A *demand* $\mathcal{D}$ in G consists of a collection Π of pairs of vertices, and, for every pair $(a, b) \in \Pi$, a value $D(a, b) > 0$, that we refer to as the *demand between a and b*. For a vertex weighting $\overline{w}$, we say that a demand $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ is *$\overline{w}$-restricted*, if, for every vertex $v \in V$, the total demand between all pairs in Π that contain v is at most $w(v)$. A *routing* of the demand $\mathcal{D}$ is a flow f, in which, for every pair $(a, b) \in \Pi$, $D(a, b)$ flow units are sent from a to b. The *congestion* of the routing is the maximum amount of flow through any edge of G. We say that the routing is via *flow-paths of length at most d*, if every flow-path with non-zero flow value contains at most d edges.

A central notion that we use in our paper is that of a *router*, that was first defined in [HHT24] as an object that is closely related to length-constrained expanders. Given a graph G with vertex weighting $\overline{w}$, and parameters η and d, we say that G is a $(\overline{w}, d, \eta)$-router[1], if every $\overline{w}$-restricted demand $\mathcal{D}$ can be routed with congestion at most η in G, via paths of length at most d.

Our main technical result, summarized in the following theorem, is an algorithm that maintains a router decomposition of a fully dynamic graph, with small worst-case update time. The algorithm also maintains a subgraph H of G of a prescribed density that has additional useful properties. In particular, as we show later, graph H is a (low-congestion and fault-tolerant) spanner of G.

Theorem 1.3 *There is a deterministic algorithm, whose input is a graph G with n vertices that initially has no edges, which undergoes an online sequence of at most n^2 edge deletions and insertions, such that G remains a simple graph throughout the update sequence. Additionally, the algorithm is given parameters $512 \leq k \leq (\log n)^{1/49}$, $\frac{1}{k} \leq \delta < \frac{1}{400}$ and $\Delta \geq n^{20/k}$, such that k, Δ and $1/\delta$ are integers. The algorithm maintains the following:*

- *a collection $\mathcal{C}$ of edge-disjoint subgraphs of G called* clusters *with $\sum_{C\in\mathcal{C}} |V(C)| \leq n^{1+O(1/k)}$, such that every cluster $C \in \mathcal{C}$ is a $(\overline{\deg}_C, d, \eta)$-router for $d = k^{19} \cdot 2^{O(1/\delta^6)}$ and $\eta = n^{O(1/k)}$. Moreover, if we denote by $E^{\mathrm{del}} = E(G) \setminus \left(\bigcup_{C\in\mathcal{C}} E(C)\right)$, then, at all times, $|E^{\mathrm{del}}| \leq n^{1+O(1/k)} \cdot \Delta$ holds;*
- *for every cluster $C \in \mathcal{C}$, a subgraph $C' \subseteq C$ with $V(C') = V(C)$, that is a $(\overline{\Delta}, d, \eta)$-router, with $\sum_{C\in\mathcal{C}} |E(C')| \leq n^{1+O(1/k)} \cdot \Delta$; and*

[1] in Section 2.3 we provide a slightly more general definition of routers, that includes a set S of supported vertices, where the router is only required to route demands defined over the vertices of S. But all our results hold for the case where $S = V(G)$, so the definition above is sufficient in order to state our results.

- *a graph $H \subseteq G$ with $|E(H)| \leq n^{1+O(1/k)} \cdot \Delta$, that contains all edges of $E^{\mathrm{del}} \cup \left(\bigcup_{C \in \mathcal{C}} E(C')\right)$.*

The worst-case update time of the algorithm is $n^{O(\delta)}$ per operation, and the total number of edge insertions and deletions that graph H undergoes after each update to G is at most $n^{O(1/k)}$.

We note that, if we set $\Delta = \lceil n^{20/k} \rceil$, then it is guaranteed that $|E(H)| \leq n^{1+O(1/k)}$ holds, and moreover, it is easy to verify that H is a d-spanner for G. In Section 9 we show that, in addition to being a d-spanner, H has many other useful properties; for example, it is also a low-congestion spanner. Moreover, given a fault parameter f, by setting the bound Δ appropriately, we can guarantee that H is an f-FD spanner. Lastly, graph H can also be used as an f-connectivity certificate for G. Interestingly, the worst-case update time to maintain these structures is independent of the parameter f.

We now provide an overview of the techniques that we employ in the proof of Theorem 1.3. Using a standard deamortization technique (see [NSWN17, JS22], and Section 2.7), it is sufficient to provide an algorithm with guarantees similar to those in Theorem 1.3 in the *online-batch dynamic model*. In this model, the initial graph G may contain an arbitrary number of edges, and it undergoes k batches $\pi_1, \ldots, \pi_k$ of updates, where, for all $1 \leq i \leq k$, the ith batch π_i of updates is a collection of edge insertions and deletions for graph G. In order to obtain the desired worst-case update time and recourse bound from Theorem 1.3, it is enough to design an algorithm in the online-batch model, whose initialization time is $m^{1+O(\delta)}$; the time required to process each batch π_i is bounded by $|\pi_i| \cdot n^{O(\delta)}$; and the total number of edge insertions and deletions that graph H undergoes following each batch π_i of updates is bounded by $|\pi_i| \cdot n^{O(1/k)}$. We develop several technical tools in order to solve this problem, that we describe next.

First tool: nice router $W_k^{N,\Delta}$ and its pruning. Our first tool is an explicit construction of a nice well-structured router graph, and an algorithm for its pruning. This construction is very similar to that of [HLS24]. However we modify their construction slightly, and we modify their pruning algorithm significantly, in order to obtain a better distance-time and distance-size tradeoffs. Suppose we are given parameters N, Δ and k. The corresponding graph $W_k^{N,\Delta}$ (that, for simplicity, we will denote by W_k), consists of N^k vertices, that are partitioned into a set V^c of N^{k-1} *center* vertices, and a set V^ℓ of remaining vertices, called *leaf* vertices. The set of edges of W_k is partitioned into k subsets $E_1, \ldots, E_k$, where, for $1 \leq i \leq k$, we refer to the edges of E_i as *level-i edges*. For all $1 \leq i \leq k$, the subgraph of W_k induced by the edges of E_i can be thought of as consisting of a union of N^{k-1} disjoint star graphs (called *level-i stars*), each of which has $N-1$ leaves, after we replace every edge of every star by a collection of Δ parallel edges (that we refer to as a *bundle*; we also refer to the corresponding star edge as a *superedge*). The center vertices of all stars lie in V^c, and their leaves lie in V^ℓ.

The construction of the graph W_k is inductive. In order to obtain the graph W_1, we start with a star S that has $N-1$ leaves. We then replace every edge of S with Δ parallel edges, obtaining the graph W_1. The center of the star S becomes the unique center vertex of W_1, and the edges of W_1 are *level-1 edges*. Assume now that, for an integer $1 < i \leq k$, we have defined the graph W_{i-1}. In order to obtain a graph W_i, we start by constructing N disjoint copies of $C_1, \ldots, C_N$ of the graph W_{i-1}; we refer to these copies as *level-$(i-1)$ clusters*. The set of the center vertices of W_i is the union of the sets of center vertices of the graphs $C_1, \ldots, C_N$, and the remaining vertices of W_i are leaf vertices. We connect the clusters $C_1, \ldots, C_N$ to each other by level-i edges, as follows. We construct N^{k-1} disjoint stars $S_1, \ldots, S_{N^{k-1}}$ (level-i stars), so that each star S_j has $N-1$ leaves; each vertex of S_j lies in a distinct cluster of $\{C_1, \ldots, C_N\}$; and the center of the star S_j is a center vertex of W_k. For each such star S_j, we then replace each of its edges with Δ parallel edges, which are referred to as level-i edges.

As mentioned already, this construction is very similar to that used in [HLS24]; the main difference is that they used cliques instead of stars, and in their construction $\Delta = 1$. Next, we define the notion

of a *properly pruned subgraph* of W_k. Suppose we are given a subgraph $W \subseteq W_k$, and k collections $U_1, \ldots, U_k$ of vertices of W, with $U_k \subseteq U_{k-1} \subseteq \cdots \subseteq U_1$. We say that W is a *properly pruned subgraph* of W_k with respect to the sets $U_1, \ldots, U_k$ of its vertices, if $V(W) = U_1$, and the graph obeys the following additional rules. First, for each level $1 \leq i \leq k$, for each level-i superedge (u, v), where v is a leaf vertex, if v remains in set U_i, then at least $\Delta/2$ parallel edges (u, v) must remain in W. Second, for each level $1 \leq i \leq k$, and each level-i star S, if the center vertex v of S remains in U_i, then a large fraction of the leaves of S must remain in U_i. Lastly, for all $1 \leq i < k$, for every level-i cluster C, if any vertex of C remains in U_1, then a significant fraction of vertices of C must lie in U_{i+1}. We prove that, if W is a properly pruned subgraph of W_k, then it is a strong router: namely, any Δ-restricted demand can be routed in W via paths of length $O(k^2)$, with congestion $k^{O(k)}$. The routing algorithm uses the star structure of the graph W_k in a natural manner. As mentioned already, our algorithm for pruning the graph W_k is more involved than that of [HLS24]; this is in order to obtain stronger routing properties in W. For example, in the construction of [HLS24], the bound on the lengths of the routing paths was at least exponential in k (see Theorem 5.1 in [HLS24]).

Finally, we provide an algorithm for pruning the graph W_k in the online-batch dynamic model. The algorithm is given k batches $\pi_1, \ldots, \pi_k$ of edge deletions from W_k, and maintains a properly pruned decremental subgraph W of W_k throughout the update sequence. The algorithm ensures that the processing time of each batch π_i is bounded by $O(|\pi_i| \cdot k^{O(k)})$, and the number of edges and vertices deleted from W after each batch π_i is small compared to $|\pi_i|$. The pruning algorithm is quite straightforward: whenever any of the above rules are not obeyed, we perform vertex deletions in order to ensure that the current graph W is compliant with the rules.

One additional advantage of our construction is that it is easy to maintain a sparsified subgraph W' of $W_k^{N,\Delta}$ of any prescribed density $\Delta' < \Delta$ that remains a strong router, even as the graph undergoes edge deletions and pruning: for every superedge (u, v), we simply include in W' arbitrary Δ' parallel copies of (u, v). This, in turn, allows us to maintain the sparsified routers $C' \subseteq C$ for the clusters C in the router decomposition $\mathcal{C}$ that our algorithm maintains.

Second tool: embedding a nice router. Our second tool is an algorithm that can be used in order to embed a nice router W_k into a given graph. Specifically, suppose we are given a graph C, and parameters k and $d = k^{O(1)}$, such that, for some vertex $v \in V(C)$, the number of vertices of C that lie within distance d from v is significantly larger than $|V(C)|^{1-1/k}$. Assume further that $|E(C)| \approx |V(C)| \cdot \Delta \cdot n^{\Theta(1/k)}$. Our algorithm either successfully embeds a graph $W_k^{N,\Delta}$ into C via short paths that cause small congestion, where N^k is quite close to $|V(C)|$; or computes a subset E' of $O(n\Delta)$ edges, so that $C \setminus E'$ becomes *scattered*: that is, for every vertex $u \in V(C)$, the ball of radius d around u in $C \setminus E'$ contains fewer than $|V(C)|^{1-1/k}$ vertices. More specifically, our algorithm uses an additional parameter δ to control the tradeoff between the lengths of the embedding paths and the running time. In the first case, the algorithm computes a relatively small collection F of edges of $W_k^{N,\Delta}$, and an embedding $\mathcal{P}$ of $W_k^{N,\Delta} \setminus F$ into C via paths of length $\mathrm{poly}(k) \cdot 2^{O(1/\delta^6)}$, with congestion at most $n^{O(1/k)}$. The running time of the algorithm is $m^{1+O(\delta)}$. If the algorithm terminates with an embedding of $W_k^{N,\Delta} \setminus F$ into C, then we can employ our pruning algorithm, in order to compute a large enough properly pruned subgraph W of $W_k^{N,\Delta}$ that does not contain any edges of F, together with its embedding $\mathcal{P}'$ via paths whose length and congestion are bounded as before. Moreover, we can compute a large enough subgraph $\hat{C} \subseteq C$ that contains all embedding paths in $\mathcal{P}'$, such that every vertex of $\hat{C}$ participates in a large number of such paths. As we show later, this is sufficient in order to prove that $\hat{C}$ is a $(\overline{\deg}_{\hat{C}}, d, \eta)$-router.

The algorithm for computing the embedding of $W_k^{N,\Delta} \setminus F$ (or a subset E' of edges whose removal makes C scattered), employs the algorithmic techniques that were developed in [Chu23] for *well-connected graphs*. Specifically, we start by selecting a large subset T of vertices of C that are close to each

other, and then use an algorithm from [Chu23] in order to attempt to embed a well-connected graph, whose vertex set is a large subset of T, into C, via short paths that cause a low congestion. If the algorithm fails to do so, then it computes a small collection E'' of edges, so that, in $C \setminus E''$, many pairs of vertices of T become far from each other. In the latter case, we either compute a new large subset T of vertices of C that remain close to each other and restart the same algorithm; or correctly establish that the graph becomes scattered and terminate the algorithm. Using simple accounting, we show that the number of such iterations in the algorithm is not too large. If we successfully embed a large well-connected graph R into C, we employ the algorithm of [Chu23] for APSP in well-connected graphs, together with the standard Even-Shiloach trees, in an attempt to embed the edges of $W_k^{N,\Delta}$ into C one by one. If we fail to embed a large fraction of the edges of $W_k^{N,\Delta}$, then we again obtain a small subset E'' of edges, such that, in $C \setminus E''$, many pairs of vertices that were originally close to each other, become far from each other. All such sets E'' of edges removed from C over the course of the algorithm are added to a set E', and, since the number of iterations in the algorithm is relatively small, we can ensure that $|E'|$ remains sufficiently small as well.

Third tool: router witness. Suppose we are given a graph W, that is a properly pruned subgraph of the graph $W_k^{N,\Delta}$. Assume further that we are given some graph C, and an embedding of W into C via paths of length d^* that cause congestion at most η^*. Assume further that all vertex degrees in C are at most $\alpha \cdot \Delta$, and every vertex $v \in V(C)$ lies on at least Δ/β of the embedding paths. We prove that, in such a case, C must be a $(\overline{\deg}_C, \tilde{d}, \tilde{\eta})$-router, for $\tilde{d} = d^* \cdot \mathrm{poly}(k)$, and $\tilde{\eta} = \alpha \cdot \beta \cdot d^* \cdot \eta^* \cdot k^{O(k)}$. Therefore, we can view the graph W and its embedding $\mathcal{P}$ into C with the above properties as a *witness* that C is a $(\overline{\deg}_C, \tilde{d}, \tilde{\eta})$-router. In our construction of the router decomposition, we employ this type of witnesses to ensure that all clusters in the decomposition remain strong routers.

Assume now that we are given a target parameter $\Delta^* \ll \Delta$, a cluster C, and a router witness $(W, \mathcal{P})$ as described above. Assume further that our goal is to compute a *sparsified router* $C' \subseteq C$ for C, so that $V(C') = V(C)$, $|E(C')| \leq \Delta^* \cdot |V(C)|$, and C' is a $(\Delta^*, \tilde{d}, \tilde{\eta}')$-router, where $\tilde{d}$ remains the same as before, and $\tilde{\eta}'$ is only slightly higher than $\tilde{\eta}$. We show that such a cluster C' can be constructed as follows. We let $\Delta' < \Delta^*$ be a parameter that is sufficiently close to Δ^*. First, for every superedge (a, b) of W, we select a subset $E'(a, b)$ of Δ' parallel edges (a, b), and we let $\mathcal{Q}' \subseteq \mathcal{P}$ be the set of all embedding paths of all edges in sets $E'(a, b)$, for all superedges (a, b) of W. Next, we construct a collection $\mathcal{Q}'' \subseteq \mathcal{P}$ of paths as follows: every vertex $v \in V(C)$ selects a subset of Δ' paths of $\mathcal{P}$ that contain v, and adds them to $\mathcal{Q}''$; but we require that every leaf vertex of W serves as an endpoint of at most $\gamma \cdot \Delta'$ such paths, for $\gamma = n^{O(1/k)}$. We then let $C' \subseteq C$ be the graph that contains all edges and vertices participating in the paths of $\mathcal{Q}' \cup \mathcal{Q}''$. We prove that such a graph C' must be a $(\Delta^*, \tilde{d}, \tilde{\eta}')$-router, by using the routing properties of properly pruned subgraphs of $W_k^{N,\Delta}$ in a straightforward manner. We also show that $|E(C')| \leq |V(C')| \cdot \Delta^*$.

Fourth tool: decremental low-diameter clustering. Our last tool is a simple algorithm for decremental low-diameter clustering. Suppose we are given a graph G and a parameter $k < \log n$. We say that a subgraph $C \subseteq G$ is a *settled cluster*, if either $|V(C)| \leq n^{1/k}$, or there is a vertex $v \in V(C)$, such that the ball of radius $\mathrm{poly}(k)$ around v contains at least $|V(C)|^{1-1/k}$ vertices. If a subgraph $C \subseteq G$ is not a settled cluster, then we call it *unsettled*.

In the decremental low-diameter clustering problem, the goal is to maintain a decomposition $\mathcal{C}$ of G into edge-disjoint clusters under edge deletions, so that each cluster in $\mathcal{C}$ is settled, and $\sum_{C \in \mathcal{C}} |V(C)| \leq n^{1+O(1/k)}$ holds (for conciseness, we refer to the latter property as a *low overlap*). We note that the settled clusters in the decomposition that we maintain are not guaranteed to have a low diameter; each such cluster either has relatively few vertices, or it has a single vertex with a sufficiently large neighborhood.

Specifically, the algorithm is required to first compute an initial decomposition $\mathcal{C}$ of G into settled

clusters with low overlap. The set $\mathcal{C}$ of clusters is then partitioned by the adversary into a set $\mathcal{C}^I$ of *inactive* and a set $\mathcal{C}^A$ of *active* clusters. At the beginning, all clusters are active. Once a cluster becomes inactive, it may not undergo any further updates. Then the algorithm proceeds in iterations. In every iteration, the adversary may (i) move some clusters from $\mathcal{C}^A$ to $\mathcal{C}^I$; and (ii) delete some edges and vertices from clusters $C \in \mathcal{C}^A$, after which each such cluster must become unsettled. After that the algorithm is required to "fix" the clusters in $\mathcal{C}^A$ via *cluster-splitting* updates: given a cluster $C \in \mathcal{C}$ and a subgraph $C' \subseteq C$, add C' to $\mathcal{C}$ and to the set $\mathcal{C}^A$ of active clusters, delete all edges of C' from C, and delete any resulting isolated vertices from C. The algorithm must perform such cluster splitting operations, until all clusters in $\mathcal{C}^A$ become settled. We provide a simple algorithm for the decremental low-diameter clustering problem that is based on the standard ball-growing technique.

Combining all tools together. We are now ready to describe the initialization algorithm for Theorem 1.3. Assume first that all vertex degrees in the input graph G are close to $\Delta \cdot n^{\Theta(1/k)}$, for some $\Delta \gg \Delta^*$. We use the algorithm for the low-diameter clustering problem, to compute an initial collection $\mathcal{C}$ of edge-disjoint clusters with low overlap, so that all clusters in $\mathcal{C}$ are settled. We set $\mathcal{C}^I = \emptyset$, $\mathcal{C}^A = \mathcal{C}$, and then iterate, as long as $\mathcal{C}^A \neq \emptyset$. In every iteration, we consider every cluster $C \in \mathcal{C}^A$ one by one. For each such cluster C, we attempt to embed a graph $W_k^{N_C,\Delta} \setminus F$ into C, where F is a relatively small set of fake edges, and N_C is chosen so that $\frac{|V(C)|}{n^{\Theta(1/k)}} \leq N_C^k \leq |V(C)|$, via short paths that cause low congestiton. If we fail to do so, then we obtain a relatively small set E_C of edges, such that graph $C \setminus E_C$ is scattered. We delete the edges of E_C from C (and add them to the set E^{del} of deleted edges that we maintain), and notify the algorithm for the low-diameter clustering problem regarding these edge deletions; we show that C now becomes unsettled. If, however, we manage to embed a graph $W_k^{N_C,\Delta} \setminus F$ into C as above, then we move C to the set $\mathcal{C}^I$ of inactive clusters. Using the pruning algorithm, we then compute a large subgraph $\hat{C} \subseteq C$, together with a router witness for $\hat{C}$, so that $\hat{C}$ is $(\overline{\deg}_{\hat{C}}, \tilde{d}, \tilde{\eta})$-router. We also construct a subgraph $\hat{C}' \subseteq \hat{C}$, with $V(\hat{C}') = V(\hat{C})$, $|E(\hat{C}')| \leq \Delta^* \cdot |V(\hat{C})|$, such that $\hat{C}'$ is a $(\overline{\Delta}^*, \tilde{d}, \tilde{\eta})$-router. We then add $\hat{C}$ to the router decomposition that we are constructing.

Once the algorithm terminates, $\mathcal{C}^A = \emptyset$ holds, and so all clusters are inactive. For each such inactive cluster C, we have constructed a large enough router $\hat{C} \subseteq C$. If the number of edges lying outside of the routers that we have constructed so far is too high, then we repeat the algorithm with these remaining edges. However, since our algorithm ensures that a large enough fraction of edges of each cluster C are included in the corresponding router $\hat{C}$, we can show that the number of times that we need to restart the algorithm is relatively small.

Our algorithm for processing each batch π_i of updates to graph G proceeds as follows. Assume first that all updates in π_i are edge deletions. Then we simply use the pruning algorithm for the nice routers that we described above, inside each cluster C of the decomposition, in order to maintain the corresponding router witness.

Lastly, so far we have assumed that all vertex degrees in the graph G are close to each other, and that the update batches π_i only contain edge deletions. In order to handle a general graph G, we partition its edges among at most k subgraphs $G_1, G_2, \ldots$, where, for each j, $|E(G_j)| \leq \Delta^* \cdot n^{1+j/k}$, and all vertex degrees in G_j are at least $\Delta^* \cdot n^{(j-2)/k}$. For each such subgraph G_j, we further perform *splitting* of its high-degree vertices, to ensure that all vertex degrees become close to $\Delta^* \cdot n^{(j-2)/k}$, and the number of vertices in the resulting graph G'_j remains at most $2n$. For all $j > 2$, we initialize the above algorithm on graph G'_j. In fact, the initialization proceeds from higher to lower indices j, and edges that are left over from the initialization algorithm for graph G'_j (that is, edges that do not lie in any of the routers), become the edges of graph G_{j-1}. The set E^{del} of edges is then set to contain all edges of $E(G_1) \cup E(G_2)$. Consider now a batch π_i of updates. We compute the smallest integer j, so that $\Delta^* \cdot n^{1+(j-2)/k} \geq |\pi_i|$. For indices $j' > j$, for each edge e of $G'_{j'}$ that is deleted in π_i, we delete e from

$G'_{j'}$, and update the router decomposition of $G'_{j'}$ following these edge deletions; this may result in an additional collection $E_{j'}$ of edges to be deleted from $G_{j'}$, where $|E_{j'}|$ is roughly bounded by the number of edges of $G_{j'}$ that are deleted by π_i. Next, we construct a set E^* of edges, that contains (i) all edge insertions from π_i; (ii) all edges lying in graphs $G_1, \ldots, G_j$; and (iii) all edges in the sets $E_{j'}$ that were deleted from graphs $G_{j'}$ for $j' > j$ while the batch π_i was processed. We let the new graph G_j contain all edges of E^*, and we initialize the algorithm for computing the router decomposition on the corresponding vertex-split graph G'_j from scratch. As before, all leftover edges that are not included in the resulting router decomposition are added to the graph G_{j-1}, and we recursively initialize the algorithm for computing the router decomposition on the corresponding graph G'_{j-1}, and so on.

Organization. We start with Preliminaries in Section 2. We formally define router decompositions, and state our main technical result: namely, an algorithm that maintains a router decomposition of a graph that undergoes a small number of batches of updates, in Section 3. We also show that the proof of Theorem 1.3 follow from this main technical result. Then in sections Sections 4–7 we develop our main technical tools for maintaining a router decomposition. The first tool is a simple construction of a nice router, and an algorithm for its pruning, that are presented in Section 4. The second tool is an algorithm for embedding a nice router into a given graph, and is presented in Section 5. The third tool is a router witness, that is presented in Section 6. Finally, the last tool is a decremental algorithm for low-diameter decomposition, that appears in Section 7. In Section 8, we combine all these tools to complete the proof of the main technical theorem. Lastly, in Section 9, we provide an algorithm for maintaining (low congestion, fault tolerant) spanners for bounded-degree faults in a fully dynamic graph, and connectivity certificates.

2 Preliminaries

All logarithms in this paper are to the base of 2. Throughout the paper, we use the $\tilde{O}(\cdot)$ notation to hide multiplicative factors that are polynomial in $\log m$ and $\log n$, where m and n are the number of edges and vertices, respectively, in the initial input graph. All graphs in this paper are undirected. By default, we allow parallel edges but we do not allow self loops. Graphs in which parallel edges are not allowed are explicitly referred to as *simple* graphs.

We follow standard graph-theoretic notation. Given a graph $G = (V, E)$ and two disjoint subsets A, B of its vertices, we denote by $E_G(A, B)$ the set of all edges with one endpoint in A and another in B, and by $E_G(A)$ the set of all edges with both endpoints in A. We also denote by $\delta_G(A)$ the set of all edges with exactly one endpoint in A. For a vertex $v \in V(G)$, we denote by $\delta_G(v)$ the set of all edges incident to v in G, and by $\deg_G(v)$ the degree of v in G. We also denote by $\overline{\deg}_G$ the n-dimensional vector of all vertex degrees in G. We may omit the subscript G when clear from context. Given a subset $S \subseteq V$ of vertices of G, we denote by $G[S]$ the subgraph of G induced by S.

If G is a graph, and $\mathcal{P}$ is a collection of paths in G, we say that the paths in $\mathcal{P}$ cause *congestion* η, if every edge $e \in E(G)$ participates in at most η paths in $\mathcal{P}$, and some edge participates in exactly η such paths.

Assume now that we are given a graph G with capacities $c(e) > 0$ on edges $e \in E(G)$. A *flow* in G is an assignment of non-negative flow values $f(P)$ to paths P in G. If P is a path whose corresponding flow value $f(P)$ is non-zero, then we say that P is a *flow-path*. A flow f is typically defined by providing a list of flow-paths P together with their corresponding flow value $f(P)$. For an edge $e \in E(G)$, the flow $f(e)$ through e is the total sum of flow values $f(P)$ over all paths P containing e. We say that flow f causes *congestion* η if $\max_{e \in E(G)} \left\{ \frac{f(e)}{c(e)} \right\} = \eta$. Notice that it is possible that $\eta < 1$. If $f(e) \leq c(e)$

holds for every edge $e \in E(G)$, we may sometimes say that f is a *valid flow*, or that it *respects edge capacities*. When the edge capacities $c(e)$ are not explicitly given, we assume that they are unit.

Distances and Balls. Suppose we are given a graph G with lengths $\ell(e) > 0$ on its edges $e \in E(G)$. For a path P in G, we denote its length by $\ell_G(P) = \sum_{e \in E(P)} \ell(e)$. For a pair of vertices $u, v \in V(G)$, we denote by $\mathsf{dist}_G(u, v)$ the *distance* between u and v in G: the smallest length $\ell_G(P)$ of any path P connecting u to v in G. For a pair S, T of subsets of vertices of G, we define the distance between S and T to be $\mathsf{dist}_G(S, T) = \min_{s \in S, t \in T} \{\mathsf{dist}_G(s, t)\}$. For a vertex $v \in V(G)$, and a subset $S \subseteq V(G)$ of vertices, we also define the distance between v and S as $\mathsf{dist}_G(v, S) = \min_{u \in S} \{\mathsf{dist}_G(v, u)\}$. The *diameter* of the graph G, denoted by $\mathsf{diam}(G)$, is the maximum distance between any pair of vertices in G.

Consider now some vertex $v \in V(G)$, and a distance parameter $D \geq 0$. The *ball of radius D around v* is defines as: $B_G(v, D) = \{u \in V(G) \mid \mathsf{dist}_G(u, v) \leq D\}$. Similarly, for a subset $S \subseteq V(G)$ of vertices, we let the ball of radius D around S be $B_G(S, D) = \{u \in V(G) \mid \mathsf{dist}_G(u, S) \leq D\}$. We will sometimes omit the subscript G when clear from context.

Embeddings of Graphs. Let G, X be two graphs with $|V(X)| \leq |V(G)|$. An *embedding* of X into G consists of a mapping $g : V(X) \to V(G)$, such that the vertices of X are mapped to **distinct** vertices of G, and a collection $\mathcal{P} = \{P(e) \mid e \in E(X)\}$ of paths in graph G, such that, for every edge $e = (x, y) \in E(X)$, path $P(e)$ connects $g(x)$ to $g(y)$. The *congestion* of the embedding is the maximum, over all edges $e' \in E(G)$, of the number of paths in $\mathcal{P}$ containing e'. Given an embedding of X into G as above, we will often identify vertices $v \in V(X)$ with the corresponding vertices $g(v) \in V(G)$ to which they are mapped, and so the embedding will only be specified by the collection $\mathcal{P}$ of embedding paths. Whenever we say that we are given an embedding of a graph X into a graph G, this implicitly means that $|V(X)| \leq |V(G)|$.

2.1 Vertex Weighting, Conductance and Expanders

Given an n-vertex graph $G = (V, E)$, a *vertex weighting* $w : V \mapsto \mathbb{Z}_{\geq 0}$ is an assignment of non-negative integral weight $w(v)$ to every vertex $v \in V(G)$. We denote by $\overline{w}$ the corresponding n-dimensional vector of all vertex weights. We may sometimes consider two special vertex weightings: weighting $\overline{\deg_G}$, where the weight of every vertex v is $\deg_G(v)$; and a weighting $\overline{\Delta}$, where $\Delta > 0$ is an real number, and $\overline{\Delta}$ is an n-dimensional vector with all entries equal to Δ, so the weight of every vertex is Δ.

Consider now a graph $G = (V, E)$ with a vertex weighting $\overline{w}$. For a subset $S \subseteq V$ of vertices, the weight of S is $w(S) = \sum_{v \in S} w(v)$. When $\overline{w} = \overline{\deg_G}$, we may denote $w(S)$ by $\mathrm{Vol}_G(S)$, and refer to it as the *volume of S in G*. If $S, V \setminus S \neq \emptyset$, we sometimes refer to $(S, V \setminus S)$ as a *cut*. Given a cut $(S, V \setminus S)$ with $w(S), w(V \setminus S) > 0$, the *conductance* of the cut with respect to vertex weighting $\overline{w}$ is:

$$\Phi_{G,\overline{w}}(S) = \frac{|\delta_G(S)|}{\min\{w(S), w(V \setminus S)\}} .$$

For simplicity, if $w(S)$ or $w(V \setminus S)$ is 0, we define $\Phi_{G,\overline{w}}(S) = 0$.

The *conductance* of a graph G with respect to vertex weighting $\overline{w}$ is denoted by $\Phi(G, \overline{w})$, and it is defined to be the minimum conductance $\Phi_{G,\overline{w}}(S)$ of any cut $(S, V \setminus S)$.

If no vertex-weighting is explicitly given for a graph $G = (V, E)$, we use the default weighting $\overline{w} = \overline{\deg_G}$.

In such a case, the conductance of a cut $(S, V \setminus S)$ is defined with respect to the standard volume $\mathrm{Vol}_G(\cdot)$ notion defined above.

Definition 2.1 (φ-expander) *Let $G = (V, E)$ be a graph, let $\overline{w}$ be a weighting of its vertices, and let $\varphi \geq 0$ be a parameter. We say that G is a φ-expander with respect to $\overline{w}$, if $\Phi(G, \overline{w}) \geq \varphi$. If G is φ-expander with respect to vertex weighting $\overline{w} = \overline{\deg_G}$, then we may simply say that G is a φ-expander.*

2.2 Demands and their Routing

Definition 2.2 ($\overline{w}$-restricted demand) *Let $G = (V, E)$ be a graph, and let $\overline{w}$ be a weighting of its vertices. A* demand *in G consists of a collection Π of pairs of vertices, and, for every pair $(a, b) \in \Pi$, a value $D(a, b) > 0$, that we refer to as the* demand between *a and b. We say that a demand $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ is* $\overline{w}$-restricted*, if, for every vertex $v \in V$, the total demand between all pairs in Π that contain v is at most $w(v)$. When $\overline{w} = \overline{\Delta}$ for an integer Δ, we may sometimes refer to a $\overline{w}$-resticted demand as a Δ-restricted demand.*

The support *of the demand $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ is the set of all vertices participating in the pairs in Π. We say that the demand is between two pairs $A, B \subseteq V$ of subsets of vertices of G, if $\Pi \subseteq A \times B$.*

Definition 2.3 (Routing of a demand) *Let $G = (V, E)$ be a graph, and let $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ be a demand in G. For every pair $(a, b) \in \Pi$, let $\mathcal{P}_{a,b}$ denote the collection of all a-b paths in G. A* fractional routing*, or just a routing, of the demand $\mathcal{D}$, is a flow f, that assigns non-zero values $f(P) > 0$ only to flow-paths $P \in \bigcup_{(a,b)\in\Pi} \mathcal{P}_{a,b}$, such that, for all $(a, b) \in \Pi$, $\sum_{P\in\mathcal{P}_{a,b}} f(P) = D(a, b)$. The routing is defined by only specifying values $f(P)$ that are non-zero. The* congestion *of the routing is the congestion caused by the flow f in G. We say that the routing is* over paths of length at most *d, if every flow-path P with $f(P) > 0$ has length at most d. If, for every path P, $f(P) \in \{0, 1\}$, then we say that the routing is* integral*. In this case, we may represent the routing as a collection of paths $\mathcal{Q} = \{P \mid f(P) = 1\}$*

2.3 Routers

A central object that we use is routers. Routers were introduced in [HHT24] as an object that is closely related to bounded-hop expanders. As we show below, routers are also closely related to the well-connected graphs, that were introduced in [Chu23]. The definition below slightly generalizes the definition of [HHT24].

Definition 2.4 ($(\overline{w}, d, \eta)$-Router.) *Let $G = (V, E)$ be a graph, let $\overline{w}$ be a weighting of its vertices, and let $S \subseteq V$ be a subset of vertices of G. Additionally, let $d, \eta \geq 1$ be parameters. We say that G is a* $(\overline{w}, d, \eta)$-router *for the set S of supported vertices, if every $\overline{w}$-restricted demand $\mathcal{D}$ defined over the vertices of S can be routed with congestion at most η in G, via paths of length at most d.*

We note that the inclusion of the set S of supported vertices in the above definition may seem redundant, since we could simply set the weight of each such vertex to 0, without the need to include the vertices of S in the definition. However, supported vertices will be useful when we consider special vertex weightings, such as $\overline{\Delta}$ for $\Delta \in \mathbb{R}^{>0}$, and $\overline{\deg_G}$.

2.4 Well-Connected Graphs, their Relationship to Routers, and Related Algorithmic Tools

We employ well-connected graphs, that were introduced in [Chu23]. We start with a formal definition of such graphs, which is identical to that from [Chu23].

Definition 2.5 (Well-Connected Graph) *Given an n-vertex graph G, a set S of its vertices called* supported vertices*, and parameters $\eta, d > 0$, we say that graph G is (d,η)-well-connected with respect to S, if, for every pair $A, B \subseteq S$ of disjoint equal-cardinality subsets of supported vertices, there is a collection $\mathcal{P}$ of paths in G, that connect every vertex of A to a distinct vertex of B, such that the length of each path in $\mathcal{P}$ is at most d, and every edge of G participates in at most η paths in $\mathcal{P}$.*

For intuition, it may be convenient to think of $d = 2^{\mathrm{poly}(1/\epsilon)}$, $\eta = n^{O(\epsilon)}$, and $|S| \geq |V(G)| - n^{1-\epsilon}$, for some precision parameter $0 < \epsilon < 1$.

2.4.1 The Relationship between Routers and Well-Connected Graphs

The notion of routers is closely related to the notion of well-connected graphs. It is immediate to see that, if G is a $(\overline{w}, d, \eta)$-router with respect to any set S of supported vertices, for any vertex weighting $\overline{w}$ where $w(v) \geq 1$ for all $v \in V(G) \setminus S$, then G is $(d, O(\eta \cdot \log n))$-well-connected with respect to S (the $O(\log n)$ factor in the congestion is lost when converting a fractional flow into integral one via standard randomized rounding).

For the connection in the opposite direction, at first glance, it appears that the notion of well-connected graphs is weaker, for two reasons. The first reason is that, from the definition, well-connected graphs only provide routings between pairs A, B of vertex subsets, and it appears that one cannot control the specific pairs that are being routed. The second reason is that they only guarantee the routing of $|A| = |B|$ flow units from A to B, which roughly corresponds to routing with vertex weighting $\overline{\Delta}$ where $\Delta = 1$. We first address the former issue, and show, in the following claim, that in fact a (d,η)-well-connected graph is a sufficiently strong router. We do not use this claim directly, but we feel that it is useful in that it illuminates the relationship between these two objects. The proof of the claim is deferred to Section A of Appendix.

Claim 2.6 *Let G be an n-vertex graph, and assume that it is an (d,η)-well-connected graph with respect to a set S of supported vertices. Then for all $1 \leq k \leq \log n$, G is a $(1, d', \eta')$-router with respect to S, for $d' = 512k \cdot d$ and $\eta' = \left(2^{17} \cdot kdn^{1/k} \log^2 n\right) \cdot \eta$.*

There are three approaches to address the second issue that is raised above – namely, that the routing guarantees provided by well-connected graphs correspond to vertex weighting $w(v) = 1$ for all vertices v. The first approach has to do with the manner in which well-connected graphs are used. Most if not all applications of such graphs known today construct a well-connected graph W that is embedded into a given input graph G via the *distanced-matching game*, that can be thought of as the analogue of the cut-matching game for constructing a well-connected graph. From the construction, graph W has a rather low degree, so it can be thought of as providing routing for the weighting $\overline{\deg}_W$ of its vertices. Moreover, in the typical application of well-connected graphs as described above, the embedding of W into G is usually then used as a certificate that G itself is a well-connected graph, and is thus a router with respect to vertex weighting $w(v) = 1$ for all vertices v, from Claim 2.6. In order to obtain a certificate of routability for a general weighting $\overline{w}$ of the vertices of G, we can construct a larger graph W, containing $\sum_{v \in V(G)} w(v)$ vertices, and then embed it into G via the distanced-matching game, where, for every vertex $v \in V(G)$, we embed $w(v)$ vertices of W into G. Such an embedding

would then serve as a certificate that G is a router for the vertex weighting $\overline{w}$. The second approach is that, given any graph H with arbitrary integral vertex weighting $\overline{w}$, we can obtain a new graph H' via the following simple transformaiton: for every vertex $v \in V(H)$, add a new collection $T(v)$ of $w(v)$ vertices to the graph, connecting them to v with new edges. Let $S = \bigcup_{v \in V(H)} T(v)$. Then if graph H' is well-connected with respect to the set S of supported vertices, the original graph H is a router with respect to vertex weighting $\overline{w}$. Lastly, the third approach is to generalize the definition of well-connected graphs directly to arbitrary vertex weighting. In view of the first two approaches, it is not clear to us if such a generalization provides an additional value.

Algorithmic tools for well-connected graphs. The work of [Chu23] provides a fast algorithm, that, given a graph G and a set T of its vertices called terminals, either computes a well-connected graph X that is defined over a large subset of T, together with its embedding into G, or returns two large sets $T_1, T_2 \subseteq T$ of terminals, together with a relatively small set E' of edges, such that $\mathsf{dist}_{G \setminus E'}(T_1, T_2)$ is large. Another algorithm that [Chu23] provides, and that we will exploit, is for decremental APSP in well-connected graphs. Given a graph X that undergoes an online sequence of edge deletions, and a set S of its vertices, such that X is well-connected with respect to S, the algorithm maintains a large subset $S' \subseteq S$ of vertices, and supports $\mathsf{short\text{-}path}$ queries between vertices of S': given a pair $x, y \in V(S')$ of such vertices, it returns a path P connecting x to y in X, such that the length of P is small, and the time required to respond to the query is $O(|E(P)|)$. However, the algorithm for decremental APSP in well-connected graphs requires one additional input, called a *Hierarchical Support Structure*, for graph X. Intuitively, Hierarchical Support Structure is a hierarchy of well-connected graphs that are embedded into each other, with graph X being the topmost graph in the hierarchy. The algorithm for embedding a well-connected graph into a given graph G, that we mentioned above, in case it produces a well-connected graph X and its embedding into G, also returns the required Hierarchical Support Structure for graph X, that can then be used by the algorithm for the APSP problem on the well-connected graph X. Therefore, the specifics of the definition of the Hierarchical Support Structure are not important for us: the algorithm for embedding a well-connected graph will produce exactly the kind of Hierarchical Support Structure that the algorithm for APSP needs to use. But for completeness, we provide the definition of the Hierarchical Support Structure from [Chu23] here.

Hierarchical Support Structure. The Hierarchical Support Structure uses two main parameters: the base parameter $N > 0$, and a level parameter $j > 0$. We also assume that we are given a precision parameter $0 < \epsilon < 1$. The notion of Hierarchical Support Structure is defined inductively, using the level parameter j. If X is a graph containing N vertices, then a level-1 Hierarchical Support Structure for X simply consists of a set $S(X)$ of vertices of X, with $|V(X) \setminus S(X)| \leq N^{1-\epsilon^4}$. Assume now that we are given a graph X containing exactly N^j vertices. A level-j Hierarchical Support Structure for X consists of a collection $\mathcal{H} = \{X_1, \ldots, X_r\}$ of $r = N - \left\lceil 2N^{1-\epsilon^4} \right\rceil$ graphs, such that for all $1 \leq i \leq r$, $V(X_i) \subseteq V(X)$; $|V(X_i)| = N^{j-1}$; and $|E(X_i)| \leq N^{j-1+32\epsilon^2}$. We also require that $V(X_1), \ldots, V(X_r)$ are all mutually disjoint. Additionally, it must contain, for all $1 \leq i \leq r$, a level-$(j-1)$ Hierarchical Support Structure for X_i, which in turn must define the set $S(X_i)$ of supported vertices for graph X_i. We require that each such graph X_i is $(\tilde{d}_{j-1}, \eta_{j-1})$-well-connected with respect to $S(X_i)$, where $\tilde{d}_{j-1} = 2^{\tilde{c}(j-1)/\epsilon^4}$ for some constant $\tilde{c}$, and $\eta_{j-1} = N^{6+256(j-1)\epsilon^2}$. Lastly, the Hierarchical Support Structure for graph X must contain an embedding of graph $X' = \bigcup_{i=1}^{r} X_i$ into X, via path of length at most $2^{O(1/\epsilon^4)}$, that cause congestion at most $N^{O(\epsilon^2)}$. We then set $S(X) = \bigcup_{i=1}^{r} S(X_i)$, and we view $S(X)$ as the set of supported vertices for graph X, that is defined by the Hierarchical Support Structure.

Embedding of a well-connected graph. We will use the following theorem from [Chu23].

Theorem 2.7 (Corollary 5.3 from [Chu23]) *There is a deterministic algorithm, whose input consists of an n-vertex graph G, a set T of q vertices of G called terminals, and parameters $\frac{2}{(\log q)^{1/12}} < \epsilon < \frac{1}{400}$, $d > 1$ and $\eta > 1$, such that $1/\epsilon$ is an integer. The algorithm computes one of the following:*

- *either a pair $T_1, T_2 \subseteq T$ of disjoint subsets of terminals, and a set E' of edges of G, such that:*
  - *$|T_1| = |T_2|$ and $|T_1| \geq \frac{q^{1-4\epsilon^3}}{4}$;*
  - *$|E'| \leq \frac{d \cdot |T_1|}{\eta}$; and*
  - *for every pair $t \in T_1, t' \in T_2$ of terminals, $\mathsf{dist}_{G \setminus E'}(t, t') > d$;*
- *or a graph X with $V(X) \subseteq T$, $|V(X)| = N^{1/\epsilon}$, where $N = \lfloor q^{\epsilon} \rfloor$, and maximum vertex degree at most $q^{32\epsilon^3}$, together with an embedding $\mathcal{P}$ of X into G via paths of length at most d that cause congestion at most $\eta \cdot q^{32\epsilon^3}$, and a level-$(1/\epsilon)$ Hierarchical Support Structure for X, such that X is $(\tilde{d}, \eta')$-well-connected with respect to the set $S(X)$ of vertices defined by the support structure, where $\eta' = N^{6+256\epsilon}$, and $\tilde{d} = 2^{\tilde{c}/\epsilon^5}$, where $\tilde{c}$ is the constant used in the definition of the Hierarchical Support Structure.*

The running time of the algorithm is $O\left(q^{1+O(\epsilon)} + |E(G)| \cdot q^{O(\epsilon^3)} \cdot (\eta + d \log n)\right)$.

APSP in Well-Connected Graphs. Lastly, we need an algorithm for decremental APSP in well-connected graphs from [Chu23]. Assume that we are given a graph X that is obtained from Theorem 2.7. In other words, we are given a level-$(1/\epsilon)$ Hierarchical Support Structure for X, together with a large set $S(X)$ of vertices of X, so that X is well-connected with respect to $S(X)$. We then assume that graph X undergoes a sequence of edge deletions. As edges are deleted from X, the well-connectedness property may no longer hold, and the Hierarchical Support Structure may be partially destroyed. Therefore, we only require that the algorithm maintains a large enough subset $S'(X) \subseteq S(X)$ of supported vertices, and that it can respond to short-path queries between pairs of vertices in $S'(X)$: given a pair x, y of such vertices, the algorithm needs to return a path of length at most $2^{O(1/\epsilon^6)}$ in the current graph X connecting them. We also require that the set $S'(X)$ is *decremental*, so vertices can leave this set but they may not join it. The algorithm is summarized in the following theorem.

Theorem 2.8 (Theorem 2.3 in [Chu23]) *There is a deterministic algorithm, whose input consists of:*

- *a parameter $0 < \epsilon < 1/400$, so that $1/\epsilon$ is an integer;*
- *an integral parameter N that is sufficiently large, so that $\frac{N^{\epsilon^4}}{\log N} \geq 2^{128/\epsilon^6}$ holds;*
- *a graph X with $|V(X)| = N^{1/\epsilon}$; and*
- *a level-$(1/\epsilon)$ Hierarchical Support Structure for X, such that X is $(\tilde{d}, \eta')$-well-connected with respect to the set $S(X)$ of vertices defined by the Hierarchical Support Structure, where $\tilde{d}$ and η' are parameters from Theorem 2.7.*

Further, we assume that graph X undergoes an online sequence of at most $\Lambda = |V(X)|^{1-10\epsilon}$ edge deletions. The algorithm maintains a set $S'(X) \subseteq S(X)$ of vertices of X, such that, at the beginning of the algorithm, $S'(X) = S(X)$, and over the course of the algorithm, vertices can leave $S'(X)$ but they may not join it. The algorithm ensures that $|S'(X)| \geq \frac{|V(X)|}{2^{4/\epsilon}}$ holds at all times, and it supports short-path queries between vertices of $S'(X)$: given a pair $x, y \in S'(X)$ of vertices, return a path P connecting x to y in the current graph X, whose length is at most $2^{O(1/\epsilon^6)}$, in time $O(|E(P)|)$. The total update time of the algorithm is $O(|E(X)|^{1+O(\epsilon)})$.

2.5 Length-Constrained Expanders

Length-constrained expanders were initially introduced in [HRG22] as a generalization of the standard notion of expanders, that allows routing via paths of sublogarithmic length; this was also the main motivation behind the well-connected graphs introduced in [Chu23]. We do not use length-constrained expanders directly (except that we use routers that can be viewed as bounded-diameter length-constrained expaners). We provide a short discussion here for completeness.

Consider any graph G, and a demand $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ for G. We say that the demand is *h-length* if, for every pair $(a,b) \in \Pi$ of vertices, $\mathsf{dist}_G(a,b) \leq h$. The following description of length-constrained expanders follows the summary in [HHG22]. The formal definition of (h,s)-length φ-expanders relies on the notion of so-called "moving cuts", and can be found in [HRG22]. Instead, as in prior work on graph spanners (e.g., [HHT23] and [BHP24]), we use the more convenient *routing characterization* of length-constrained expanders. Informally, up to a constant factor in the length parameter, and a poly-logarithmic factor in the congestion parameter, a graph G is an (h,s)-hop φ-expander for a vertex weighting $\overline{w}$, if any h-length $\overline{w}$-bounded demand $\mathcal{D}$ can be routed in G via paths of length at most $(s \cdot h)$, with congestion roughly $\frac{1}{\varphi}$. The precise characterization of length-constrained expanders in terms of routing appears in the following theorem.

Theorem 2.9 (Theorem 5.17 of [HHT24]) *Let G be a graph, let $\overline{w}$ be a weighting of its vertices, and let $h \geq 1$, $0 < \varphi < 1$, and $s \geq 1$ be parameters. Then:*

1. *if G is an (h,s)-length φ-expander for vertex weighting $\overline{w}$, then every h-length $\overline{w}$-restricted demand $\mathcal{D}$ can be routed in G via paths of length at most $(s \cdot h)$, with congestion $O\left(\frac{\log(n)}{\varphi}\right)$; and*
2. *if G is not an (h,s)-length φ-expander for vertex weighting $\overline{w}$, then there is some h-length $\overline{w}$-bounded demand $\mathcal{D}$ that cannot be routed in G via paths of length at most $(\frac{s}{2} \cdot h)$ with congestion at most $\frac{1}{2\varphi}$.*

The next claim summarized a bidirectional connection between routers and length-restricted expanders that was shown by [HRG22, HHT24], and also follows from Theorem 2.9.

Claim 2.10 (Immediate from Lemma 5.24 in [HHT24]) *If G is an (h,s)-length φ-expander for vertex weighting $\overline{w}$ with $\mathsf{diam}(G) \leq h$, then G is a $\left(\overline{w}, hs, O\left(\frac{\log n}{\varphi}\right)\right)$-router for the set $S = V(G)$ of supported vertices. Conversely, if an G is a graph of diameter at most h that is not an (h,s)-length φ-expander for vertex weighting $\overline{w}$, then G is not a $\left(\overline{w}, \frac{hs}{2}, O\left(\frac{1}{2\varphi}\right)\right)$-router.*

To summarize, one can intuitively think of a $(\overline{\deg}, d, \eta)$-router as being roughly equivalent to a bounded-diameter length-constrained φ-expander, where $\varphi = \tilde{\Theta}(1/\eta)$, and the bound on the diameter is roughly d.

2.6 Even-Shiloach Trees

Suppose we are given a graph $G = (V, E)$ with integral lengths $\ell(e) \geq 1$ on its edges $e \in E$, a source s, and a distance bound $d \geq 1$. The Even-Shiloach Tree (ES-Tree) algorithm of [ES81, Din06, HK95] maintains, for every vertex $v \in V$ with $\mathsf{dist}_G(s, v) \leq d$, the distance $\mathsf{dist}_G(s, v)$, under the deletion of edges from G. It also maintains a shortest-path tree τ of G rooted at s, that includes all vertices v with $\mathsf{dist}_G(s, v) \leq d$. We denote the corresponding data structure by $\mathsf{ES\text{-}Tree}(G, s, d)$. The total update time of the algorithm is $O(m \cdot d \cdot \log n)$, where m is the initial number of edges in G and $n = |V|$.

2.7 The Online-Batch Dynamic Model and Deamortization

Our algorithm uses the by now standard online-batch dynamic model, and a reduction from this model to the standard dynamic model with bounded worst-case update time, that was formalized in [JS22]. Much of the exposition in this subsection is borrowed from Section 10.1 of [JS22]

We assume that we are given two main parameters: batch number σ and sensitivity parameter w. We let DS be a data stucture that we would like to maintain for an input graph G undergoing updates. A dynamic algorithm for maintaining DS in the online-batch dynamic model consists of $\sigma + 1$ phases $\Phi_0, \ldots, \Phi_\sigma$, where Φ_0 is the *preprocessing*, or the *initialization* phase, and $\Phi_1, \ldots, \Phi_\sigma$ are *regular* phases. For all $1 \leq i \leq \sigma$, at the beginning of Phase Φ_i, the algorithm is given the ith batch Π_i of updates to graph G; in our case, all updates are insertions and deletions of edges to and from G. The number of such updates in each batch Π_i must be bounded by w. The algorithm is required to maintain the data structure DS, so that, at the end of the 0th phase, the data structure DS that it obtains correctly corresponds to the initial graph $G^{(0)}$. Additionally, for all $1 \leq i \leq \sigma$, the data structure DS that the algorithm obtains at the end of Phase Φ_i must correctly correspond to the graph $G^{(i)}$, that is obtained from the initial graph $G^{(0)}$, after the sequence $\Pi_1 \circ \cdots \circ \Pi_i$ of updates is applied to it. The algorithm has amortized time t, if, for all $1 \leq i \leq \sigma$, the running time of the algorithm during Phase Φ_i is bounded by $t \cdot |\Pi_i|$. Here, t is allowed to be some function of the graph parameters, such as, for example, n^ϵ, where $n = |V(G)|$, and ϵ a given parameter.

The following lemma was formally proved in [JS22], and is implied by the previous work of [NSWN17]. We provide a slightly simplified version of the lemma that is sufficient for us.

Lemma 2.11 (Lemma 10.1 in [JS22]; see also Section 5 in [NSWN17]) *Let G be a graph undergoing batch updates, and let $\sigma > 0$ and $w \geq 2 \cdot 6^\sigma$ be parameters. Assume that there is an algorithm in the batch update model with batch number σ and sensitivity parameter w, that maintains some data structure* DS *for G, whose preprocessing time is T_0, and amortized update time is T', where both T_0 and T' are functions that map some graph measure (e.g. $|E(G^{(0)})|$ or $|V(G)|$) to non-negative numbers.*

Then there is a dynamic algorithm, with preprocessing time $O\left(2^{O(\sigma)} \cdot T_0\right)$, and worst case update time $O\left(4^\sigma \cdot \left(\frac{T_0}{w} + w^{1/\sigma} \cdot T'\right)\right)$, that maintains a set of $O(2^{O(\sigma)})$ instances of the data structure DS*, such that, after each update, the algorithm indicates one of the maintained instances of the data structure as the "current" one. The current instance of data structure* DS *satisfies the following conditions:*

- *it is up-to-date with respect to the current graph; and*
- *the online-batch update algorithm underwent at most σ batches of updates, with each update batch size at most w.*

2.8 Vertex-Split Graphs

Some of our subroutines work best when all degrees in the given graph are close to each other. In order to extend such subroutines to general graphs, we will employ *vertex-split graphs*.

Consider a graph $G = (V, E)$. A *vertex split operation* in G is defined as follows. The input to the vertex split operation is a vertex $v \in V$, and a subset $E_1 \subseteq \delta_G(v)$ of its incident edges. In order to execute the split operation, we insert a new vertex v' into G, that we call a *copy* of v. For every edge $e = (u, v) \in E_1$, we then delete e from G, and insert the edge (u, v'), that we call a *copy of* e into G. We will not distinguish between the original edges of G and their copies in the resulting graph. We are now ready to define vertex-split graphs.

Definition 2.12 *Let G and G' be two graphs. We say that G' is a* vertex-split graph for G, *if G' can be obtained from G via a sequence of vertex split operations.*

Next, we provide a simple claim that allows us to produce a vertex-split graph for a given graph G, in which no vertex degree is too high.

Claim 2.13 *There is a deterministic algorithm, that, given a graph G with average vertex degree Δ, computes a vertex-split graph G' for G, such that the degrees of all vertices in G' are at most 2Δ, and $|V(G')| \leq 2|V(G)|$. The running time of the algorithm is $O(|E(G)|)$.*

Proof: Let $S \subseteq V(G)$ be the set of all vertices v with $\deg_G(S) > 2\Delta$. Our algorithm processes every vertex $v \in S$ one by one. In order to process a vertex $v \in S$, we perform iterations, as long as $\deg_G(v) > 2\Delta$. In every iteration, we select an arbitrary subset $E_1 \subseteq \delta_G(v)$ of Δ edges, and then perform a vertex splitting of v using E_1. Notice that the splitting operation can be executed in time $O(|E_1|)$. We then continue to the next iteration. It is easy to verify that the number of new vertices inserted into G while processing v is at most $\frac{\deg_G(v)}{\Delta}$.

Once all vertices in S are processed, we obtain the final graph G', that is a vertex-split graph for G. Clearly, all vertex degrees in G' are bounded by 2Δ. Since $\sum_{v \in S} \deg_G(v) \leq \Delta \cdot |V(G)|$, the total number of new vertices inserted into G' is bounded by $|V(G)|$, and so $|V(G')| \leq 2|V(G)|$. Lastly, it is easy to verify that the running time of the algorithm is $O(|E(G)|)$. □

Next, we obtain the following immediate observation.

Observation 2.14 *Let G is a graph, and let G' be a vertex-split graph for G. Assume that G' is a $(\overline{\deg}_{G'}, d, \eta)$-router, for some parameters d and η, with respect to the set $V(G')$ of supported vertices. Then G is a $(\overline{\deg}_G, d, \eta)$-router with respect to the set $V(G)$ of supported vertices.*

Proof: For every vertex $v \in V(G)$, let $S(v)$ be the collection of its copies in G'. Recall that $\sum_{v' \in S(v)} \deg_{G'}(v') = \deg_G(v)$.

Let $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b) \in \Pi}\right)$ be any $\overline{\deg}_G$-restricted demand in G. It is enough to show that it can be routed in G via paths of length at most d, with congestion at most η.

We use the demand $\mathcal{D}$, in order to define a demand $\mathcal{D}' = \left(\Pi', \{D'(a',b')\}_{(a',b') \in \Pi'}\right)$ in G', that is $\overline{\deg}_{G'}$-restricted, so that a routing of $\mathcal{D}'$ in G' via paths of length at most d and congestion at most η defines a routing of $\mathcal{D}$ in G via paths of length at most d and congestion at most η.

In order to define the demand $\mathcal{D}'$, we consider every pair $(a, b) \in \Pi$ one by one. Let $\Pi'(a, b) = S(a) \times S(b)$. For every pair $(a', b') \in \Pi'(a, b)$, we define a demand $D'(a', b') = D(a, b) \cdot \frac{\deg_{G'}(a') \cdot \deg_{G'}(b')}{\deg_G(a) \cdot \deg_G(b)}$. Notice that:

$$\sum_{(a',b')\in\Pi'(a,b)} \deg_{G'}(a')\cdot\deg_{G'}(b') = \left(\sum_{a'\in S(a)}\deg_{G'}(a')\right)\cdot\left(\sum_{b'\in S(b)}\deg_{G'}(b')\right) = \deg_G(a)\cdot\deg_G(b).$$

Therefore:

$$\sum_{(a',b')\in\Pi'(a,b)} D'(a',b') = D(a,b). \tag{1}$$

Moreover, for every vertex $a'\in S(a)$:

$$\sum_{b'\in S(b)} D'(a',b') = D(a,b)\cdot\frac{\deg_G(a')}{\deg_G(a)\cdot\deg_G(b)}\cdot\sum_{b'\in S(b)}\deg_{G'}(b') = D(a,b)\cdot\frac{\deg_{G'}(a')}{\deg_G(a)}. \tag{2}$$

Similarly, for every vertex $b'\in S(b)$, $\sum_{a'\in S(a)} D'(a',b') = D(a,b)\cdot\frac{\deg_{G'}(b')}{\deg_G(b)}$.

We are now ready to define the demand $\mathcal{D}' = \left(\Pi', \{D'(a',b')\}_{(a',b')\in\Pi'}\right)$ for G': we set $\Pi' = \bigcup_{(a,b)\in\Pi}\Pi'(a,b)$, and we set the demand $D'(a',b')$ for every pair $(a',b')\in\Pi'$ as above.

Consider now any vertex $v'\in V(G')$. Assume w.l.o.g. that in every pair $(a',b')\in\Pi'$ that contains v', $v'=a'$ holds. Then, from Inequality 2:

$$\sum_{(v',u')\in\Pi'} D'(v',u') = \sum_{\substack{(a,b)\in\Pi:\\ v'\in S(a)}} D(a,b)\cdot\frac{\deg_{G'}(v')}{\deg_G(a)} \le \deg_{G'}(v'),$$

since the demand $\mathcal{D}$ is $\overline{\deg}_G$-restricted. Therefore, demand $\mathcal{D}'$ is $\overline{\deg}_{G'}$-restricted, and so there is a routing f' of $\mathcal{D}'$ in G' via paths of length at most d, with congestion at most η. Since every edge of G' corresponds to a distinct edge of G, routing f' immediately defines a routing f of $\mathcal{D}$ via paths of length at most d, with congestion at most η. □

We will also use the following observation, whose proof is immediate.

Observation 2.15 *Let G is a graph, and let G' be a vertex-split graph for G. Assume that G' is a $(\overline{\Delta},d,\eta)$-router, for some parameters d, η and Δ, with respect to the set $V(G')$ of supported vertices. Then G is a $(\overline{\Delta},d,\eta)$-router with respect to the set $V(G)$ of supported vertices.*

2.9 Chernoff Bound

We use the following standard version of Chernoff Bound (see e.g., [DP09]).

Lemma 2.16 (Chernoff Bound) *Let $X_1,\ldots,X_n$ be independent random variables taking values in $\{0,1\}$. Let $X=\sum_{1\le i\le n}X_i$, and let $\mu=\mathbf{E}[X]$. Then for any $t>2e\mu$,*

$$\Pr\left[X>t\right]\le 2^{-t}.$$

3 Low-Diameter Router Decomposition

In this section we formally define one of our main combinatorial objects – a Low-Diameter Router Decomposition, that, for conciseness, we refer to as "Router Decomposition". We then provide the statement of our main technical result, namely, an algorithm that maintains such a decomposition in the online-batch dynamic model, and show that the proof of Theorem 1.3 follows from it. We complete the proof of the main technical result in Sections 4–8. We start with a formal definition of a router decomposition.

Definition 3.1 (Router Decomposition of a Graph) *Let G be a simple graph, and let $\Delta^*, \tilde{d}, \tilde{\eta}$ and ρ be non-negative parametes. A* router decomosition *of G with parameters $\Delta^*, \tilde{d}, \tilde{\eta}$ and ρ consists of the following ingredients:*

- *a collection $\mathcal{C}$ of subgraphs of G called* clusters *with $\sum_{C\in\mathcal{C}} |V(C)| \leq \rho \cdot |V(G)|$, such that every cluster $C \in \mathcal{C}$ is a $(\overline{\deg}_C, \tilde{d}, \tilde{\eta})$-router with respect to the set $V(C)$ of supported vertices, and every edge of G lies in at most one cluster $C \in \mathcal{C}$; and*
- *for every cluster $C \in \mathcal{C}$, a subgraph $C' \subseteq C$ with $V(C') = V(C)$, such that C' is a $(\overline{\Delta}^*, \tilde{d}, \tilde{\eta})$-router for the set $V(C)$ of supported vertices, and $\sum_{C\in\mathcal{C}} |E(C')| \leq \rho \cdot \Delta^* \cdot |V(G)|$.*

We denote by $E^{\mathrm{del}} = E(G) \setminus \left(\bigcup_{C\in\mathcal{C}} E(C)\right)$ the set of all edges of G that do not lie in any cluster of $\mathcal{C}$.

We note that generally we will ensure that $|E^{\mathrm{del}}|$ is sufficiently small in a router decomposition. The following theorem summarizes our main technical result, and a significant part of the paper is devoted to proving it.

Theorem 3.2 *There is a deterministic algorithm, that receives as input a parameter n that is greater than a large enough constant, and additional parameters $512 \leq k \leq (\log n)^{1/49}$, $\frac{1}{k} \leq \delta < \frac{1}{400}$, Δ^*, and Δ, such that $n^{18/k} \leq \Delta^* \leq \frac{\Delta}{n^{8/k^2}} \leq n$ holds, and k, Δ, Δ^* and $1/\delta$ are integers. The algorithm is also given as input a simple graph G with $|V(G)| \leq n$, such that, initially, all vertex degrees in G are at most $\Delta \cdot n^{16/k}$. Lastly, the algorithm receives k online batches $\pi_1, \ldots, \pi_k$ of updates to graph G, where for $1 \leq i \leq k$, the ith batch π_i is a collection of at most $\Delta \cdot n$ edges that are deleted from G.*

The algorithm maintains a router decomposition $\left(\mathcal{C}, \{C'\}_{C\in\mathcal{C}}\right)$ of the graph G, for parameters Δ^*, $\tilde{d} = k^{19} \cdot 2^{O(1/\delta^6)}$, $\tilde{\eta} = n^{17/k}$, and $\rho = n^{20/k}$. After the initialization, $|E^{\mathrm{del}}| \leq |V(G)| \cdot \Delta \cdot n^{2/k^2}$ holds, and, for all $1 \leq i \leq k$, at most $|\pi_i| \cdot n^{13/k^2}$ edges may be added to E^{del} while processing the batch π_i. For all $1 \leq i \leq k$, after the ith batch of updates is received, the clusters $C \in \mathcal{C}$ may only be updated by removing some of their edges (which are then added to E^{del}) and some of their vertices. The corresponding graphs $C' \subseteq C$ may be updated by removing or inserting edges into them, and by removing vertices from them. Additionally, clusters of $\mathcal{C}$ that become empty are removed from $\mathcal{C}$. These are the only updates to the router decomposition. The initialization time of the algorithm is $O\left((n\Delta)^{1+O(\delta)}\right)$, and for all $1 \leq i \leq k$, the time required to process the ith update batch π_i is bounded by $O\left(|\pi_i| \cdot n^{O(1/k^2)}\right)$. The total number of edges inserted into or deleted from all subgraphs in $\{C' \mid C \in \mathcal{C}\}$ while batch π_i is processed is at most $O\left(|\pi_i| \cdot n^{O(1/k^2)}\right)$.

We will prove Theorem 3.2 in sections 4–8. In the remainder of this section, we complete the proof of Theorem 1.3 using it. We start with the following simple corollary of Theorem 3.2; its main difference from the theorem is that we only require that the total number of edges in G is bounded by $\Delta \cdot n^{1+15/k}$, while the degrees of individual edges may be higher. Other than this and slight changes

to the input parameters, the corollary is essentially identical to Theorem 3.2. The corollary follows from Theorem 3.2 in a straightforward manner by employing vertex-split graphs. We defer its proof to Section B of Appendix.

Corollary 3.3 *There is a deterministic algorithm, that receives as input a simple n-vertex graph G, where n is greater than a large enough constant, and additional parameters $512 \leq k \leq (\log n)^{1/49}$, $\frac{1}{k} \leq \delta < \frac{1}{400}$, Δ^*, and $\Delta \leq n$, such that $n^{19/k} \leq \Delta^* \leq \frac{\Delta}{n^{9/k^2}}$ holds, $|E(G)| \leq \Delta \cdot n^{1+15/k}$, and k, Δ^* and $1/\delta$ are integers. Lastly, the algorithm receives k online batches $\pi_1, \ldots, \pi_k$ of updates to graph G, where for $1 \leq i \leq k$, the ith batch π_i is a collection of at most $\Delta \cdot n$ edges that are deleted from G.*

The algorithm maintains a router decomposition $\left(\mathcal{C}, \{C'\}_{C \in \mathcal{C}}\right)$ of the graph G, for parameters Δ^*, $\tilde{d} = k^{19} \cdot 2^{O(1/\delta^6)}$, $\tilde{\eta} = n^{19/k}$, and $\rho = n^{21/k}$. After the initialization step, $|E^{\mathrm{del}}| \leq n^{1+4/k^2} \cdot \Delta$ holds, and, for all $1 \leq i \leq k$, at most $|\pi_i| \cdot n^{15/k^2}$ edges may be added to E^{del} while processing the batch π_i. For all $1 \leq i \leq k$, after the ith batch of updates is received, the clusters $C \in \mathcal{C}$ may only be updated by removing some of their edges (which are then added to E^{del}) and some of their vertices. The corresponding graphs $C' \subseteq C$ may be updated by removing or inserting edges into them, and by removing vertices from them. Additionally, clusters of $\mathcal{C}$ that become empty are removed from $\mathcal{C}$. These are the only updates to the router decomposition. The initialization time of the algorithm is $O\left((n\Delta)^{1+O(\delta)}\right)$, and for all $1 \leq i \leq k$, the time required to process the ith update batch π_i is bounded by $O\left(|\pi_i| \cdot n^{O(1/k^2)}\right)$. The total number of edges inserted into or deleted from all subgraphs in $\{C' \mid C \in \mathcal{C}\}$ while batch π_i is processed is at most $O\left(|\pi_i| \cdot n^{O(1/k^2)}\right)$.

Next, we prove a corollary of Corollary 3.3, where we remove the bound on the number of edges in G. Other than that, the corollary is very similar to Corollary 3.3, and can be thought of as an analogue of our main technical result – Theorem 1.3, for the online-batch dynamic model.

Corollary 3.4 *There is a deterministic algorithm, that receives as input a simple n-vertex graph G, where n is greater than a large enough constant, and additional parameters $512 \leq k \leq (\log n)^{1/49}$, $\frac{1}{k} \leq \delta < \frac{1}{400}$ and $\Delta^* \geq n^{19/k}$, such that k, Δ^* and $1/\delta$ are integers. Lastly, the algorithm receives k online batches $\pi_1, \ldots, \pi_k$ of updates to graph G, where for $1 \leq i \leq k$, the ith batch π_i is a collection of edge insertions and deletions for G; graph G is guaranteed to remain simple throughout the update sequence.*

The algorithm maintains a router decomposition $\left(\mathcal{C}, \{C'\}_{C \in \mathcal{C}}\right)$ of the graph G, for parameters Δ^*, $\tilde{d} = k^{19} \cdot 2^{O(1/\delta^6)}$, $\tilde{\eta} = n^{19/k}$, and $\rho = n^{O(1/k)}$, such that $|E^{\mathrm{del}}| \leq n^{1+O(1/k)} \cdot \Delta^*$ holds. The algorithm also maintains a graph H with $V(H) = V(G)$ and $E(H) = E^{\mathrm{del}} \cup \left(\bigcup_{C \in \mathcal{C}} E(C')\right)$, with $|E(H)| \leq n^{1+O(1/k)} \cdot \Delta^*$. The initialization time of the algorithm is $O\left(m^{1+O(\delta)}\right)$, where m is the initial number of edges and vertices in G. For all $1 \leq i \leq k$, the time required to process the ith update batch π_i is bounded by $O\left(|\pi_i| \cdot n^{O(\delta)}\right)$. The number of edge insertions and deletions that graph H undergoes while batch π_i is processed is bounded by $O\left(|\pi_i| \cdot n^{O(1/k)}\right)$.

Proof: Let z be the smallest integer, so that $\Delta^* \cdot n^{z/k} \geq 2n$. Clearly, $z \leq k$ must hold. For $1 \leq j \leq z+1$, let $\Delta_j = \Delta^* \cdot n^{j/k}$. Notice that all vertex degrees in G are bounded by Δ_z at all times, and $\Delta_1 = \Delta^* \cdot n^{1/k} \geq n^{9/k^2} \cdot \Delta^*$. Throughout the algorithm, we maintain subgraphs $G_1, \ldots, G_z$ of G, where $V(G_1) = \cdots = V(G_z) = V(G)$, and every edge of G lies in exactly one of the graphs $G_1, \ldots, G_z$. We will ensure that, for all $1 \leq j \leq z$, for all $1 \leq i \leq k$, just before the ith batch π_i of updates arrives, $|E(G_j)| \leq i \cdot \Delta_j \cdot n$ holds. For all $3 \leq j \leq z$, we also maintain a router decomposition $\left(\mathcal{C}_j, \{C'\}_{C \in \mathcal{C}_j}\right)$ of graph G_j with parameters Δ^*, $\tilde{d} = k^{19} \cdot 2^{O(1/\delta^6)}$, $\tilde{\eta} = n^{19/k}$, and $\rho' = n^{21/k}$, such

that $\bigcup_{C\in\mathcal{C}_j} E(C) = E(G_j)$. We refer to the graph G_j and its router decomposition as *level-j data structure* (for levels 1 and 2, the correponding data structure only contains the graph G_j).

We will then let $\mathcal{C} = \bigcup_{j=3}^{z} \mathcal{C}_j$, obtaining a router decomposition $\left(\mathcal{C}, \{C'\}_{C\in\mathcal{C}}\right)$ of graph G with parameters Δ^*, $\tilde{d} = k^{19}\cdot 2^{O(1/\delta^6)}$, $\tilde{\eta} = n^{19/k}$, and $\rho = k\cdot\rho' \leq n^{O(1/k)}$. Notice that the edges of $E^{\mathrm{del}} = E(G)\setminus\left(\bigcup_{C\in\mathcal{C}} E(C)\right)$ are precisely the edges of $E(G_1)\cup E(G_2)$, and so $|E^{\mathrm{del}}| \leq |E(G_1)| + |E(G_2)| \leq 2nk\Delta_2 \leq n^{1+O(1/k)}\cdot\Delta^*$ holds throughout the algorithm.

The main subroutine that we use is the initialization of the data structures from levels $1,\ldots,z$. Consider some level $3\leq j\leq z$, and assume that we are given a graph G'_j that contains at most $kn\Delta_j$ edges. In order to initialize the level-j data structure, we initialize the algorithm from Corollary 3.3 on graph G'_j, with parameters k,δ and Δ^* remaining unchanged, and parameter Δ replaced with Δ_{j-2}. Since $j\geq 3$ and $\Delta_{j-2} = \Delta^*\cdot n^{(j-2)/k} \geq \Delta^*\cdot n^{1/k}$, it is easy to verify that $\Delta^* \leq \frac{\Delta}{n^{9/k^2}}$, as required, and, from the definition of z, $\Delta_{j-2}\leq\Delta_{z-2}<n$. Moreover, $|E(G'_j)|\leq kn\Delta_j \leq kn^{1+2/k}\cdot\Delta_{j-2}\leq n^{1+15/k}\cdot\Delta$, as required. The algorithm from Corollary 3.3 then computes a router decomposition $\left(\mathcal{C},\{C'\}_{C\in\mathcal{C}}\right)$ of the graph G'_j, for parameters Δ^*, $\tilde{d}$, $\tilde{\eta}$, and $\rho' = n^{21/k}$. We then define the graph G_j to contain the same vertex set as G'_j, and all edges that lie in the clusters of $\mathcal{C}$. Let $\tilde{E}_j$ denote the resulting set $E^{\mathrm{del}} = E(G'_j)\setminus E(G_j)$ of edges. Then we are guaranted that:

$$|\tilde{E}_j| \leq n^{1+4/k^2}\cdot\Delta = n^{1+4/k^2}\cdot\Delta_{j-2} \leq n^{1+4/k^2+(j-2)/k}\cdot\Delta^* \leq n^{1+(j-1)/k}\cdot\Delta^* = n\cdot\Delta_{j-1}.$$

We then construct a graph G'_{j-1}, with $V(G'_{j-1}) = V(G'_j)$, and $E(G'_{j-1}) = \tilde{E}_j$. If $j=3$, then we set $G_{j-1} = G'_{j-1}$, and terminate the initialization algorithm. Otherwise, we call the initialization algorithm for level $(j-1)$ with the graph G'_{j-1}. Notice that the time required to initialize the level-j data structure (excluding the time spent on recursive calls to initialize data structures from lower levels) is bounded by $O\left((kn\Delta_j)^{1+O(\delta)}\right)\leq O\left((n\Delta_j)^{1+O(\delta)}\right)$. We are now ready to complete the proof of Corollary 3.4. We start by describing the initialization algorithm, followed by the algorithm for processing the batches π_i of edge deletions.

Initialization. Consider the initial graph G, and let $m = |E(G)| + |V(G)|$. Let $z'\geq 1$ be the smallest integer, for which $m\leq n\cdot\Delta_{z'}$ holds. It is easy to verify that $z'\leq z$ must hold, and moreover, $n\cdot\Delta_{z'}\leq mn^{1/k}\leq m^{1+1/k}$. For all for all $z'<j\leq z$, the graph G_j is initialized to contain n vertices and no edges. If $z'<3$, then we let $G_{z'} = G$. Otherwise, we apply the initialization algorithm described above to graph $G'_{z'} = G$ in order to initialize level z'. The algorithm then recursively initializes levels $z'-1,\ldots,1$. Notice that $|E(G'_{z'})|\leq n\Delta_{z'}$, and our initialization algorithm ensures that, for all $1\leq j\leq z'$, $|E(G'_j)|\leq n\cdot\Delta_j$ holds. We then let $\mathcal{C} = \bigcup_{j=3}^{z}\mathcal{C}_j$, obtaining a router decomposition $\left(\mathcal{C},\{C'\}_{C\in\mathcal{C}}\right)$ of graph G with parameters Δ^*, $\tilde{d}$, $\tilde{\eta}$, and ρ as required, where $E^{\mathrm{del}} = E(G)\setminus\left(\bigcup_{C\in\mathcal{C}}E(C)\right) = E(G_1)\cup E(G_2)$, and so $|E^{\mathrm{del}}|\leq n^{1+O(1/k)}\cdot\Delta^*$ holds. From our discussion, it is easy to verify that the total time required for the initialization algorithm is bounded by:

$$\sum_{j=1}^{z'} O\left((n\Delta_j)^{1+O(\delta)}\right)\leq O\left((n\Delta_{z'})^{1+O(\delta)}\right)\leq O\left(m^{1+O(\delta)}\right)$$

since $\delta\geq 1/k$. We also initialize the graph H with $V(H) = V(G)$ and $E(H) = E^{\mathrm{del}}\cup\left(\bigcup_{C\in\mathcal{C}}E(C')\right)$. It is easy to verify that graph H can be initialized without increasing the asymptotic running time of the algorithm. Additionally, since, for all $3\leq j\leq z'$, $\sum_{C\in\mathcal{C}_j}|E(C')|\leq n^{1+21/k}\cdot\Delta^*$, while $|E^{\mathrm{del}}|\leq$

$n^{1+O(1/k)}\Delta^*$, we get that $|E(H)| \leq n^{1+O(1/k)} \cdot \Delta^*$ holds.

Processing update batch π_i. We now provide an algorithm for processing the ith batch π_i of updates to graph G. Assume first that $|\pi_i| \geq n^{2-4/k}$. Let $G^{(i)}$ be the graph obtained from G after applying the first i batches of updates to it. Since $G^{(i)}$ is guaranteed to be a simple graph, $|E(G^{(i)}|) \leq n^2 \leq |\pi_i| \cdot n^{4/k}$. In this case, we simply initialize all data structures from scratch, using the initialization algorithm described above. The time required to process the batch π_i is then bounded by:

$$O\left((|E(G^{(i)})| + n)^{1+O(\delta)}\right) \leq O\left(|\pi_i| \cdot n^{4/k}\right)^{1+O(\delta)} \leq O(|\pi_i| \cdot n^{O(\delta)}),$$

since $\delta \geq 1/k$. In order to update the graph H, we initially delete all edges from it, and then add the edges obtained from initializing the graph H as described above. The total number of edge deletions and insertions that graph H undergoes is bounded by $O(n^2) \leq |\pi_i| \cdot n^{O(1/k)}$.

From now on we assume that $|\pi_i| < n^{2-4/k}$. We let $\hat{z}$ be the smallest integer, so that $|\pi_i| \leq n \cdot \Delta_{\hat{z}-2}$. Since $\Delta_z \geq 2n$, while $|\pi_i| < n^{2-4/k}$, it is easy to verify that $n \cdot \Delta_{z-3} > |\pi_i|$, and so $\hat{z} < z$. Additionally, $|\pi_i| \geq n \cdot \Delta_{\hat{z}-3}$ must hold. Let $\hat{z}' = \max\{\hat{z}, 4\}$. For each integer $\hat{z}' \leq j \leq z$, we let π_i^j be the set of all edges of graph G_j that are deleted from G by the batch π_i of edge deletions. From our discussion, $|\pi_i^j| \leq |\pi_i| \leq n \cdot \Delta_{j-2}$ holds. We then provide π_i^j as a batch of edge deletions to the algorithm that maintains a router decomposition of graph G_j. Recall that, as the result, the clusters $C \in \mathcal{C}_j$ may be modified, with edges or vertices deleted from C. We denote by E_i^j the set of all edges that were deleted from the clusters of $\mathcal{C}_j$ during the update. Recall that the algorithm from Corollary 3.3 ensures that $|E_i^j| \leq |\pi_i^j| \cdot n^{15/k^2}$. Additionally, for every cluster $C \in \mathcal{C}_j$, the corresponding graph C' may be updated by removing or inserting edges into it, and possibly removing vertices from it. The total number of edges inserted into or deleted from all subgraphs in $\{C' \mid C \in \mathcal{C}_j\}$ is bounded by $O\left(|\pi_i^j| \cdot n^{O(1/k^2)}\right)$. We update the graph H accordingly, so that at most $O\left(|\pi_i^j| \cdot n^{O(1/k^2)}\right)$ edges are deleted from or inserted into H. The time that is required in order to process batch π_i^j of deletions from graph G_j is at most $O\left(|\pi_i^j| \cdot n^{O(1/k^2)}\right) \leq O\left(|\pi_i^j| \cdot n^{O(\delta)}\right)$. We also delete the edges of E_i^j from graph G_j, so that $E(G_j) = \bigcup_{C \in \mathcal{C}_j} E(C)$ continues to hold.

To summarize, the time required to update data structures for all levels $\hat{z}' \leq j \leq z$ is bounded by:

$$\sum_{j=\hat{z}'}^{z} O\left(|\pi_i^j| \cdot n^{O(\delta)}\right) \leq O\left(|\pi_i| \cdot n^{O(\delta)}\right).$$

Using similar reasoning, the total number of edge insertions and deletions that graph H undergoes due to updating the data structures from levels $\hat{z}' \leq j \leq z$ is at most $O\left(|\pi_i| \cdot n^{O(1/k)}\right)$.

Assume now that $\hat{z} \geq 4$, so that $\hat{z} = \hat{z}'$ holds. We construct a set E^* of edges, that consists of all edges in graphs $G_1, G_2, \ldots, G_{\hat{z}-1}$ (excluding the edges that are deleted in batch π_i); all edges that are inserted into G via the batch π_i; and all edges in sets $E_i^{\hat{z}}, \ldots, E_i^z$ that we have computed (the edges that were deleted from graphs $G_{\hat{z}}, \ldots, G_z$).

Recall that, for all $\hat{z} \leq j \leq z$, $|E_i^j| \leq |\pi_i^j| \cdot n^{15/k^2}$, and so $\sum_{j=\hat{z}}^{z} |E_i^j| \leq |\pi_i| \cdot n^{15/k^2}$. Recall also that $|\pi_i| \leq n \cdot \Delta_{\hat{z}-2}$. Lastly, from our invariants, for all $1 \leq j < \hat{z}$, $|E(G_j)| \leq i \cdot \Delta_j \cdot n$ holds. Therefore, we get that:

$$|E^*| \leq \sum_{j=1}^{\hat{z}-1} i \cdot \Delta_j \cdot n + 2|\pi_i| \cdot n^{15/k^2} \leq i \cdot \Delta_{\hat{z}-1} \cdot n + 2i \cdot \Delta_{\hat{z}-2} \cdot n + 2n^{1+18/k^2} \cdot \Delta_{\hat{z}-2} \leq (i+1) \cdot \Delta_{\hat{z}-1} \cdot n.$$

We let $G'_{\hat{z}-1}$ be the graph whose vertex set is $V(G)$, and edge set is E^*. We then initialize the level-$(\hat{z}-1)$ data structure from scratch, using the initialization algorithm described above, which also recursively initializes data structures from levels $1, \ldots, \hat{z}-2$. The running time required to initialize the data structures from levels $1, \ldots, \hat{z}-1$ is bounded by:

$$O\left((n \cdot \Delta_{\hat{z}-1})^{1+O(\delta)}\right) \leq O\left(|\pi_i| \cdot n^{O(\delta+1/k)}\right) \leq O\left(|\pi_i| \cdot n^{O(\delta)}\right),$$

since $|\pi_i| \geq n \cdot \Delta_{\hat{z}-3} \geq n^{1-2/k} \cdot \Delta_{\hat{z}-1}$.

Lastly, we delete from H all edges that originally lied in graphs C' for $C \in \bigcup_{j=3}^{\hat{z}-1} \mathcal{C}_j$ and in the original graphs G_1 and G_2, and insert edges that lie in the new graphs C' for $C \in \bigcup_{j=3}^{\hat{z}-1} \mathcal{C}_j$, and in the new graphs G_1 and G_2. Recall that, for all $3 \leq j \leq z$, $\sum_{C \in \mathcal{C}_j} |E(C')| \leq n^{1+O(1/k)} \cdot \Delta^*$ holds. Since we have assumed that $\hat{z} \geq 4$, we get that $|\pi_i| \geq n^{1-1/k} \cdot \Delta^*$, and so $\sum_{C \in \mathcal{C}_j} |E(C')| \leq n^{1+O(1/k)} \cdot \Delta^* \leq |\pi_i| \cdot n^{O(1/k)}$ holds. Therefore, the total number of updates the graph H undergoes while processing batch π_i is bounded by $|\pi_i| \cdot n^{O(1/k)}$.

Finally, if $\hat{z} < 4$, then $|\pi_i| \leq n \cdot \Delta_1$. In this case, let $\hat{E}$ be the collection of edges that includes all edges inserted into G via the batch π_i of updates, and the edges of $\bigcup_{j=\hat{z}'}^{z} E_i^j$. Since $\sum_{j=\hat{z}'}^{z} |E_i^j| \leq |\pi_i| \cdot n^{15/k^2}$, we get that $|\hat{E}| \leq 2|\pi_i| \cdot n^{15/k^2} \leq n \cdot \Delta_2$. We simply insert the edges of $\hat{E}$ into G_2 and into H, and we delete from them all edges that were deleted by π_i. This completes the algorithm for processing the batch π_i of updates. The total time required to process the batch π_i is bounded $O\left(|\pi_i| \cdot n^{O(\delta)}\right)$, and the number of edge insertions and deletions that H undergoes is bounded by $O\left(|\pi_i| \cdot n^{O(1/k)}\right)$. □

We are now ready to complete the proof of Theorem 1.3. Let G be an initial graph with n vertices and no edges, that undergoes an online sequence of at most n^2 edge deletions and insertions, such that G remains a simple graph throughout the update sequence. We can assume w.l.o.g. that n is greater than a sufficiently large constant, since, if it is not the case, we can simply increase it, and add some dummy vertices to G. We also assume that we are given parameters $512 \leq k \leq (\log n)^{1/49}$, $\frac{1}{k} \leq \delta < \frac{1}{400}$ and $\Delta^* \geq n^{19/k}$, such that k, Δ^* and $1/\delta$ are integers (if n was initially too small, then Δ^* may need to be increased to accommodate the increase in n).

Notice that the algorithm from Corollary 3.4 allows us to maintain the collection $\mathcal{C}$ of edge-disjoint clusters of G, the subgraphs $C' \subseteq C$ for all clusters $C \in \mathcal{C}$, and the graph H with all required properties, in the online-batch dynamic model, with batch number k and sensitivity parameter $w = n^2$. The initialization time of the algorithm is bounded by $O(n^{2+O(\delta)})$, and, for all $1 \leq i \leq k$, the time required to process the ith batch π_i of updates is bounded by $O(|\pi_i|) \cdot n^{O(\delta)}$. We denote the data structure maintained by the algorithm from Corollary 3.4 by DS.

From Lemma 2.11, we then obtain an algorithm that maintains the collection $\mathcal{C}$ of clusters, and, for every cluster $C \in \mathcal{C}$, a subgraph $C' \subseteq C$ with all required properties, whose worst-case update time is $O\left(2^{O(k)} \cdot \left(n^{O(\delta+1/k)}\right)\right) \leq O\left(n^{O(\delta)}\right)$ (since $2^{O(k)} \leq n^{O(1/k)}$ as $k \leq (\log n)^{1/49}$, and $\delta \geq 1/k$).

The only subtlety is in maintaining the graph H, so that, after each update to graph G, graph H undergoes at most $n^{O(1/k)}$ updates. The reason is that the algorithm from Lemma 2.11 can be thought of as maintaining, at all times, 3^k different instances of data structure DS, each of which is associated with a different graph H. At any given time, the algorithm indicates which of the data structures is up

to date. However, following a single update to G, the algorithm may decide to switch the designated current data structure, which may lead to a large number of changes in the graph H. In order to overcome this difficulty, we will simply include in our final sparsifier H^* the edges of all graphs H that are maintained by all currently maintained copies of data structure DS.

We now provide more details of the algorithm from Lemma 2.11, and the algorithm for maintaining the final graph H^*. It is convenient to think of the algorithm from Lemma 2.11 as consisting of $k+1$ levels, which are associated with a hierarchical partition of the time line into intervals. Let T be the time horizon of the algorithm. We first partition T into consecutive intervals of length w each, and denote the resulting collection of intervals by $\mathcal{I}_0$; we refer to these intervals as *level-0 intervals*. We can think of the level-0 algorithm as having 3^{k+1} copies of data structure DS from Corollary 3.4, that are partitioned into 3 groups $\mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3$. Each group $\mathcal{G}_r$ contains 3^k identical copies of data structure DS. For each interval $I \in \mathcal{I}_0$, every data structure DS is in one of the following three states: (i) preparation: given some initial graph G' at the beginning of interval I, the data structure needs to initialize the algorithm from Corollary 3.4 for graph G', during the first half of interval I; or (ii) active: the data structure will be used, during time interval I, by data structures from levels $1, \ldots, k$; or (iii) rollback: we can think of this part as "undoing" all computation that has been done during the preparation step. If data structure DS is active during interval I, then it is in preparation state during the preceding interval, and in the rollback state during the following interval. Moreover, all data structures in a single group $\mathcal{G}_r$ are always in the same state, and exactly one group is active during each interval $I \in \mathcal{I}_0$. Recall that the time to initialize the data structure DS is bounded by $n^{2+O(\delta)}$, and during this time the initial graph H_{DS}, that contains all edges of E^{del} and $\bigcup_{C \in \mathcal{C}} E(C')$ is constructed. We can think of the construction of the graph H_{DS} as being stretched over the course of the first half of a single level-0 interval when the data structure is in the preparation state, where we start with H_{DS} containing no edges, and then insert at most $n^{O(1/k)}$ edges into H_{DS} per time unit. Then during the rollback state, we will delete the edges from H_{DS}, at the rate of $n^{O(1/k)}$ edge deletions per time unit, until no edges remain in H_{DS}. We define the level-0 graph H^0 to contain the edges of all graphs H_{DS}, for all level-0 data structures DS, excluding the edges that were already deleted from G. Note that graph H^0 undergoes at most $n^{O(1/k)} \cdot 2^{O(k)} \leq n^{O(1/k)}$ updates per time unit.

Consider now some integer $1 \leq j \leq k$, and some level-$(j-1)$ interval I. This interval is partitioned into $\lceil n^{2/k} \rceil$ level-j intervals of identical length; we denote the length of each such interval by L_j. Moreover, there are 3^{k+1-j} identical copies of data structure DS that are active during time interval I. Each of these copies is obtained by applying the algorithm from Corollary 3.4 to some initial graph G, which then undergoes $j-1$ batches of updates. These copies are in turn partitioned into 3 groups $\mathcal{G}_1^j, \mathcal{G}_2^j, \mathcal{G}_3^j$, each of which contains 3^{k-j} identical copes of the data structure. For each level-j interval $I' \subseteq I$, each such data structure is in one of the following three states: (i) preparation: the data structure is processing the jth batch of updates, whose size is at most $4L_j \cdot n^{2/k}$, and the processing needs to complete by the middle of interval I'; (ii) active: the data structure is ready to be used by levels $j+1, \ldots k$; and (iii) rollback: we undo the changes that the data structure underwent during the preparation state. As before, if data structure DS is active during a level-j interval I, then it is in preparation state in the preceding interval, and in rollback state during the subsequent interval. Consider now any copy DS of the data structure, and suppose it is in preparation state during the level-j interval I'. Recall that the length of the interval is denoted by L_j, and the length of the update batch to the data structure is at most $4L_j \cdot n^{2/k}$. During this update sequence, the graph H_{DS} associated with the data structure undergoes at most $L_j \cdot n^{O(1/k)}$ edge insertions and deletions. We think of these updates as being spread over the course of interval I, so graph H_{DS} undergoes at most $n^{O(1/k)}$ updates per time unit. During the following rollback stage, we undo the changes that graph H_{DS} underwent during the preparation stage, with the graph undergoing at most $n^{O(1/k)}$ updates per time unit. We now define the level-j graph H^j to be the union of all graphs H_{DS} for all data structures

$\mathsf{DS} \in \mathcal{G}_1^j \cup \mathcal{D}_2^j \cup \mathcal{D}_3^j$, excluding edges that have been deleted from G. As before, graph H^j undergoes at most $n^{O(1/k)} \cdot 2^{O(k)} \leq n^{O(1/k)}$ updates per time unit.

We let $j^* \leq k-1$ be smallest integer, such that level-j^* time intervals have length at most $n^{2/k}$ each. In this case, we think of level-(j^*+1) time intervals as consisting of a single time slot each. For each level-(j^*+1) time interval $I = \{\tau\}$, there is at least one data structure DS, that is up to date with respect to the graph G at time τ. This is the data structure that the algorithm from Lemma 2.11 considers as current. The corresponding graph H_{DS} must then be contained in the graph $H^* = H^0 \cup \cdots \cup H^{j^*+1}$ that our algorithm maintains. From our discussion, graph H^* undergoes at most $n^{O(1/k)}$ edge insertions or deletions per time unit. Since the algorithm from Corollary 3.4 ensures that each graph H maintained by a copy of the data structure contains at most $n^{1+O(1/k)} \cdot \Delta^*$ edges, and since, at all times, our graph H^* is contained in the union of at most $2^{O(k)}$ such graphs, we get that $|E(H^*)| \leq 2^{O(k)} \cdot n^{1+O(1/k)} \cdot \Delta^* \leq n^{1+O(1/k)} \cdot \Delta^*$ holds. This completes the proof of Theorem 1.3.

4 First Tool: a Nice Router and its Pruning

In this section we provide an explicit construction of a router graph, and provide an algorithm for pruning this graph. We show that the graph remains a router even after pruning. As mentioned in the introduction, our construction is very similar to that of [HLS24] (see Section 5 in the full version). Our pruning algorithm is more involved, and ensures somewhat better routing parameters, in terms of the lengths of the routing paths, in the resulting pruned graph.

4.1 Construction of a Nice Router Graph

We assume that we are given three integral parameters: $k \geq 256$, $N \geq k^{3k}$, and $\Delta \geq 4k^2$. We sometimes refer to Δ the *target degree*. We now describe a construction of a corresponding router, that we denote by $W_k^{N,\Delta}$. To avoid clutter, in the remainder of this section, we assume that N and Δ are fixed, and we do not include them in superscripts, so in this section we denote graph $W_k^{N,\Delta}$ by W_k. The construction of graph W_k is recursive. We start by constructing a graph W_1, and then, for $1 < i \leq k$, we construct graph W_i by combining N copies of graph W_{i-1}. Parameters N and Δ remain unchanged for these graphs.

Graph W_i – a High Level View. Consider an integer $1 \leq i \leq k$. We now provide a high-level description of graph $W_i = (V_i, E_i)$. The set V_i contains exactly N^i vertices, and it is partitioned into two subsets: set V_i^c of N^{i-1} vertices called *center* vertices, and set $V_i^\ell = V_i \setminus V_i^c$ of remaining vertices, called *leaf* vertices. The set $E(W_i)$ of edges of the graph is partitioned into i subsets $E_1^i, \ldots, E_i^i$, where for $1 \leq z \leq i$, we refer to the edges of E_z^i as *level-z edges*. If we let $H_z = G_i[E_z^i]$ be the subgraph of G_i induced by the set E_z^i of level-z edges, and denote by $\mathcal{R}_z$ the collection of the connected components of H_z, then each such connected component $C \in \mathcal{R}_z$ contains exactly N vertices, out of which exactly one is a center vertex. Additionally, if we let S be a star defined over the vertices of C, whose center is the unique vertex of $V(C) \cap V_i^c$, then C can be obtained from S by replacing every edge of S with Δ parallel copies (we refer to these copies as a *bundle*; the edges of S are referred to as *superedges*). We denote by $\mathcal{S}_z$ a collection of star graphs that contains, for every connected component $C \in \mathcal{R}_z$, the corresponding star graph S on $V(C)$ (without the parallel edges). Note that every edge in W_i connects a leaf vertex to a center vertex. We now describe the construction of the graph W_i in more detail. The construction is recursive, from smaller to larger values of i.

Graph W_1. The set of vertices of the graph W_1 is $V_1 = \{v_1, \ldots, v_N\}$, where vertex v_1 is the *center* vertex, and vertices $v_2, \ldots, v_N$ are *leaf vertices*. In other words, $V_1^c = \{v_1\}$, and $V_1^\ell = \{v_2, \ldots, v_N\}$. Let S be a star defined on the vertices of V_1, whose center is vertex v_1, and leaves are $v_2, \ldots, v_N$. Graph W_1 is obtained from S by replacing every edge of S with a collection of Δ parallel edges. All edges of W_1 belong to level 1, so $E(W_1) = E_1^1$. The collection $\mathcal{S}_1$ of level-1 stars contains a single star S.

Graph W_i for $i > 1$. We assume that we are given an integer $i > 1$, and that we have already defined the graph W_{i-1}. We now describe the construction of graph W_i.

We start by creating N copies of graph W_{i-1}, that we denote by $C_1, \ldots, C_N$. We refer to $C_1, \ldots, C_N$ as *level-$(i-1)$ clusters*. We let the set V_i of vertices of W_i be $\bigcup_{j=1}^{N} V(C_j)$. For all $1 \leq j \leq N$, if $v \in V(C_j)$ is a center vertex for C_j, then it becomes a center vertex for W_i, and otherwise it becomes a leaf vertex for W_i. Recall that, for all $1 \leq j \leq N$, $|V(C_j)| = N^{i-1}$, and the number of center vertices in C_j is N^{i-2}. Therefore, we get that $|V_i| = N^i$ and the number of center vertices in W_i is N^{i-1}, as required.

For all $1 \leq i' < i$, the set $E_{i'}^i$ of level-i' edges of W_i is the union of the sets of level-i' edges of $C_1, \ldots, C_N$. It now remains to define the set E_i^i of level-i edges for graph W_i. In order to do so, we will construct a collection $\mathcal{S}_i$ of N^{i-1} disjoint stars, where each star $S \in \mathcal{S}_i$ contains exactly one vertex from each of the clusters $C_1, \ldots, C_N$, and the center of the star lies in V_i^c. The set E_i^i of level-i edges is then obtained by replacing each edge $e \in \bigcup_{S \in \mathcal{S}_i} E(S)$ with Δ parallel edges.

In order to construct the collection $\mathcal{S}_i$ of stars, we first define, for each $1 \leq j \leq N$, an ordering $v_1^j, \ldots, v_{N^{i-1}}^j$ of the vertices of C_j, such that, for all $1 \leq z \leq N^{i-1}$, if we consider the vertices $v_z^1, \ldots, v_z^N$ (that is, the vertices that appear in the zth position in the orderings of $V(C_1), \ldots, V(C_N)$), then exactly one of these vertices is a center vertex in V_i^c.

Such orderings are easy to define: consider the collection $I = \left\{1, \ldots, N^{i-1}\right\}$ of indices, and partition it into N disjoint subsets $I_1, \ldots, I_N$, each of which contains exactly N^{i-2} contiguous indices. For each $1 \leq j \leq N$, we order the vertices of C_j arbitrarily, but we ensure that the center vertices of C_j lie in the positions whose indices belong to I_j in this ordering. This ensures that, for all $1 \leq z \leq N^{i-1}$, exactly one vertex that lies in the zth position in the orderings of $C_1, \ldots, C_N$ is a center vertex.

For all $1 \leq z \leq N^{i-1}$, we let S_z be the star that consists of the vertices $\left\{v_z^1, \ldots, v_z^N\right\}$, where the center of the star is the unique vertex v_z^j that lies in V_i^c. Then for every edge (x, y) of the star S_z, we create Δ parallel edges (x, y), which are added to set E_i^i. These parallel edges are called a *bundle*, and the edge of the star corresponding to them is called a *superedge*. We denote by $\mathcal{S}_i = \{S_1, \ldots, S_{N^{i-1}}\}$ the resulting collection of level-i stars. This completes the construction of the set E_i^i of level-i edges of graph W_i. Lastly, we let $E(W_i) = \bigcup_{i'=1}^{N} E_{i'}^i$, completing the construction of the graph W_i. For all $1 \leq i' < i$, the collection $\mathcal{S}_{i'}$ of level-i' stars is the union of the collections of level-i' stars of the graphs $C_1, \ldots, C_N$.

Consider now the level-k graph W_k. For all $1 \leq i < k$, graph W_k contains N^{k-i} copies of the level-i graph W_i. We let $\mathcal{C}_i$ denote the partition of the vertices of W_k defined by these copies of W_i. We refer to $\mathcal{C}_i$ as *level-i clustering*, and we refer to each subgraph $C \in \mathcal{C}_i$ as a *level-i cluster*. Notice that for all $1 \leq i < k$, each level-i cluster is contained in a single level-$(i+1)$ cluster, and moreover, each level-$(i+1)$ cluster contains the union of exactly N level-i clusters. There is a single level-k cluster, containing all vertices of W_k. From our construction, it is easy to verify that, for all $1 \leq i \leq k$, if we denote by $H_i = W_k[E_i^k]$ the subgraph of W_k induced by the level-i edges, then there is a collection $\mathcal{S}_i$ of N^{k-1} disjoint star graphs, where each star $S \in \mathcal{S}_i$ has N vertices, and the center of S lies in V_k^c, such that the set E_i^k of edges can be obtained from $E_i' = \bigcup_{S \in \mathcal{S}_i} E(S)$, by replacing each edge $e \in E_i'$

with Δ parallel edges. We refer to the set $\mathcal{S}_i$ of stars as *level-i stars*. Lastly, note that for all $1 \leq i \leq k$, the number of level-i superedges in W_k is at most N^k, and $|E_i^k| \leq \Delta \cdot N^k$. Overall, W_k contains at most $k \cdot N^k$ superedges, and $|E(W_k)| \leq k \cdot \Delta \cdot N^k$. Moreover, the degree of every center vertex in W_k is exactly $(N-1) \cdot \Delta \cdot k$, and the degree of every leaf vertex is exactly $\Delta \cdot k$.

As mentioned already, the construction above is almost identical to that of [HLS24]. The main difference is that they used cliques instead of stars, and their construction used the parameter $\Delta = 1$. We do not directly prove that the graph W_k is a router. Instead, we first define *properly pruned* subgraphs of W_k, and we prove that each such graph is a router. This definition is somewhat more involved than that in [HLS24], but it achieves better routing parameters in the resulting graph.

4.2 Properly Pruned Subgraphs of W_k

For all $1 \leq i \leq k$, we now define the notion of a *properly pruned subgraph* of W_i. We then show that any Δ-restricted demand can be routed with low congestion on paths of length $O(k^2)$ in a properly pruned subgraph of W_k.

Definition 4.1 *For $1 \leq i \leq k$, a subgraph $W \subseteq W_i$ is a* properly pruned *subgraph of W_i with respect to sets $U_1, U_2, \ldots, U_i \subseteq V(W_i)$ of its vertices, where $U_i \subseteq U_{i-1} \subseteq \cdots \subseteq U_1$, if the following hold:*

P1. for every level $1 \leq i' \leq i$, for every leaf vertex $v \in V_i^{\ell}$:

- *if $v \in U_{i'}$, then at least $\lceil \Delta/2 \rceil$ level-i' edges that are incident to v in W_i lie in W (recall that all these edges must be parallel edges that correspond to the unique level-i' superedge that is incident to v);*
- *otherwise, W contains no level-i' edges that are incident to v;*

P2. for each level $1 \leq i' \leq i$, for every center vertex $v \in V_i^c$, if we denote by $S \in \mathcal{S}_{i'}$ the unique level-i' star for which v serves as a center, then:

- *if $v \in U_{i'}$, then at least $N \cdot \left(1 - \frac{1}{k}\right)$ leaf vertices of S lie in $U_{i'}$ (and we say that star S* survives *in this case);*
- *otherwise, no leaf vertices of S lie in $U_{i'}$ (and we say that star S is* destroyed*);*

P3. $V(W) = U_1$; and

P4. for every level $1 \leq i' < i$, for every level-i' cluster $C \in \mathcal{C}_{i'}$ of W_i, either $V(C) \cap U_1 = \emptyset$ (in which case we say that cluster C is destroyed*), or at least $\frac{|V(C)|}{k^{3k}}$ vertices of C lie in $U_{i'+1}$ (and we say that cluster C* survives*).*

The following observation follows immediately from the definition of a properly pruned subgraph.

Observation 4.2 *Let W be a subgraph of W_i, for some $1 \leq i \leq k$, and let $U_1, \ldots, U_i$ be subsets of vertices of W with $U_i \subseteq U_{i-1} \subseteq \cdots \subseteq U_1$. Assume that W is a properly pruned subgraph of W_i with respect to $U_1, \ldots, U_i$. Consider a level $1 \leq i' < i$, a level-i' cluster C of W_i (which is a copy of $W_{i'}$), and let W^C be the subgraph of C induced by the set $V(W) \cap V(C)$ of vertices. For $1 \leq i'' \leq i'$, let $U_{i''}^C = U_{i''} \cap V(C)$. Then W^C is a properly pruned subgraph of $W_{i'}$, with respect to the sets $U_1^C, \ldots, U_{i'}^C$ of its vertices.*

In the following subsection, we prove that a properly pruned subgraph of W_k is a good router. After that we show an algorithm that maintains a properly pruned subgraph of W_k, as it undergoes adversarial edge deletions.

4.3 Routing in Properly Pruned Subgraphs of W_k

The main goal of this subsection is to prove the following theorem, that shows that a properly pruned subgraph of W_k is a good router.

Theorem 4.3 *Let W be a properly pruned subgraph of W_k, with respect to sets $U_1, \ldots, U_k$ of its vertices, and let $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ be a $\frac{\Delta}{k^{4k}}$-restricted demand for W. Then there is a fractional routing of $\mathcal{D}$ in W with no edge-congestion, such that every flow-path P with $f(P) > 0$ has length at most $20k^2$. Moreover, there is an algorithm, that, given the graph W, the sets $U_1, \ldots, U_k$ of its vertices, and the demand $\mathcal{D}$, computes such a routing in time $O\left(k^2 \cdot \Delta \cdot N \cdot (N^k + |\Pi|)\right)$.*

The remainder of this subsection is dedicated to the proof of Theorem 4.3. For all $1 \leq i \leq k$, let $r_i = \frac{\Delta}{32^i}$. We start by proving a slightly weaker result: namely, that for all $1 \leq i \leq k$, if W is a properly pruned subgraph of W_i with respect to sets $U_1, \ldots, U_i$ of its vertices, and $\mathcal{D}$ is any r_i-restricted demand that is defined over the vertices of U_i, then $\mathcal{D}$ can be fractionally routed in W_i over short paths with no edge-congestion.

Lemma 4.4 *Fix an integer $1 \leq i \leq k$, let W be a subgraph of W_i, and let $U_1, \ldots, U_i$ be subsets of vertices of W with $U_i \subseteq U_{i-1} \subseteq \cdots \subseteq U_1$. Assume further that W is a properly pruned subgraph of W_i with respect to $U_1, \ldots, U_i$. Then for any r_i-restricted demand $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ with $\Pi \subseteq U_i \times U_i$, there is a routing of $\mathcal{D}$ in W with no edge-congestion over flow-paths of length at most $4i$. If the demand $\mathcal{D}$ is integral, then so is the routing. Moreover, there is an algorithm, that, given the graph W and the sets $U_1, \ldots, U_i$ of its vertices, computes such a routing in time $O\left(i \cdot \Delta \cdot N \cdot (N^i + |\Pi|)\right)$.*

Notice that the main difference between Lemma 4.4 and Theorem 4.3 is that in Lemma 4.4 all demands must be between the vertices of U_i, while in Theorem 4.3 the demands are between the vertices of $V(W) = U_1$.

Proof: The proof is by induction on i. For the induction base, we consider a subgraph W of W_1. Recall that W_1 is a graph on N vertices, that can be defined as follows: let S be a star graph on N vertices. Then W_1 is obtained from S by replacing every edge of S by a set of Δ parallel edges. From Properties P1–P3, $U_1 = V(W)$ must hold. Additionally, if $W \neq \emptyset$, then there is a star graph $S' \subseteq S$, that contains the center vertex of S, such that W can be obtained from S' by replacing every edge of S' with parallel edges, whose number is between $\lceil \Delta/2 \rceil$ and Δ. Consider now any r_1-restricted demand $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$, with $\Pi \subseteq U_1 \times U_1$, and recall that $r_1 = \Delta/32$. It is immediately to see that demand $\mathcal{D}$ can be routed in W, via paths of length at most 2. Each such path simply connects a pair of leaves in the star, or it connects the center of the star to one of its leaves. It is also immediate to design an algorithm that computes such a routing in time $O(N^2 \cdot \Delta)$. If the demand is integral, then so is the routing.

Consider now an integer $i > 1$, and assume that the claim holds for $i-1$. Let W be a subgraph of W_i, and let $U_1, \ldots, U_i$ be subsets of its vertices with $U_i \subseteq U_{i-1} \subseteq \cdots \subseteq U_1$, such that W is a properly pruned subgraph of W_i with respect to $U_1, \ldots, U_i$. Recall that W_i contains a disjoint union of N copies of W_{i-1}, that we denote by $\mathcal{C} = \{C_1, \ldots, C_N\}$, and refer to as *clusters*. For each such cluster $C \in \mathcal{C}$, let $W^C = W \cap C$, and, for all $1 \leq i' \leq i$, let $U^C_{i'} = U_{i'} \cap V(C)$.

Let $\mathcal{C}' \subseteq \mathcal{C}$ be the set of clusters $C \in \mathcal{C}$, whose corresponding graph W^C is non-empty. For every cluster $C \in \mathcal{C}'$, we denote the sets of its leaf and center vertices by $V^\ell(C)$ and $V^c(C)$, respectively. From Observation 4.2, graph W^C is a properly pruned subgraph of W_{i-1}, with respect to the sets $U^C_1, \ldots, U^C_{i-1}$ of its vertices.

Consider now an r_i-restricted demand $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ for W, where $\Pi \subseteq U_i \times U_i$. For every pair $(a,b) \in \Pi$, we will define a *proxy pair* (a',b') of vertices, and we will set a *proxy demand* $D'(a',b') = D(a,b)$. We will ensure that a and a' lie in the same level-i star, and that the same is true for b and b'. We will also ensure there is some cluster $C \in \mathcal{C}'$ with $a', b' \in U_{i-1}^C$. We say that a' is a *proxy vertex for a with respect to pair* (a,b), and we similarly say that b' is a *proxy vertex for b with respect to pair* (a,b). Additionally, we will define a flow $F_{(a,b)}$ in W, in which a sends $D(a,b)$ flow units to a', and b sends $D(a,b)$ flow units to b'. The flow $F_{(a,b)}$ will only use level-i edges.

For every cluster $C \in \mathcal{C}'$, we will therefore obtain a new demand $\mathcal{D}_C = \left\{\Pi_C, \{D'(a',b')\}_{(a',b')\in\Pi_C}\right\}$, where $\Pi_C \subseteq U_{i-1}^C \times U_{i-1}^C$. The demand $\mathcal{D}_C$ is obtained by combining all proxy demands $D'(a',b')$ where $a', b' \in V(C)$. We will ensure that this demand is r_{i-1}-restricted. We can then use the induction hypothesis to route, for each cluster $C \in \mathcal{C}'$, the corresponding demand $\mathcal{D}_C$ in W^C, obtaining a flow f_C. The final routing of $\mathcal{D}$ is obtained by combining the flows f_C for all clusters $C \in \mathcal{C}'$ with the partial routings $F_{(a,b)}$ for $(a,b) \in \Pi$ that we have computed.

We now describe the algorithm more formally. Throughout the algorithm, for every vertex $v \in V(W)$, the *load* on vertex v, denoted by $\lambda(v)$, is the total amount of the proxy demand $\sum_u D'(u,v)$ in which v currently participates. Initially, the proxy demands are undefined, and so the load on every vertex is 0.

We process the pairs $(a,b) \in \Pi$ one by one. Consider an iteration when the pair (a,b) is processed. Let $S, S' \in \mathcal{S}_i$ be the level-i stars containing a and b, respectively. Assume first that $S = S'$. In this case, we let $F_{(a,b)}$ be the flow that sends $D(a,b)$ flow units from a to b by employing the edges of $E_i \cap E(W)$ (the level-i edges of W_i that lie in W), that correspond to the edges of S that lie on the unique a-b path in S, completing the routing of $D(a,b)$ flow units from a to b, via paths of length at most 2.

Assume now that $S \neq S'$. Recall that every star in $\mathcal{S}_i$ contains exactly one vertex from each of the graphs $C_1, \ldots, C_N \in \mathcal{C}$. Moreover, from Property P2, at least $N \cdot \left(1 - \frac{1}{k}\right)$ leaves of S, and at least $N \cdot \left(1 - \frac{1}{k}\right)$ leaves of S' lie in U_i. Therefore, there is a collection $\mathcal{C}_{(a,b)} \subseteq \mathcal{C}'$ of clusters, with $|\mathcal{C}_{(a,b)}| \geq N \cdot \left(1 - \frac{2}{k}\right)$, so that, for every cluster $C \in \mathcal{C}_{(a,b)}$, vertex set $V(S) \cap U_i$ contains a leaf vertex of C, and so does $V(S') \cap U_i$. For a cluster $C \in \mathcal{C}_{(a,b)}$, let a^C be the unique vertex of C that lies in $V(S) \cap U_i$, and let b^C be the unique vertex of C that lies in $V(S') \cap U_i$. We say that cluster $C \in \mathcal{C}_{(a,b)}$ is *admissible* for pair (a,b) if $\lambda(a^C), \lambda(b^C) \leq r_{i-1} - D(a,b)$. We will later show that at least one cluster $C \in \mathcal{C}_{(a,b)}$ must be admissible for (a,b). Let C be any such cluster. Then we define the proxy pair for (a,b) as $(a',b') = (a^C, b^C)$, and we set $D'(a',b') = D(a,b)$. We also increase $\lambda(a'), \lambda(b')$ by $D(a,b)$. Finally, we construct the flow $F_{(a,b)}$, in which a routes $D(a,b)$ flow units to a' and b routes $D(a,b)$ flow units to b' in a straightforward manner: the flow from a to a' simply follows the unique a-a' path in star S, and the flow from b to b' follows the unique b-b' path in star S'. Therefore, all flow-paths in $F_{(a,b)}$ have length at most 2. This completes the desription of the algorithm for constructing proxy demands. Let F be the flow obtained by taking the union of all flows $F_{a,b}$ for all $(a,b) \in \Pi$. Then all flow-paths in F have length at most 2, and they only use level-i edges. Since the demand $\mathcal{D}$ is r_i-restricted, and $r_i \leq \Delta/4$, it is immediate to verify that flow F causes no edge-congestion. It is also immediate to verify that every pair $(a,b) \in \Pi$ can be processed in time $O(N + \Delta)$. Next, we show that, when a pair $(a,b) \in \Pi$ is processed, an admissible cluster $C \in \mathcal{C}_{(a,b)}$ must exist.

Observation 4.5 *For every pair $(a,b) \in \Pi$, when the pair is processed, an admissible cluster $C \in \mathcal{C}_{(a,b)}$ must exist.*

Proof: Assume otherwise. Let A be the set of all leaf vertices $v \in V(S) \cap U_i$, such that v lies in some cluster of $\mathcal{C}_{(a,b)}$, and let B be the set of all leaf vertices $u \in V(S') \cap U_{i'}$, such that u lies in some cluster

of $\mathcal{C}_{(a,b)}$. Recall that $|A| = |B| = |\mathcal{C}_{(a,b)}| \geq N \cdot \left(1 - \frac{2}{k}\right) \geq \frac{N}{2}$. Let $A' \subseteq A$ be the set of vertices v whose current load $\lambda(v) > r_{i-1} - D(a,b)$, and let $B' \subseteq B$ be defined similarly. Since no cluster of $\mathcal{C}_{(a,b)}$ is admissible, it must be the case that either $|A'| \geq \frac{|A|}{2} - 1 > \frac{|A|}{4}$, or $|B'| > \frac{|B|}{4}$. Assume w.l.o.g. that it is the former. Then the total load on the vertices of A' is:

$$\sum_{v \in A'} \lambda(v) > \frac{|A|}{4} \cdot (r_{i-1} - D(a,b)) \geq \frac{|\mathcal{C}_{(a,b)}|}{4} \cdot (r_{i-1} - r_i) \geq \frac{N \cdot r_{i-1}}{16},$$

since $D(a,b) \leq r_i$ and $r_{i-1} > 2r_i$. However, the total demand of the vertices of S in $\mathcal{D}$ is bounded by $N \cdot r_i$. The vertices of A' may only serve as proxy vertices of the vertices of S, and so the total load on the vertices of A' is bounded by $Nr_i < \frac{N \cdot r_{i-1}}{16}$, since $r_i = \frac{\Delta}{32^i}$, a contradiction. □

For each cluster $C \in \mathcal{C}'$, we now obtain an r_{i-1}-restricted demand $\mathcal{D}_C = \left\{\Pi_C, \{D'(a',b')\}_{(a',b') \in \Pi_C}\right\}$, that includes all proxy pairs (a',b') with $a', b' \in V(C) \cap U_i$. Clearly, for each such proxy pair (a',b'), $a', b' \in U_{i-1}^C$. Using the induction hypothesis, we compute, for each such cluster $C \in \mathcal{C}'$, a routing F^C of the demand $\mathcal{D}_C$ in graph W^C via paths of length at most $4(i-1)$ with no congestion. The time required to compute the flow F^C is bounded by $O\left((i-1) \cdot \Delta \cdot N \cdot (N^{i-1} + |\Pi_C|)\right)$. Since $\sum_{C \in \mathcal{C}} |\Pi_C| \leq |\Pi|$, we get the total time requried to compute all such flows F^C is bounded by $O\left((i-1) \cdot \Delta \cdot N \cdot (N^i + |\Pi|)\right)$. Lastly, by combining the partial flow F that we have computed with the flows F^C for clusers $C \in \mathcal{C}'$, we obtain a routing of the demand $\mathcal{D}$ over flow-paths of length at most $4i$. The routing causes no congestion, since the flow F only uses level-i edges, while the flows F_C for $C \in \mathcal{C}'$ cannot use any level-i edges. The total running time of the algorithm is bounded by:

$$O\left((i-1) \cdot \Delta \cdot N \cdot (N^i + |\Pi|)\right) + O(|\Pi| \cdot (N + \Delta)) \leq O\left(i \cdot \Delta \cdot N \cdot (N^i + |\Pi|)\right).$$

If the demand $\mathcal{D}$ is integral, then it is easy to verify that the resulting routing is integral as well. □

The following lemma summarizes our second step in the proof of Theorem 4.3. The lemma provides an algorithm for routing vertices of U_1 to vertices of U_i in a properly pruned subgraph W of W_i.

Lemma 4.6 *Fix an index $1 \leq i \leq k$, and let W be a properly pruned subgraph of W_i with respect to sets $U_1, \ldots, U_i$ of its vertices. Then there is an integral flow f in W, in which every flow-path P with $f(P) > 0$ connects a vertex of U_1 to a vertex of U_i; every vertex of U_1 sends Δ flow units; every vertex of U_i receives at most $\Delta \cdot k^{3k}$ flow units; the congestion of the flow is at most $32^i \cdot k^{3k}$; and the length of every flow-path is at most $(4i)^2$. Additionally, for every vertex $v \in U_1$, all flow originating from v terminates at a single vertex of U_i. Moreover, there is an algorithm, that, given W and the sets $U_1, \ldots, U_i$ of its vertices, computes such a flow in time $O(i^2 \cdot \Delta \cdot N^{i+1})$.*

Proof: The proof is by induction on i. The base case is when $i = 1$. Let W be a properly pruned subgraph of W_1 with respect to U_1. We can then let f be a trivial flow, in which every vertex $v \in U_1$ sends Δ flow units to itself, via a flow-path $P_v = (v)$.

Consider now an integer $i > 1$, and assume that the claim holds for $i-1$. Let W be any properly pruned subgraph of W_i with respect to sets $U_1, \ldots, U_i$ of its vertices. Recall that W_i contains a disjoint union of N copies of W_{i-1}, that we denote by $\{C_1, \ldots, C_N\}$, and refer to as *clusters*. Let $\mathcal{C} \subseteq \{C_1, \ldots, C_N\}$ be the set of clusters that survived, and consider any such cluster $C \in \mathcal{C}$.

Denote $W^C = W \cap C$, and, for all $1 \leq i' \leq i$, we denote by $U_{i'}^C = U_{i'} \cap V(C)$. From Observation 4.2, graph W^C is a properly pruned subgraph of W_{i-1} with respect to vertex sets $U_1^C, \ldots, U_{i-1}^C$, and, from Property P4 of properly pruned subgraphs, $|U_i^C| \geq \frac{V(C)}{k^{3k}}$. By using the induction hypothesis, we

compute, in time $O((i-1)^2 \cdot \Delta \cdot N^i)$, a flow f_C in graph W^C, where every vertex of U_1^C sends Δ flow units, and every vertex of U_{i-1}^C receives at most $\Delta \cdot k^{3k}$ flow units, so that the congestion of the flow is at most $32^{i-1} \cdot k^{3k}$, and the length of every flow-path is at most $(4(i-1))^2$. Additionally, for every vertex $v \in U_1^C$, all flow originating at v terminates at a unique vertex of U_{i-1}^C.

In the remainder of the proof, we will compute another flow $\hat{f}_C$, whose purpose is to route the flow that the vertices in U_{i-1}^C receive via f_C to the vertices of U_i^C, inside the graph W^C. In order to do so, we will first define an r_{i-1}-restricted demand $\mathcal{D}_C$ over the vertices of U_{i-1}^C, and then use Lemma 4.4 in order to route it in W^C. After that, by properly scaling the resulting flow, we will obtain the desired flow $\hat{f}_C$. Lastly, by composing the flows f_C and $\hat{f}_C$, we obtain a flow f_C^* from vertices of U_1^C to vertices of U_i^C, where every vertex in U_1^C sends Δ flow units and every vertex in U_i^C receives at most $\Delta \cdot k^{3k}$ flow units. The final flow is obtained by taking the union of flows f_C^* for all $C \in \mathcal{C}$.

We now turn to define the demand $\mathcal{D}_C = \left(\Pi^C, \{D'(a,b)\}_{(a,b)\in\Pi^C}\right)$. Recall that, from Property P4 of properly pruned subgraphs, $|U_i^C| \geq \frac{|V(C)|}{k^{3k}} \geq \frac{|U_1^C|}{k^{3k}}$. Using a simple greedy algorithm we can then compute a mapping σ, that maps every vertex $v \in U_1^C$, to some vertex $\sigma(v) \in U_i^C$, such that, for every vertex $u \in U_i^C$, at most k^{3k} vertices of U_1^C are mapped to u. Recall that every vertex $v \in U_1^C$ sends Δ flow units to some vertex $u' \in U_{i-1}^C$ via the flow f_C. We denote $\sigma'(v) = u'$. We are now ready to define the demand $\mathcal{D}_C$. For every vertex $v \in U_1^C$, we add the pair $(\sigma'(v), \sigma(v))$ to Π^C, with demand $D'(\sigma'(v), \sigma(v)) = \Delta$. It is easy to verify that $\Pi^C \subseteq U_{i-1}^C \times U_{i-1}^C$, and that the resulting demand $\mathcal{D}_C$ is $(\Delta \cdot k^{3k})$-restricted.

Let $\mathcal{D}'_C$ be the demand that is identical to $\mathcal{D}_C$, except that we scale all demands down by factor $32^{i-1} \cdot k^{3k}$. Then $\mathcal{D}'_C$ is an r_{i-1}-restricted demand, defined over the set U_{i-1}^C of vertices. We can now apply the algorithm from Lemma 4.4 to graph W^C and demand $\mathcal{D}'_C$, to compute a routing f'_C of the demand $\mathcal{D}'_C$ in W^C with no edge-congestion, with flow-paths of length at most $4(i-1)$. The time required to compute this flow is bounded by:

$$O\left((i-1) \cdot \Delta \cdot N \cdot (N^{i-1} + |\Pi^C|)\right) \leq O\left((i-1) \cdot \Delta \cdot N^i\right),$$

since $|\Pi^C| \leq |U_1^C| \leq N^{i-1}$. By scaling the flow f'_C up by factor $32^{i-1} \cdot k^{3k}$, we obtain a flow $\hat{f}^C$, that routes the demand $\mathcal{D}_C$ via paths of length at most $4(i-1)$, with congestion at most $32^{i-1} \cdot k^{3k}$.

Next, we combine flow f_C and $\hat{f}_C$, obtaining a flow f_C^*, in which every vertex $v \in U_1^C$ sends Δ flow units to vertex $\sigma(v) \in U_i^C$; the length of each flow-path is at most $(4(i-1))^2 + 4(i-1) \leq (4i)^2$, and the flow causes congestion at most $2 \cdot 32^{i-1} \cdot k^{3k} \leq 32^i \cdot k^{3k}$.

The final routing is obtained by taking the union of the flows f_C^* for all clusters $C \in \mathcal{C}$.

The time required to compute the flows $\hat{f}_C$ for all $C \in \mathcal{C}$ is bounded by $O\left((i-1) \cdot \Delta \cdot N^{i+1}\right)$. The time required to compute the flows f_C for all $C \in \mathcal{C}$ by the induction hypothesis is bounded by $O((i-1)^2 \cdot \Delta \cdot N^{i+1})$. Therefore, the total running time of the algorithm is bounded by $O(i^2 \cdot \Delta \cdot N^{i+1})$.
□

We are now ready to complete the proof of Theorem 4.3. We assume that we are given a properly pruned subgraph W of W_k, with respect to sets $U_1, \ldots, U_k$ of its vertices, together with a $\frac{\Delta}{k^{4k}}$-restricted demand demand $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ for W.

We use the algorithm from Lemma 4.6 to compute an integral flow f, in which every flow-path P with $f(P) > 0$ connects a vertex of U_1 to a vertex of U_k; every vertex of U_1 sends Δ flow units; every vertex of U_k receives at most $\Delta \cdot k^{3k}$ flow units; the congestion of the flow is at most $32^k \cdot k^{3k} \leq \frac{k^{4k}}{4}$ (since $k \geq 256$); and the length of every flow-path is at most $(4k)^2$. Recall that we are guaranteed that, for

every vertex $v \in U_1$, all flow originating from v terminates at a single vertex of U_k, that we denote by $\sigma(v)$. The running time of the algorithm so far is $O(k^2 \cdot \Delta \cdot N^{k+1})$. We slightly modify the flow f by scaling the flow on every flow-path down by factor k^{4k}, obtaining a flow in which every edge carries at most $\frac{1}{4}$ flow units. Notice that now every vertex $u \in U_k$ receives at most $\frac{\Delta}{k^k} \leq \frac{\Delta}{2 \cdot 32^k}$ flow units, and every vertex $v \in U_1$ sends $\frac{\Delta}{k^{4k}}$ flow units.

Next, we define a new demand $\mathcal{D}' = \left\{\Pi', \{D'(a,b)\}_{(a,b)\in\Pi'}\right\}$, as follows. For every pair $(a,b) \in \Pi$, we add the pair $(\sigma(a), \sigma(b))$ to Π', with demand $D'(\sigma(a), \sigma(b)) = D(a,b)$. Clearly, $\Pi' \subseteq U_k \times U_k$. We claim that the resulting demand $\mathcal{D}'$ is $\frac{r_k}{2}$-restricted, where $r_k = \frac{\Delta}{32^k}$. Indeed, consider any vertex $u \in U_k$, and let $A \subseteq U_1$ be the collection of all vertices v with $\sigma(v) = u$. Then $|A| \leq k^{3k}$ must hold, and, since the original demand $\mathcal{D}$ was $\frac{\Delta}{k^{4k}}$-restricted, we get that $\mathcal{D}'$ must be $\frac{\Delta}{k^k} \leq \frac{\Delta}{2 \cdot 32^k} = \frac{r_k}{2}$-restricted. We can now use the algorithm from Lemma 4.4 in order to compute a routing f' of $\mathcal{D}'$ in W via flow-paths of length at most $4k$, such that every edge of W carries at most $1/2$ flow units (in order to do this, we first scale all demands in $\mathcal{D}'$ up by factor 2 to obtain an r_k-restricted demand; compute its routing with no congestion via Lemma 4.4, and then scale the flow down by factor 2). The running time required to compute flow f' is bounded by $O\left(k \cdot \Delta \cdot N \cdot (N^k + |\Pi|)\right)$. Lastly, we combine the flows f and f', possibly reducing the flow f on some of its flow-paths as needed, to obtain the final routing f^* of the demand $\mathcal{D}$ in W via flow-paths of length at most $(4k)^2 + 4k \leq 20k^2$, with no edge congestion. The total running time of the algorithm is bounded by $O\left(k^2 \cdot \Delta \cdot N \cdot (N^k + |\Pi|)\right)$.

4.4 Efficient Pruning of Graph W_k

In this subsection we provide an algorithm to efficiently prune the graph $W_k^{N,\Delta}$, as it undergoes an online sequence of edge deletions. In order to motivate the model that we use in this section, we note that our approach for proving Theorem 3.2 uses the online-batch dynamic update model (see Section 2.7). Intuitively, during initialization, we will construct an initial clustering $\mathcal{C}$ of the input graph G, and, for every cluster $C \in \mathcal{C}$, we will compute an embedding of a graph $W_k^{N,\Delta}$, for suitably chosen parameters N and Δ, into C. In order to ensure that C has strong routing properties, we need to ensure that every vertex of C participates in the embeddings of many edges of $W_k^{N,\Delta}$. Recall that in the online-batch dynamic update model, the algorithm receives updates in σ batches $\pi_1, \ldots, \pi_\sigma$. In our case, we will use $\sigma = k$, and the only type of updates we will need to process is the deletion of edges from the input graph. Consider now some cluster C, and let E' be the set of edges deleted from C in a single batch π_i. Then every edge $e \in E(W_k^{N,\Delta})$ whose embedding path contains an edge from E' needs to be deleted from $W_k^{N,\Delta}$. In order to ensure that we obtain a properly pruned subgraph $W \subseteq W_k^{N,\Delta}$, we may delete some additional edges from W. As the result of these deletions, it is possible that, for some vertices $v \in V(C)$, the number of edges $e \in E(W)$ whose embedding path contains v becomes too small. When this happens, we need to delete v from C, and we need to further delete all edges $e \in E(W)$, whose embedding path uses v, from W. This, in turn, may trigger another round of pruning, and so on. Therefore, even though our algorithm for maintaining a router decomposition works in the online-batch dynamic model, the input to the pruning algorithm does not arrive in batches. Instead, we partition the timeline into phases $\Phi_0, \ldots, \Phi_k$. Phase Φ_0 is special and is only used during initialization, while phases $\Phi_1, \ldots, \Phi_k$ can be thought of as corresponding to the batches $\pi_1, \ldots, \pi_k$ of updates to the router decomposition algorithm. For all $1 \leq \tau \leq k$, the pruning algorithm receives as input an online sequence E'_τ of edges that are deleted from $W_k^{N,\Delta}$ over the course of the phase Φ_τ. We emphasize that these edges do not arrive simultaneously, but rather their arrival occurs over the course of the phase Φ_τ. Intuitively, the algorithm needs to maintain subsets $U_1, \ldots, U_k$ of vertices of $W_k^{N,\Delta}$ with $U_k \subseteq \cdots \subseteq \cdots U_1$, and a subgraph $W \subseteq W_k^{N,\Delta}$, such that, at all times, W is a properly pruned subgraph of $W_k^{N,\Delta}$ with respect to $U_1, \ldots, U_k$. We require that, for all $0 \leq \tau \leq k$,

the total number of vertices deleted from U_k over the course of the phase is not much larger than $|E'_\tau|/\Delta$, and the total number of edges deleted from W over the course of the phase is not much larger than $|E'_\tau|$. We now define a new problem, that we call NiceRouterPruning, which abstracts our pruning algorithm.

Definition 4.7 (The NiceRouterPruning Problem) *The input to the* NiceRouterPruning *problem consists of parameters $k \geq 256$, $N \geq k^{3k}$ and $\Delta \geq 4k^2$, and an online sequence $e_1, \ldots, e_r$ of edge deletions from the corresponding graph $W_k = W_k^{N,\Delta}$. We denote the resulting dynamic graph by H, so $H^{(0)} = W_k$, and, for $t > 0$, $H^{(t)} = H^{(0)} \setminus \{e_1, \ldots, e_t\}$.*

The algorithm for the problem is required to maintain subsets $U_1, \ldots, U_k$ of vertices of H with $U_k \subseteq \cdots \subseteq U_1$, and a subgraph $W \subseteq H$ for which the following properties hold:

- *for all $1 \leq i \leq k$, at the beginning of the algorithm, $U_i = V(W_k)$, and, as the algorithm progresses, vertices may be deleted from U_i, but no vertices may be inserted into U_i;*
- *the only changes that graph W undergoes is the deletion of edges and vertices; and*
- *at all times, W must be a properly pruned subgraph of W_k with respect to the current sets $U_1, \ldots, U_k$.*

The algorithm's timeline is partitioned into $k+1$ phases $\Phi_0, \Phi_1, \ldots, \Phi_k$, and the algorithm is notified when a new phase starts. For all $1 \leq \tau \leq k$, if we denote by E'_τ the sequence of edge deletions from W_k that algorithm receives as input during the phase Φ_τ, then it is guaranteed that $|E'_0| \leq \frac{N^k \cdot \Delta}{2k^{4k}}$, and, for all $1 \leq \tau \leq k$, $|E'_\tau| \leq \frac{N^k \cdot \Delta}{k^{8k}}$. We require that, for all $0 \leq \tau \leq k$, the total number of vertices deleted from U_k over the course of phase Φ_τ is at most $\frac{|E'_\tau|}{\Delta} \cdot k^{4k}$; the total number of vertices deleted from U_1 over the course of pahse Φ_τ is at most $\frac{2k^{7k+2} \cdot |E'_\tau|}{\Delta}$, and the total number of edges deleted from W over the course of phase Φ_τ is at most $|E'_\tau| \cdot k^{7k+3}$.

The main result of the current subsection is the following theorem, that provides an efficient algorithm for the NiceRouterPruning problem.

Theorem 4.8 (Algorithm for NiceRouterPruning) *There is a deterministic algorithm for the* NiceRouterPruning *problem, whose initialization time is $O(k^2 \cdot N^k)$, and the running time of each phase Φ_τ, for $0 \leq \tau \leq k$, is bounded by $O\left(|E'_\tau| \cdot k^{O(k)}\right)$.*

The remainder of this subsection is dedicated to the proof of Theorem 4.8. At the beginning of the algorithm, we set $U_1 = U_2 = \cdots = U_k = V(W_k)$ and $W = W_k$. We start by describing the data structures that the algorithm maintains, together with the algorithm for their initialization. After that we describe our algorithm for implementing a single phase.

4.4.1 Data Structures and their Initialization

For every superedge $\hat{e} = (u, v)$ and phase $0 \leq \tau \leq k$, our algorithm maintains a counter $n_\tau(\hat{e})$, that counts the number of copies of the superedge that have been deleted from H during Phase Φ_τ. All such counters are initialized to 0.

For every level $1 \leq i \leq k$, for every level-i star $S \in \mathcal{S}_i$, and for every phase $0 \leq \tau \leq k$, our algorithm maintains a counter $n'_\tau(S)$, that counts the number of leaf vertices of S that have been deleted from U_i over the course of Phase Φ_τ. All these counters are initialized to 0.

Lastly, for every level $1 < i \leq k$, for every level-$(i-1)$ cluster $C \in \mathcal{C}_{i-1}$, our algorithm maintains a counter $n''(C)$, that counts the number of vertices of C that have been deleted from U_i over the course of the entire algorithm so far. All these counters are initialized to 0.

It is easy to verify that all these data structures can be initialized in time $O(k^2 \cdot N^k)$.

We now turn to describe our algorithm for implementing a single phase.

4.4.2 Algorithm for a Single Phase

Our algorithm will ensure that, for all $0 \leq \tau \leq k$, at all times t during Phase Φ_τ, the following invariants hold:

I1. For every level $1 \leq i \leq k$, for every leaf vertex v that lies in U_i at time t, the total number of level-i edges that are incident to v in W_k and were deleted over the course of Phase Φ_τ so far is at most $\frac{\Delta}{4k}$. Moreover, if $v \notin U_i$, then no level-i edges are incident to v in the current graph W;

I2. For every level $1 \leq i \leq k$, for every level-i star $S \in \mathcal{S}_i$, if the center vertex of S currently lies in U_i, then the total number of leaf vertices of S that have been deleted from U_i by the algorithm over the course of the current phase so far is at most $\frac{N}{4k^2}$. Moreover, if the center vertex of S does not lie in U_i at time t, then no leaf vertex of S lies in U_i at time t; and

I3. For every level $1 < i \leq k$, for every level-$(i-1)$ cluster $C \in \mathcal{C}_{i-1}$ of W_k, if $V(C) \cap U_1 \neq \emptyset$ currently holds, then $|V(C) \cap U_i| \geq \frac{\hat{n}(C)}{16k^2}$ must hold, where $\hat{n}(C)$ is the number of vertices of C that lied in U_i at the beginning of the current phase.

Observation 4.9 *Assume that the above invariants hold throughout the algorithm, and that $V(W) = U_1$ holds at all times. Then, at all times t, the current graph W is a properly pruned subgraph of W_k with respect to the current sets $U_1, \ldots, U_k$.*

Proof: Consider any time t during the algorithm's execution. Let $1 \leq i \leq k$ be any level, and let v be a leaf vertex that lies in U_i at time t. Let $\hat{e} = (u, v)$ be the unique level-i superedge incident to v. Then, from Invariant I1, the total number of copies of $\hat{e}$ deleted from H by the adversary up to time t is bounded by $\sum_{\tau=0}^{k} \frac{\Delta}{4k} < \frac{\Delta}{2}$. If $v \notin U_i$ at time t, then Invariant I1 ensures that no level-i edges are incident to v in the current graph W. Therefore, Property P1 holds throughout the algorithm.

Consider now a level $1 \leq i \leq k$, and a level-i star $S \in \mathcal{S}_i$, whose center vertex v lies in U_i at time t. Then, from Invariant I2, the total number of leaf vertices of S that have been deleted from U_i by the algorithm so far is at most $\sum_{\tau=0}^{k} \frac{N}{4k^2} \leq \frac{N}{k}$. If the center vertex of S does not lie in U_i at time t, then Invariant I2 ensures that no leaf vertex of S lies in U_i at time t. Therefore, Property P2 holds throughout the algorithm.

Lastly, consider a level $1 < i \leq k$, and a level-$(i-1)$ cluster $C \in \mathcal{C}_{i-1}$ of W_k, for which $V(C) \cap U_1 \neq \emptyset$ currently holds. Let τ be the index of the current phase at time t. For all $1 \leq \tau' \leq \tau$, let $\hat{n}_{\tau'}(C)$ be the cardinality of the set $V(C) \cap U_i$ at the beginning of Phase $\Phi_{\tau'}$. Then, from Invariant I3, for all $1 \leq \tau' < \tau$, $\hat{n}_{\tau'+1}(C) \geq \frac{\hat{n}_{\tau'}(C)}{16k^2}$, and moreover, at time t, $|V(C) \cap U_i| \geq \frac{\hat{n}_\tau(C)}{16k^2}$. Therefore, at time t:

$$|V(C) \cap U_i| \geq \frac{\hat{n}_\tau(C)}{16k^2} \geq \frac{|V(C)|}{(16k^2)^\tau} \geq \frac{|V(C)|}{k^{3k}},$$

since $k \geq 32$ and $\tau \leq k$. Therefore, Property P4 holds throughout the algorithm.

Since the algorithm explicitly ensures that $V(W) = U_1$ holds at all time, we conclude that, at all times t, the current graph W must be a properly pruned subgraph of W_k with respect to the current sets $U_1, \ldots, U_k$. □

From now on, we fix an index $0 \leq \tau \leq k$, and consider Phase Φ_τ. It is now enough for us to provide an algorithm for implementing Phase Φ_τ that ensures that Invariants I1–I3 hold over the course of the algorithm, and that $V(W) = U_1$ always holds. For convenience, we denote Φ_τ by Φ, and we denote counters $n_\tau(\hat{e})$ for superedges $\hat{e}$ and $n'_\tau(S)$ for stars S, by $n(\hat{e})$ and $n'(S)$, respectively. We also denote the set of edges that the adversary deletes from H over the course of the phase by E' (this set is not known to the algorithm beforehand). It is enough for us to ensure that the running time of the algorithm to implement the phase is bounded by $O(|E'| \cdot k^{O(k)})$; that the total number of vertices deleted from U_1 and from U_k over the course of the phase is bounded by $\frac{|E'| \cdot 2k^{7k+2}}{\Delta}$ and $\frac{|E'|}{\Delta} \cdot k^{4k}$ respectively; and that the total number of edges deleted from W over the course of the phase is bounded by $|E'| \cdot k^{7k+3}$.

The Algorithm Description

We now formally describe the algorithm for implementing Phase Φ. Throughout the algorithm, we maintain a list Λ, that can be thought of as a "to do" list. This list contains entries of three types. An entry of type 1 is a pair (i, v), where $1 \leq i \leq k$ is a level, $v \in U_i$ is a vertex, and the entry indicates that v should be deleted from U_i. The *level* of entry (i, v) is i. An entry of type 2 is a pair (i, S), where $1 \leq i \leq k$ is a level, and $S \in \mathcal{S}_i$ is a level-i star. The entry indicates that star S must be destroyed, either because its center vertex was just deleted from U_i, or because the number of leaf vertices of S that were deleted from U_i over the course of the current phase is at least $\frac{N}{4k^2}$. The *level* of entry (i, S) is i. Lastly, an entry of type 3 is a pair (i, C), where $1 < i \leq k$ is a level, and $C \in \mathcal{C}_{i-1}$ is a level-$(i-1)$ cluster. The entry indicates that cluster C needs to be destroyed, because $|V(C) \cap U_i|$ fell below $\frac{\hat{n}(C)}{16k^2}$, where $\hat{n}(C)$ is the number of vertices of C that lied in U_i at the beginning of the current phase. The *level* of entry (i, C) is i.

We are now ready to describe the algorithm for processing the deletion of a single edge $e = (u, v)$ from graph H. If e does not lie in the current graph W, then no further processing is needed. Assume now that e lies in W, and let $\hat{e} = (u, v)$ be the corresponding superedge. We increase the counter $n(\hat{e})$ by 1. If counter $n(\hat{e})$ remains bounded by $\frac{\Delta}{4k}$, then no further updates are needed. From now on we assume that $n(\hat{e}) > \frac{\Delta}{4k}$ holds. We assume w.l.o.g. that v is a leaf vertex, and u is a center vertex. Let $1 \leq i \leq k$ be the level to which superedge (u, v) belongs. We initialize the list $\Lambda = \{(i, v)\}$, indicating that vertex v must be deleted from set U_i. We say that entry (i, v) was added to Λ *directly*. Next, we iterate, as long as $\Lambda \neq \emptyset$. In every iteration, we select a single entry of Λ to process. The entry that we choose to process in each iteration must be that of the smallest level in Λ, and among entries of the same level, it must be of the lowest type (that is, type-1 entries are processed before type-2 entries, which are processed before type-3 entries of a single level). We now describe the procedure for processing each entry.

Processing type-1 entries. Suppose the entry of Λ that needs to be processed in the current iteration is of type 1, and denote it by (i, v), where v is a vertex that currently lies in U_i. We start by deleting vertex v from U_i, and removing entry (i, v) from Λ. We also delete all level-i edges that are incident to v from W. If $i < k$, then we add type-1 entry $(i+1, v)$ to Λ, so that v is also deleted from subsequent layers. In this case, we say that entry $(i+1, v)$ was added to Λ *directly*. Next, we consider the unique level-i star S that contains v. If S is already marked as destroyed then no further updates to it needed. Otherwise, if v is the center of the star S, then we add a type-2 entry (i, S) to Λ, to

indicate that the star S needs to be destroyed. Assume now that v is a leaf of S. We increase the counter $n'(S)$ by 1, and, if the counter becomes greater than $\frac{N}{4k^2}$, then we add a type-2 entry (i, S) to Λ, indicating that star S must be destroyed. Lastly, if $i > 1$, then then we also consider the level-$(i-1)$ cluster C of W_k to which v belongs. If no vertex of C has been deleted from U_i in the current phase yet, then we mark C to indicate that one of its vertices was deleted from U_i in the current phase, and we record that the number of vertices of C that lied in U_i at the beginning of the current phase is $\hat{n}(C) = |V(C)| - n''(C)$. If C is already marked as destroyed, then no further processing of C is needed. Otherwise, we increase $n''(C)$ by 1, and, if $|V(C)| - n''(C) < \frac{\hat{n}(C)}{16k^2}$ holds, we add a type-3 entry (i, C) to Λ, indicating that cluster C must be destroyed.

Processing type-2 entries. Assume now that the entry of Λ that needs to be processed in the current iteration is of type 2, and denote it by (i, S), where S is a level-i star. For every vertex $x \in V(S)$ that still lies in U_i, such that entry (i, x) does not currently lie in Λ, we add the type-1 entry (i, x) to Λ, and we say that this entry was added to Λ *indirectly*. We then delete entry (i, S) from the list Λ, and we mark S as being destroyed.

Processing type-3 entries. Finally, assume that the entry of Λ that needs to be processed in the current iteration is of type 3, and denote it by (i, C), where C is a level-$(i-1)$ cluster. For every vertex $x \in V(C) \cap U_1$, we delete x from sets $U_1, \ldots, U_{i-1}$, and we delete from W all edges of levels $1, \ldots, i-1$ that are incident to x in the current graph W. Lastly, we add a type-1 entry (i, x) to list Λ, and we say that the entry was added to the list *indirectly*. We delete entry (i, C) from Λ, and we mark C as being destroyed.

The algorithm for processing a single edge deletion from H terminates when Λ becomes empty. Notice that, if some vertex v was deleted from set U_1 during the update procedure, then v was also deleted from $U_2, \ldots, U_k$, and it has become an isolated vertex in W. In such a case, we also delete v from $V(W)$, to ensure that $V(W) = U_1$ always holds.

For all $1 \leq i \leq k$, let U_i^D be the set of vertices that were deleted from U_i over the course of the update procedure, and let $M = \sum_{i=1}^{k} |U_i^D|$. Then the running time of the procedure is $O(1) + O(M \cdot \Delta)$, and the total number of edges deleted from W over the course of the procedure is at most $M\Delta$. This is since, when a center vertex u is deleted from U_i by the algorithm, every leaf vertex that lies in the level-i star S for which u serves as a center is also deleted from U_i, and so the deletion of the level-i edges incident to u in H can be charged to the corresponding leaf vertices; the deletion of a leaf vertex v from a set U_i can trigger the deletion of at most Δ edges from W. It is immediate to verify that, at the end of the update procedure Invariants I1–I3 hold.

Consider now some level $1 \leq i \leq k$, and a type-1 level-i entry (i, v) that ever lied in Λ over the course of the update procedure. Each such entry was added to Λ either directly or indirectly. The entry can only be added to Λ indirectly during the processing of a level-i type-2 entry (i, S), if v is a vertex of the star S that is being destroyed, or during the processing of a level-i type-3 entry (i, C), where C is a level-$(i-1)$ cluster being destroyed, and $v \in V(C)$. Entry (i, v) may be added to Λ directly only in one of the following two cases: either (i) while processing a level-$(i-1)$ type-1 entry $(i-1, v)$; or (ii) if edge (u, v) was just deleted from H by the adversary, v is a leaf vertex, and the number of copies of superedge (u, v) deleted from H in the current phase became greater than $\frac{\Delta}{4k}$. We denote by J_i the set of all leaf vertices v that lie in U_i at the beginning of the phase, so that at least $\frac{\Delta}{4k}$ level-i edges incident to v lie in E'. Clearly, $|J_i| \leq \frac{4k|E'|}{\Delta}$.

We denote by R_i the set of all vertices v, such that type-1 entry (i, v) ever lied in Λ_i during the current phase, and it was added to Λ_i directly. We also include in R_i all vertices of J_i, even if, for some such

vertex v, entry (i,v) was first added to Λ indirectly. Similarly, we denote by R'_i the set of all vertices $v \notin J_i$, such that entry (i,v) ever lied in Λ_i during the current phase, and it was added to Λ_i indirectly. Lastly, we denote by R''_i all vertices v that were deleted from U_i during the current phase, but the type-1 entry (i,v) never lied in Λ during the current phase. The latter may only happen if, for some level $i' > i$, the level-$(i'-1)$ cluster C containing v was destroyed after the corresponding type-3 entry (i', C) was added to Λ.

Notice that, when a type-3 entry (i, C) is processed, every vertex of C that currently lies in U_1 is deleted from sets $U_1, \ldots, U_{i-1}$. Since, as we showed in Observation 4.9, at the beginning of the current phase Property P4 held, we get that $\hat{n}(C) \geq \frac{|V(C)|}{k^{3k}}$ at the beginning of the phase. By the time the algorithm starts processing entry (i, C), at least $\frac{\hat{n}(C)}{2} \geq \frac{|V(C)|}{2k^{3k}}$ type-1 entries (i, x) for vertices $x \in V(C)$ have been added to Λ, and so at least $\frac{|V(C)|}{2k^{3k}}$ vertices of C lie in $R_i \cup R'_i$. Intuitively, we can *charge* these vertices for the deletion of the vertices of C from sets $U_1, \ldots, U_{i-1}$, when cluster C is destroyed. Let Π denote the collection of all pairs (i, x), where x is a vertex that was deleted from U_i over the course of the current phase. Then, from the above discussion:

$$|\Pi| \leq 2k \cdot k^{3k} \cdot \sum_{i=1}^{k} (|R_i| + |R'_i|) \leq k^{3k+2} \sum_{i=1}^{k} (|R_i| + |R'_i|).$$

Additionally, from our discussion, the total number of edges deleted from W over the course of the current phase is at most:

$$|E'| + \Delta \cdot \Pi \leq |E'| + \Delta \cdot k^{3k+2} \sum_{i=1}^{k} (|R_i| + |R'_i|),$$

and the total running time for processing the current phase is bounded by $O(|E'|) + O(\Delta \cdot |\Pi|) \leq O(|E'|) + k^{O(k)} \cdot \Delta \cdot \sum_{i=1}^{k} (|R_i| + |R'_i|)$.

Notice also that $R''_k = \emptyset$, and so the total number of vertices deleted from U_k in the current phase is bounded by $|R_k| + |R'_k|$. Next, we will bound the cardinalities of the sets R_i, R'_i of vertices for $1 \leq i \leq k$. This will allow us to bound the running time of a phase, the number of edges deleted from W over the course of the phase, and the total number of vertices deleted from U_1 and from U_k in the current phase.

Before we proceed, consider an index $1 \leq i \leq k$, and denote $\lambda_i = |R_i|$. Consider now some center vertex $v \in R_i$. Recall that the only way that entry (i, v) may be added to Λ directly when v is a center vertex, is if the type-1 entry $(i-1, v)$ was processed by the algorithm. If S is the unique level-$(i-1)$ star whose center is v, then star S must have been destroyed by our algorithm, and so every leaf vertex x of S that lied in U_{i-1} at the beginning of the current phase was deleted from U_{i-1}, and the corresponding type-1 entry (i, x) was added to Λ directly. Since, from Property P2, at least $N/2$ leaf vertices of S lied in U_{i-1} at the beginning of the phase, we get that, for each each center vertex $v \in R_i$, there are at least $N/2$ leaf vertices that lie in R_i, and these sets of leaf vertices that correspond to each center vertex are disjoint. In other words, the total number of center vertices in R_i must be bounded by $2|R_i|/N$.

4.4.3 Analysis

In this subsection, we again consider the phase Φ, and the set $E' = \{e_1, \ldots, e_q\}$ of edges that were deleted from H during this phase. Our goal is to bound the cardinalities of the sets R_i and R'_i for all

$1 \leq i \leq k$. Recall that we have denoted $\lambda_i = |R_i|$. We start with the following claim.

Claim 4.10 *For all* $1 \leq i \leq k$, $|R_i| + |R'_i| \leq 32k^2\lambda_i$.

Proof: Let U'_i be the set U_i at the beginning of the current phase. In order to prove the claim, we assign, to every vertex $v \in U'_i$, a potential $\psi(v)$, that may change over the course of the phase.

Recall that we denoted by J_i the set of all leaf vertices $v \in U'_i$, so that at least $\frac{\Delta}{4k}$ level-i edges incident to v lie in E'. For the sake of the analysis, we will view the set R_i of vertices as fixed, so R_i is the set of all vertices v, whose corresponding type-1 entry (i, v) was ever added to Λ directly during the current phase. This set also includes all vertices in J_i, even if, for some such vertex v, entry (i, v) was first added to Λ indirectly. We also define a dynamically evolving set $\hat{R}_i$, that, at any time t during the current phase, contains all vertices $v \in U'_i \setminus R_i$, whose corresponding type-1 entry (i, v) was added to Λ indirectly, since the beginning of the phase, and until time t (inclusive). We emphasize that set $\hat{R}_i$ does not contain vertices of J_i, even if, for some such vertex v, entry (i, v) was first added to Λ indirectly. Therefore, at the beginning of the phase, $\hat{R}_i = \emptyset$, at the end of the phase, $\hat{R}_i = R'_i$, and over the course of the phase, vertices may join $\hat{R}_i$, but they may never leave this set.

Consider some time t during the execution of the algorithm, and let $v \in U'_i$ be any vertex. If $v \notin R_i \cup \hat{R}_i$, then we set the potential $\psi(v) = 0$.

Assume now that $v \in R_i \cup \hat{R}_i$ currently holds. We now define values $\psi'(v)$ and $\psi''(v)$, and we will eventually set the potential $\psi(v) = \psi'(v) + \psi''(v)$. Let S be the unique level-i star containing v. If a type-2 entry (i, S) has not been processed in the current phase yet (either because this entry was never added to Λ, or because it was added but not yet processed), then we set $\psi'(v) = 8k^2$ if v is a leaf vertex, and we set $\psi'(v) = 2N$ if it is a center vertex. Otherwise, if entry (i, S) was processed in the current phase already, then we set $\psi'(v) = 0$.

If $i = 1$, then we set $\psi''(v) = 0$. Otherwise, let C be the unique level-$(i-1)$ cluster containing v. If type-3 entry (i, C) was not processed in the current phase yet (either because it was never added to Λ, or because it was added but was not processed yet), then we set $\psi''(v) = 8$ if $v \in J_i$ and $\psi''(v) = 2$ if $v \notin J_i$. Otherwise, if type-3 entry (i, C) was already processed, we set $\psi''(v) = 1$. Lastly, we let $\psi(v) = \psi'(v) + \psi''(v)$ be the potential of v. Notice that for every vertex $v \in U'_i$, if v is a leaf vertex then $\psi(v) \leq 8k^2 + 8$ always holds, and if v is a center vertex, then $\psi(v) \leq 2N + 2$ always holds.

Recall that vertices may be deleted from set U_i when, for some level $i' > i$, some level-$(i'-1)$ cluster is destroyed. In this proof we ignore all such updates to U_i. They do not affect the potentials of the vertices, and they do not cause the additions of any new level-i entries into Λ. Moreover, observe that, for every level-i star S, following such an update, either all vertices of S are deleted from U_i, or none of them are. Similarly, for every level-$(i-1)$ cluster C, following such an update, either all vertices of C are deleted from U_i, or none of them are. Therefore, all such updates can be ignored in this proof.

Let $\Psi = \sum_{v \in U'_i} \psi(v)$ be the total potential at the system. Note that, at the beginning of the phase, for every vertex $v \in R_i$, if v is a leaf vertex, then its potential is $\psi(v) = 8k^2 + 8 \leq 9k^2$, and if it is a center vertex, then $\psi(v) \leq 2N + 8$. Moreover, if $v \notin R_i$, then $\psi(v) = 0$. Since the number of center vertices in R_i is bounded by $\frac{2\lambda_i}{N}$ and $|R_i| = \lambda_i$, we get that $\Psi \leq 32k^2\lambda_i$. Throughout the phase, every vertex $v \in R_i \cup \hat{R}_i$ has potential $\psi(v) \geq 1$. In the remainder of the proof, we show that the potential Ψ never increases over the course of the phase. Since, at the end of the phase, the potential of every vertex in $R_i \cup \hat{R}_i$ is at least 1, it will then follow that $|R_i| + |\hat{R}_i| \leq 32k^2\lambda_i$ always holds, and so in particular, $|R_i| + |R'_i| \leq 32k^2\lambda_i$.

Notice that the potential of the vertices of U'_i may only change when a level-i entry of Λ is being processed. This is since the processing of entries whose level is $i' < i$ may only add new level-i entries

into Λ that are of type 1, and such entries (i, v) are added to Λ directly, so the corresponding vertices v lie in R_i already; the processing of entries of levels $i' > i$ may not affect the potentials of the vertices. Consider now some time t during the phase, when a level-i entry is being processed. We show that the potential Ψ may not grow as the result of processing the entry.

Type-1 entries. Assume first that the entry is of type 1, and denote it by (i, v). Then at time t, $v \in R_i \cup \hat{R}_i$ already holds. It is easy to verify that processing this entry does not change the potential of any vertex in U_i'.

Type-2 entries. Assume now that the processed entry is of type 2, and denote it by (i, S), where S is the corresponding level-i star. Let X be the set of all leaf vertices of S that currently lie in $R_i \cup \hat{R}_i$, and let Y be the set of all leaf vertices of S that lie in $U_i' \setminus (R_i \cup \hat{R}_i)$. Let u be the center vertex of S.

Assume first that $u \in R_i \cup \hat{R}_i$. Then, since entry (i, S) has not yet been processed, $\psi'(u) = 2N$ currently holds, while $|Y| \leq N$. Once entry (i, S) is processed, $\psi'(u)$ is set to 0, while $\psi''(u)$ remains unchanged. For every vertex $v \in X$, $\psi(v)$ decreases, while for every vertex $v \in Y$, $\psi(v)$ grows from 0 to at most 2 (since v may not lie in $J_i \subseteq R_i$ in this case). Overall, the potential of u decreases by at least $2N$, while the potential of the vertices in Y grows by at most $2|Y| \leq 2N$. The potential of the vertices in X may only decrease. Therefore, Ψ does not increase.

Assume now that $u \notin R_i \cup \hat{R}_i$. Then, for every vertex $v \in X$, $\psi'(v) = 8k^2$ holds before entry (i, S) is processed, and $\psi'(v) = 0$ holds after it is processed. Therefore, the potential of the vertices in X decreases by $8k^2 \cdot |X|$. For every vertex $v' \in Y \cup \{u\}$, $\psi(v')$ grows from 0 to at most 2 once entry (i, S) is procesed. So the total potential of the vertices in $Y \cup \{u\}$ increases by at most $2(|Y| + 1)$. Since entry (i, S) was added to Λ, while the center vertex of S does not lie in $R_i \cup \hat{R}_i$ and hence remains in U_i', it must be the case that $|X| \geq \frac{N}{4k^2} \geq \frac{|Y|+1}{4k^2}$. Therefore, the potential of the vertices in X decreases by at least $8k^2 \cdot |X| \geq 2(|Y| + 1)$, and overall the potential Ψ does not increase over the course of processing type-2 entries.

Type-3 entries. It now remains to consider the case where a type-3 entry (i, C) is being processed, where C is a level-$(i-1)$ cluster. Let X' be the subset of vertices of $V(C) \cap U_i'$ that lie in $R_i \cup \hat{R}_i$ and let Y' be the set containing the remaining vertices of $V(C) \cap U_i'$. Since entry (i, C) was added to Λ, $|Y'| \leq \frac{\hat{n}(C)}{16k^2}$ holds, while $|X'| = \hat{n}(C) - |Y'|$. When entry (i, C) is processed, for every vertex $v \in Y'$, entry (i, v) is added to Λ indirectly, and so v is added to $\hat{R}_i$, and the potential $\psi(v)$ may grow from 0 to $8k^2 + 1$ if v is a leaf vertex, and from 0 to $2N + 1$ if it is a center vertex.

Let $Y_c' \subseteq Y'$ be the set of all center vertices that lie in Y'. Consider any vertex $v \in Y_c'$, and let S be the unique level-$(i-1)$ star for which v serves as a center. Since our algorithm processes the entries of Λ from lower to higher levels, entry $(i-1, S)$ has not been added to Λ yet. From Invariant I2, and from the fact that W is a properly pruned subgraph of W_k at all times, at least $\frac{3N}{4}$ leaf vertices of S currently lie in U_{i-1}; we denote the set of all these leaf vertices by $Z(v)$. Notice that $Z(v) \subseteq V(C)$ must hold. Consider now a vertex $u \in Z(v)$. From our discussion, when entry (i, C) is processed, entry $(i-1, u)$ has not been added to Λ yet. Since u is deleted from U_{i-1} when entry (i, C) is processed, entry $(i-1, u)$ is never be added to Λ over the course of the current phase. Therefore, the only way that u may lie in R_i is if $u \in J_i$. We partition the set $Z(v)$ of vertices into two subsets: $Z'(v) = Z(v) \cap J_i$ and $Z''(v) = Z(v) \setminus J_i$. From our discussion, $Z''(v) \cap R_i = \emptyset$, and so $Z''(v) \subseteq Y'$ must hold. Since, for every vertex $v \in Y_c'$, $|Z(v)| \geq \frac{3N}{4}$ we get tha either (i) $\sum_{v \in Y_c'} |Z'(v)| \geq \frac{|Y_c'| \cdot N}{2}$, or (ii) $\sum_{v \in Y_c'} |Z''(v)| \geq \frac{|Y_c'| \cdot N}{4}$ must hold. We consider each of these two cases separately.

Assume first that $\sum_{v\in Y'_c}|Z''(v)| \geq \frac{|Y'_c|\cdot N}{4}$. It is easy to verify that the vertex sets $\{Z''(v)\}_{v\in Y'_c}$ are mutually disjoint, and, as observed already, for every vertex $v\in Y'_c$, $Z''(v)\subseteq Y'$ must hold. Therefore, in this case, $|Y'_c|\leq \frac{4|Y'|}{N}$. Altogether, the total potential $\psi(v)$ of all vertices $v\in Y'$ grows by at most:

$$\begin{aligned}(8k^2+1)\cdot|Y'|+(2N+1)\cdot|Y'_c| &\leq (8k^2+1)\cdot|Y'|+(2N+1)\cdot\frac{4|Y'|}{N}\\ &\leq (8k^2+10)\cdot|Y'|\\ &\leq (8k^2+10)\cdot\frac{\hat{n}(C)}{16k^2}.\end{aligned}$$

At the same time, after entry (i,C) is processed, the potential $\psi(v')$ of every vertex $v'\in X'$ decreases by at least 1. Since $|X'|=\hat{n}(C)-|Y'|\geq 0.99\hat{n}(C)$, the total potential Ψ does not grow.

It remains to consider the case where $\sum_{v\in Y'_c}|Z'(v)|\geq\frac{|Y'_c|\cdot N}{2}$. As before, the vertex sets $\{Z'(v)\}_{v\in Y'_c}$ are mutually disjoint. As observed already, for every vertex $v\in Y'_c$, $Z'(v)\subseteq J_i$ must hold. Therefore, in this case, $|Y'_c|\leq\frac{2|V(C)\cap J_i|}{N}$. Altogether, the total potential $\psi(v)$ of all vertices $v\in Y'$ grows by at most:

$$\begin{aligned}(8k^2+1)\cdot|Y'|+(2N+1)\cdot|Y'_c| &\leq (8k^2+1)\cdot|Y'|+(2N+1)\cdot\frac{2|V(C)\cap J_i|}{N}\\ &\leq (8k^2+1)\cdot\frac{\hat{n}(C)}{16k^2}+5|V(C)\cap J_i|\\ &\leq 0.51\hat{n}(C)+5|V(C)\cap J_i|.\end{aligned}$$

(We have used the fact that $k\geq 256$.) At the same time, after entry (i,C) is processed, the potential $\psi(v')$ of every vertex $v'\in X'$ decreases by at least 1, while the potential of every vertex in $V(C)\cap J_i$ decreases by at least 7. Therefore, the total decrease in the potential is at least:

$$7|V(C)\cap J_i|+(|X'|-|V(C)\cap J_i|)\geq 5|V(C)\cap J_i|+|X'|\geq 5|V(C)\cap J_i|+0.51\hat{n}(C).$$

(We have used the fact that $|X'|=\hat{n}(C)-|Y'|\geq 0.99\hat{n}(C)$). Altogether, in either case, the total potential does not grow when a type-3 entry of Λ is processed.

We conclude that, throughout the phase, $\Phi\leq 32k^2\lambda_i$ must hold. Since, throughout the phase, for every vertex $v\in R_i\cup\hat{R}_i$, $\psi(v)\geq 1$ holds, and since, at the end of the phase, $\hat{R}_i=R'_i$, we get that $|R_i|+|R'_i|\leq 32k^2\lambda_i$.

□

We now obtain the following easy corollary.

Corollary 4.11 *For all $1\leq i\leq k$, $|R_i\cup R'_i|\leq k^{4i}\cdot\frac{q}{\Delta}$, where $q=|E'|$ is the number of edges deleted from H in the current phase.*

Proof: Consider the set E' of edges deleted over the course of the algorithm, and let $\hat{E}$ contain all superedges $\hat{e}=(u,v)$, such that at least $\frac{\Delta}{4k}$ copies of $\hat{e}$ lie in E'. Clearly, $|\hat{E}|\leq\frac{4qk}{\Delta}$. For all $1\leq i\leq k$, let $\hat{E}_i\subseteq\hat{E}$ contain the subset of level-i superedges of $\hat{E}$. Recall that we denoted by J_i be the set of all leaf vertices v that lie in U_i at the beginning of the phase and serve as endpoints of the superedges in $\hat{E}_i$. Clearly, $|J_i|\leq|\hat{E}|\leq\frac{4qk}{\Delta}$ for all i.

We prove the corollary by induction. The induction base is when $i = 1$. Notice that the only type-1 entries $(1, v)$ that are added to Λ directly over the course of the phase are those corresponding to vertices $v \in J_1$, so $\lambda_1 = |R_1| \leq \frac{4qk}{\Delta}$. From Claim 4.10 we then get that $|R_1| + |R'_1| \leq 32k^2\lambda_1 \leq \frac{128k^3q}{\Delta} \leq \frac{k^4q}{\Delta}$, since $k \geq 256$.

Consider now an integer $i > 1$, and assume that the claim holds for $i - 1$. Note that the only type-1 entries (i, v) that are added to Λ directly are those corresponding to vertices $v \in J_i$, or those whose corresponding level-$(i-1)$ entry $(i-1, v)$ was added to Λ as well. Therefore, $\lambda_i = |R_i| \leq |J_i| + |R_{i-1}| + |R'_{i-1}| \leq \frac{4qk}{\Delta} + k^{4i-4} \cdot \frac{q}{\Delta} \leq 2k^{4i-4} \cdot \frac{q}{\Delta}$. From Claim 4.10, we then get that $|R_i| + |R'_i| \leq 32k^2\lambda_i \leq 64k^{4i-2} \cdot \frac{q}{\Delta} \leq k^{4i} \cdot \frac{q}{\Delta}$. □

We conclude that the total number of vertices deleted from U_k over the course of the phase is bounded by $|R_k \cup R'_k| \leq k^{4k} \cdot \frac{q}{\Delta}$. From our discussion, the running time of the phase is bounded by:

$$O(q) + k^{O(k)} \cdot \Delta \cdot \sum_{i=1}^{k} (|R_i| + |R'_i|) \leq O\left(k^{O(k)} \cdot q\right).$$

Recall that we denoted by Π the collection of all pairs (i, v), such that v is a vertex that was deleted from U_i in the current phase, and we have shown that $|\Pi| \leq k^{3k+2} \sum_{i=1}^{k} (|R_i| + |R'_i|)$. We then get that the total number of vertices deleted from U_1 over the course of the current phase is bounded by:

$$|\Pi| \leq k^{3k+2} \sum_{i=1}^{k} (|R_i| + |R'_i|) \leq \frac{2k^{3k+2} \cdot k^{4k} \cdot q}{\Delta} \leq \frac{2k^{7k+2} \cdot q}{\Delta}.$$

Lastly, from our discussion, the total number of edges deleted from W in this phase is bounded by:

$$q + \Delta \cdot k^{3k+2} \cdot \sum_{i=1}^{k} (|R_i| + |R'_i|) \leq q + 2q \cdot k^{3k+2} \cdot k^{4k} \leq q \cdot k^{7k+3}.$$

5 Second Tool: Embedding a Nice Router

One of the main notions that we use in this section is that of a scattered graph, that we define next.

Definition 5.1 (Scattered Graph.) *Let G be a graph, and let $d \geq 1$ and $0 < \epsilon < 1$ be parameters. We say that G is* (d, ϵ)-scattered *if, for every vertex $v \in V(G)$, $|B_G(v, d)| \leq |V(G)|^{1-\epsilon}$.*

Next, we provide a simple algorithm, that, given a graph G and parameters d and ϵ, either correctly certifies that G is (d, ϵ)-scattered, or computes a vertex $v \in V(G)$, such that $|B_G(v, 4d/\epsilon)| \geq |V(G)|^{1-\epsilon}$. The proof follows standard techniques and is deferred to Section C of Appendix.

Claim 5.2 *There is an algorithm, that, given an n-vertex graph G and parameters $d \geq 1$ and $0 < \epsilon < 1$, either correctly certifies that G is (d, ϵ)-scattered, or computes a vertex $v \in V(G)$, such that $|B_G(v, 4d/\epsilon)| \geq |V(G)|^{1-\epsilon}$. The running time of the algorithm is $O(m^{1+\epsilon})$, where $m = |E(G)|$.*

Next, we define a new problem, that we call EmbedOrScatter. The input to the problem consists of an n-vertex graph G, and parameters Δ and k. Additionally, we are given a distance parameter d, and a congestion parameter η. The goal of the algorithm is to either (i) compute an embedding of graph $W_k^{N,\Delta}$ into G via paths whose length is not much larger than d, that cause congestion at most

η, where parameter N is chosen so that $N^k \geq n^{1-2/k}$; or (ii) compute a set E' of at most $2\Delta n$ edges of G, such that $G \setminus E'$ is $(d, 1/k)$-scattered. In the former case, we also allow the algorithm to compute a small subset $F \subseteq E(W_k^{N,\Delta})$ of edges, that we call *fake edges*, for which the algorithm does not need to provide an embedding. In addition to the above input, the algorithm is given a parameter $0 < \delta < 1$, that will determine the tradeoff between the lengths of the embedding paths (which will be bounded by $d \cdot k \cdot 2^{O(1/\delta^6)}$), and the running time of the algorithm (that will depend on m^δ, where $m = |E(G)|$). We now provide a formal definition of the EmbedOrScatter problem.

Definition 5.3 (EmbedOrScatter problem) *The input to the* EmbedOrScatter *problem is a simple n-vertex graph G with no isolated vertices, and parameters $16 \leq \Delta \leq n$, $d \geq 1$, $4 \leq k < (\log n)^{1/3}$, and $\frac{2}{(\log n)^{1/24}} < \delta < \frac{1}{400}$, such that Δ, k and $1/\delta$ are integers. We require that n is sufficiently large, so that $\sqrt{\log n} > 100 \log\log n$ holds. Let $N = \left\lfloor n^{1/k - 1/k^2} \right\rfloor$ and let $\eta = n^{4/k} \cdot d \cdot k^{ck} \cdot 2^{c/\delta^6}$ for a large enough constant c. The output of the problem is one of the following:*

- *either a collection $F \subseteq E(W_k^{N,\Delta})$ of at most $\frac{N^k \cdot \Delta}{(512k)^{4k}}$ edges of $W_k^{N,\Delta}$ (that we call* fake edges*), together with an embedding of $W_k^{N,\Delta} \setminus F$ into G via paths of length at most $d \cdot k \cdot 2^{O(1/\delta^6)}$ that cause congestion at most η; or*
- *a subset E' of at most $2n\Delta$ edges, such that $G \setminus E'$ is $(d, 1/k)$-scattered.*

The main goal of the current section is to provide an algorithm for solving the EmbedOrScatter problem, that is summarized in the following theorem.

Theorem 5.4 *There is a deterministic algorithm for the* EmbedOrScatter *problem, whose running time on input $(G, \Delta, k, d, \eta, \delta)$ is at most $O\left(m^{1+O(\delta+1/k)} \cdot d^2 \cdot 2^{O(1/\delta^6)}\right)$, where $m = |E(G)|$.*

The remainder of this section is dedicated to the proof of Theorem 5.6. We start by defining a slightly different problem, called EmbedOrSeparate, and then show that it is sufficient to obtain an efficient algorithm for EmbedOrSeparate in order to prove Theorem 5.4. We then provide an efficient algorithm for EmbedOrSeparate.

5.1 EmbedOrSeparate Problem

In this subsection, we define the EmbedOrSeparate problem, and state our main result for this problem, namely an algorithm that solves the problem efficiently. We then show that this algorithm can be used in order to prove Theorem 5.4.

Intuitively, in the EmbedOrSeparate problem, we are given a graph G and a subset T of its vertices that we call terminals. We are also given parameters Δ, N, k, together with a distance parameter d. We require that $|T| = N^k$, and every vertex in T has degree at least Δ in G. The goal is to either (i) compute a large subset $T^{\text{del}} \subseteq T$ of terminals together with a relatively small set E^{del} of edges of G such that, for every terminal $t \in T^{\text{del}}$, $B_{G \setminus E'}(t, d)$ contains relatively few terminals; or (ii) to compute an embedding of $W_k^{N,\Delta}$ into G, via paths that are relatively short and cause a relatively small congestion, such that the vertices of $W_k^{N,\Delta}$ are mapped to the terminals in T. In the latter case, we allow the algorithm to select a small subset $F \subseteq E(W_k^{N,\Delta})$ of edges, that we call *fake edges*, that do not need to be embedded, so the algorithm only needs to embed the non-fake edges of $W_k^{N,\Delta}$. In addition to the parameters N, Δ and k, that are used in the definition of the graph $W_k^{N,\Delta}$, and the distance parameter d, the algorithm is given as input a congestion parameter η, specifying the desired

bound on the congestion caused by the embedding paths, and an additional parameter $0 < \delta < 1$. The latter parameter plays a role in ensuring that the resulting algorithm is efficient, and is used in a tradeoff between the running time of the algorithm and the lengths of the embedding paths. We now define the EmbedOrSeparate problem formally.

Definition 5.5 (EmbedOrSeparate problem) *The input to the* EmbedOrSeparate *problem is a graph G, a subset $T \subseteq V(G)$ of its vertices that we call* terminals*, and parameters $\Delta \geq 16$, $N, k, d, \eta \geq 1$ and $\frac{2}{(\log(|T|))^{1/12}} < \delta < \frac{1}{400}$, such that N, k, Δ and $1/\delta$ are integers. We require that $|T| = N^k$, that every vertex in T has degree at least Δ in G. We also require that $\eta > N^4 \cdot k \cdot d \cdot 2^{c/\delta^6}$ holds, for a large enough constant c, and that $|T| \cdot \Delta$ is sufficiently large, so that $(|T| \cdot \Delta)^{\delta^5/2} \geq 2\log(|T| \cdot \Delta)$ holds.*

The output of the problem is one of the following:

- *either a collection $F \subseteq E(W_k^{N,\Delta})$ of at most $\frac{N^k \cdot \Delta}{(512k)^{4k}}$ edges of $W_k^{N,\Delta}$ (that we call* fake edges*), together with an embedding of $W_k^{N,\Delta} \setminus F$ into G via paths of length at most $d \cdot 2^{O(1/\delta^6)}$ that cause congestion at most η, such that the vertices of $W_k^{N,\Delta}$ are mapped to the vertices of T; or*
- *a subset $T^{\mathrm{del}} \subseteq T$ of at least $\frac{N^{k-1}}{k \cdot (512k)^{4k}}$ terminals, and a subset $E^{\mathrm{del}} \subseteq E(G)$ of at most $N^{k+1} \cdot \frac{k \cdot \Delta \cdot d \cdot 2^{O(1/\delta^6)}}{\eta}$ edges, such that, for all $t \in T^{\mathrm{del}}$, $B_{G \setminus E^{\mathrm{del}}}(t, d)$ contains at most $\frac{N^k}{2}$ vertices of T.*

The following theorem summarizes our algorithm for the EmbedOrSeparate problem.

Theorem 5.6 *There is an algorithm for the* EmbedOrSeparate *problem, whose running time on input $(G, T, \Delta, N, k, d, \eta, \delta)$ is at most $O\left(m^{1+O(\delta)} \cdot \eta^2\right)$, where $m = |E(G)|$.*

We provide the proof of Theorem 5.6 in Section 5.3, after we complete the proof of Theorem 5.4 using it.

5.2 Algorithm for EmbedOrScatter– Proof of Theorem 5.4

We assume that we are given an n-vertex graph G with no isolated vertices, and parameters $16 \leq \Delta \leq n$, $d \geq 1$, $2 \leq k < (\log n)^{1/3}$, and $\frac{2}{(\log n)^{1/24}} < \delta < \frac{1}{400}$, such that Δ, k and $1/\delta$ are integers. We also assume that n is sufficiently large, so that $\sqrt{\log n} > 100 \log\log n$ holds. Let $N = \left\lfloor n^{1/k - 1/k^2} \right\rfloor$ and let $\eta = n^{4/k} \cdot d \cdot k^{ck} \cdot 2^{c/\delta^6}$ for a large enough constant c.

The algorithm consists of a number of phases. Throughout the algorithm, we maintain a set E' of deleted edges, that is initialized to $E' = \emptyset$, and then we may gradually add edges to E'. We will ensure that $|E'| \leq 2n\Delta$ always holds. We also maintain a graph $G' = G \setminus E'$. Let $d' = 8kd$. We now describe the ith phase, for $i \geq 1$.

Description of the ith phase. We start with a preprocessing step, where, for every vertex $v \in V(G')$, if $\deg_{G'}(v) < \Delta$, then we add all edges incident to v in G' to E', and delete them from G'. We continue this process until, for every vertex v of G', if v is not isolated in G', then its degree is at least Δ. If e is an edge that is added to E' during this step, then we say that e is added to E' *directly*. Notice that the total number of edges that are aver added to E' directly is bounded by $n \cdot \Delta$. The preprocessing step can be implemented in time $\tilde{O}(m)$: we start by computing the degree of every vertex in G', and then maintain a heap H, containing all non-isolated vertices of G', with the key being the current degree of the vertex in G'. In each iteration, we remove the smallest-degree vertex

v from H. If $\deg_{G'}(v) < \Delta$, then we delete all edges incident to v from E', and we update the degrees of its neighbors, reinserting them into H. We can charge the time required to update the neighbors of v and to reinsert them into H to the edges incident to v that were delted from E', so that the total running time of the algorithm is $\tilde{O}(m)$.

In our second step, we apply the algorithm from Claim 5.2 to graph G', with parameter $\epsilon = 1/k$, and parameter d remaining unchaged. Recall that the running time of the algorithm is $O(m^{1+\epsilon}) = O(m^{1+1/k})$. If the algorithm correctly certified that G' is $(d, 1/k)$-scattered, then we terminate the algorithm, and return the current set E' of edges. Therefore, we assume from now on that the algorithm from Claim 5.2 returned a vertex $v \in V(G')$ with $|B_{G'}(v, 4d/\epsilon)| \geq n^{1-1/k}$.

Let $T' = B_{G'}(v, 4d/\epsilon)$. Clearly, every vertex $x \in T'$ has $\deg_{G'}(x) \geq \Delta$, and moreover, for every pair $x, y \in T'$ of vertices, $\mathsf{dist}_{G'}(x, y) \leq 8d/\epsilon = d'$. Recall that $N = \left\lfloor n^{1/k - 1/k^2} \right\rfloor$, and so $N^k \leq n^{1-1/k} \leq |T'|$. We let $T \subseteq T'$ be an arbitrary subset of N^k vertices, that we refer to as *terminals* for the current phase.

Next, we set up an instance of the EmbedOrSeparate problem on graph G', the set T of its vertices that we defined before, parameter d replaced by d', and the remaining parameters N, k, η and δ unchanged. We first verify that this is indeed a valid instance of the problem.

From the definition of the EmbedOrScatter problem, $\Delta \geq 16$ and $k, \eta \geq 1$ hold. Additionally, $d' \geq d \geq 1$ must hold. From our construction, $|T| = N^k$, that every vertex in T has degree at least Δ in G. Since $d' = 8dk$, $N \leq n^{1/k}$, and, from the definition of the EmbedOrScatter problem, $\eta = n^{4/k} \cdot d \cdot k^{ck} \cdot 2^{c/\delta^6}$ for a large enough constant c, we get that $\eta > N^4 \cdot k \cdot d' \cdot 2^{c/\delta^6}$ holds. Since $N = \left\lfloor n^{1/k - 1/k^2} \right\rfloor$, while $4 \leq k < (\log n)^{1/3}$ and n is sufficiently large, we get that $N \geq 1$. From the definition of the EmbedOrScatter problem, $\frac{2}{(\log n)^{1/24}} < \delta < \frac{1}{400}$, and $1/\delta$ is an integer. Since $|T| = N^k \geq \frac{n^{1-1/k}}{2^k} \geq \sqrt{n}$, we get that $\log(|T|) \geq \frac{\log n}{2}$, and so $\frac{2}{(\log(|T|))^{1/12}} < \delta < \frac{1}{400}$ holds.

Finally, we need to show that $(|T| \cdot \Delta)^{\delta^5/2} \geq 2\log(|T| \cdot \Delta)$ holds. Recall that that $|T| = N^k = \left\lfloor n^{1/k - 1/k^2} \right\rfloor^k$. Therefore, $\frac{n^{1-1/k}}{2^k} \leq |T| \leq n^{1-1/k}$, and so:

$$\begin{aligned}
(|T| \cdot \Delta)^{\delta^5/2} &\geq \left(\frac{n^{1-1/k} \cdot \Delta}{2^k} \right)^{\delta^5/2} \\
&\geq n^{\delta^5/4} \\
&\geq n^{1/(\log n)^{1/4}} \\
&= 2^{(\log n)^{3/4}} \\
&\geq 8 \log n \\
&\geq 4 \log(n \cdot \Delta) \\
&\geq 2 \log(|T| \cdot \Delta).
\end{aligned}$$

For the second inequality, we have used the fact that $k \leq (\log n)^{1/3}$, so $2^k \leq n^{1/8}$. For the third inequality, we have used the fact that $\delta \geq \frac{2}{(\log n)^{1/24}}$. For the following inequality, we used the fact that $\sqrt{\log n} > 100 \log\log n$, and for the penultimate inequality we used the fact that $\Delta \leq n$.

We conclude that we obtain a valid instance $(G', T, \Delta, N, k, d', \eta, \delta)$ of the EmbedOrSeparate problem. Next, we apply the algorithm from Theorem 5.6 to this problem instance. Recall that the running time of the algorithm is $O\left(m^{1+O(\delta)} \cdot \eta^2\right)$. We now consider two cases. The first case happens if the

algorithm from Theorem 5.6 returned a collection $F \subseteq E(W_k^{N,\Delta})$ of at most $\frac{N^k \cdot \Delta}{(512k)^{4k}}$ edges of $W_k^{N,\Delta}$, together with an embedding of $W_k^{N,\Delta} \setminus F$ into G' via paths of length at most $d' \cdot 2^{O(1/\delta^6)} \leq d \cdot k \cdot 2^{O(1/\delta^6)}$ that cause congestion at most η. In this case we terminate the algorithm, and return the set F of fake edges, and the resulting embedding of $W_k^{N,\Delta} \setminus F$ into G.

Assume now that the second case happens, that is, the algorithm from Theorem 5.6 returned a subset $T^{\mathrm{del}} \subseteq T$ of at least $\frac{N^{k-1}}{k \cdot (512k)^{4k}}$ terminals, and a subset $E_i^{\mathrm{del}} \subseteq E(G')$ of at most $N^{k+1} \cdot \frac{k \cdot \Delta \cdot d' \cdot 2^{O(1/\delta^6)}}{\eta}$ edges, such that, for all $t \in T^{\mathrm{del}}$, $B_{G' \setminus E^{\mathrm{del}}}(t, d')$ contains at most $\frac{N^k}{2}$ vertices of T. In the latter case, for the sake of analysis, we define a set Π_i of pairs of terminals $(t, t') \in T$, such that $\mathsf{dist}_{G' \setminus E^{\mathrm{del}}}(t, t') > d'$. Clearly, $|\Pi_i| \geq |T^{\mathrm{del}}| \cdot \frac{N^k}{2} \geq \frac{N^{2k-1}}{2k \cdot (512k)^{4k}}$. Since $N = \left\lfloor n^{1/k - 1/k^2} \right\rfloor$, we get that $N^k \geq \frac{n^{1-1/k}}{2^k}$, and so $|\Pi_i| \geq \frac{n^{2-3/k}}{(512k)^{6k}}$. We add the edges of E^{del} to E' and delete them from G'. We say that the edges of E^{del} are added to E' *indirectly*. We then continue to the following phase.

Notice that, for every pair $(t, t') \in \Pi_i$ of vertices, $\mathsf{dist}_{G'}(t, t') > d'$ holds from now on. Therefore, in subsequent phases, we may never include both t and t' in the same set T of terminals, since every pair of terminals in T must be at distance at most d' from each other in G'. It then follows that all sets $\Pi_1, \Pi_2, \ldots$ of pairs of vertices are disjoint, and so the number of phases in the algorithm is bounded by $n^{3/k} \cdot (512k)^{6k}$. The number of edges added to set E' in every phase is bounded by: $N^{k+1} \cdot \frac{k^2 \cdot \Delta \cdot d \cdot 2^{O(1/\delta^6)}}{\eta} \leq \frac{n \cdot k^2 \cdot \Delta \cdot d \cdot 2^{O(1/\delta^6)}}{\eta}$. Therefore, at the end of the algorithm, the number of edges that are added indirectly to E' is bounded by:

$$\begin{aligned}
|E'| &\leq \frac{n \cdot k^2 \cdot \Delta \cdot d \cdot 2^{O(1/\delta^6)}}{\eta} \cdot n^{3/k} \cdot (512k)^{6k} \\
&\leq \frac{n^{1+3/k} \cdot \Delta \cdot d \cdot (512k)^{7k} \cdot 2^{O(1/\delta^6)}}{\eta} \\
&< n \cdot \Delta,
\end{aligned}$$

since $\eta = n^{4/k} \cdot d \cdot k^{ck} \cdot 2^{c/\delta^6}$ for a large enough constant c.

Recall that the preprocessing step takes time $\tilde{O}(m)$, and the running time of the algorithm from Claim 5.2 is $O(m^{1+1/k})$. The running time of the algorithm from Theorem 5.6 is:

$$O\left(m^{1+O(\delta)} \cdot \eta^2\right) \leq O\left(m^{1+O(\delta+1/k)} \cdot d^2 \cdot k^{O(k)} \cdot 2^{O(1/\delta^6)}\right) \leq O\left(m^{1+O(\delta+1/k)} \cdot d^2 \cdot 2^{O(1/\delta^6)}\right).$$

since $\eta \leq n^{4/k} \cdot d \cdot k^{O(k)} \cdot 2^{O(1/\delta^6)}$. Additionally, we have used the fact that $k < (\log n)^{1/3}$, so $k^k \leq n^{1/k}$. We conclude that the running time of a single phase is bounded by $O\left(m^{1+O(\delta+1/k)} \cdot d^2 \cdot 2^{O(1/\delta^6)}\right)$. Since, as observed before, the number of phases is bounded by $n^{3/k} \cdot k^{O(k)} \leq n^{O(1/k)}$, we get that the total running time of the algorithm is bounded by $O\left(m^{1+O(\delta+1/k)} \cdot d^2 \cdot 2^{O(1/\delta^6)}\right)$.

5.3 Algorithm for EmbedOrSeparate– Proof of Theorem 5.6

In this subsection, we provide an algorithm for the EmbedOrSeparate problem, proving Theorem 5.6. Throughout the proof, we denote $m = |E(G)|$ and $n = |V(G)|$. For convenience, we will denote $W_k^{N,\Delta}$ by W_k in the remainder of this subsection. We start by mapping every vertex of W_k to a distinct

vertex of T. For convenience, we will not distinguish between vertices of W_k and the terminals to which they are mapped. We initialize $F=\emptyset$, and we initialize the embedding $\mathcal{Q}$ of $E(W_k)\setminus F$ into G by setting $\mathcal{Q}=\emptyset$. Over the course of the algorithm, we will gradually add paths to $\mathcal{Q}$, where every path $Q(\hat{e})$ added to $\mathcal{Q}$ is an embedding of a distinct edge $\hat{e}\in E(W_k)$, and has length at most $d\cdot 2^{O(1/\delta^6)}$. We will ensure that every edge of G participates in at most η paths in $\mathcal{Q}$. To this end, for every edge $e\in E(G)$, we maintain a counter n_e, that counts the number of paths in $\mathcal{Q}$ that use the edge e. Once the counter n_e reaches η, we will ensure that the edge will never participate in any subsequent embedding paths. We initialize all such counters n_e to 0.

Throughout the algorithm, we maintain a set E^{del} of edges that have been deleted from G. We also maintain a set T^{del} of terminals, such that, for every terminal $t\in T^{\mathrm{del}}$, $B_{G\setminus E^{\mathrm{del}}}(t,d)$ contains at most $\frac{N^k}{2}$ vertices of T. We will gradually add vertices to T^{del}, and, whenever $|T^{\mathrm{del}}|$ reaches $\frac{N^{k-1}}{k\cdot(512k)^{4k}}$, the algorithm terminates with the output $(E^{\mathrm{del}},T^{\mathrm{del}})$. Therefore, throughout the algorithm, $|T^{\mathrm{del}}|\leq\frac{N^{k-1}}{k\cdot(512k)^{4k}}$ holds (except possibly when the algorithm terminates).

The set E^{del} of edges that the algorithm maintains is partitioned into two subsets, that we denote by E_1^{del} and E_2^{del}, respectively. An edge $e\in E(G)$ may be added to E_1^{del} only if it participates in at least $\eta/2$ paths in the current set $\mathcal{Q}$. Since $|E(W_k)|\leq N^k\cdot k\cdot\Delta$, and since we ensure that the length of every embedding path in $\mathcal{Q}$ is bounded by $d\cdot 2^{O(1/\delta^6)}$, we get that, throughout the algorithm, the cardinality of E_1^{del} is bounded as follows:

$$|E_1^{\mathrm{del}}|\leq\frac{N^k\cdot k\cdot\Delta\cdot d\cdot 2^{O(1/\delta^6)}}{\eta} \tag{3}$$

Over the course of the algorithm, we will add paths to $\mathcal{Q}$ one by one, and we will never delete paths from $\mathcal{Q}$, so that the set E_1^{del} of edges is incremental.

The second set, E_2^{del} of edges, may contain some additional edges, and we ensure that it is also incremental. We will ensure that, throughout the algorithm:

$$|E_2^{\mathrm{del}}|\leq|T^{\mathrm{del}}|\cdot N\cdot\Delta\cdot\frac{32d}{\eta} \tag{4}$$

holds. Since $|T^{\mathrm{del}}|\leq N^{k-1}$ over the course of the algorithm, we get that, throughout the algorithm:

$$|E^{\mathrm{del}}|\leq|E_1^{\mathrm{del}}|+|E_2^{\mathrm{del}}|\leq N^k\cdot\frac{k\cdot\Delta\cdot d\cdot 2^{O(1/\delta^6)}}{\eta}+N^k\cdot\Delta\cdot\frac{32d}{\eta}\leq N^k\cdot\frac{k\cdot\Delta\cdot d\cdot 2^{O(1/\delta^6)}}{\eta}. \tag{5}$$

Since, at the end of the algorithm, $|T^{\mathrm{del}}|\leq N^k$, we get that, at the end of the algorithm:

$$|E^{\mathrm{del}}|\leq|E_1^{\mathrm{del}}|+|E_2^{\mathrm{del}}|\leq N^k\cdot\frac{k\cdot\Delta\cdot d\cdot 2^{O(1/\delta^6)}}{\eta}+N^{k+1}\cdot\Delta\cdot\frac{32d}{\eta}\leq N^{k+1}\cdot\frac{k\cdot\Delta\cdot d\cdot 2^{O(1/\delta^6)}}{\eta}, \tag{6}$$

as required. The algorithm also maintains a set $F\subseteq E(W_k)$ of fake edges. At the beginning of the algorithm, we initialize $F=\emptyset$, and then, over the course of the algorithm, we may add edges to F. However, an edge of W_k may only be added to F if at least one of its endpoints currently lies in T^{del}. Since $|T^{\mathrm{del}}|\leq\frac{N^{k-1}}{k\cdot(512k)^{4k}}$ holds throughout the algorithm, and since the degree of every vertex in W_k is at most $N\cdot k\cdot\Delta$, this guarantees that $|F|\leq\frac{N^{k-1}}{k\cdot(512k)^{4k}}\cdot N\cdot k\cdot\Delta\leq\frac{N^k\cdot\Delta}{(512k)^{4k}}$ always holds, as required (if, at the end of the algorithm, $|T^{\mathrm{del}}|\geq\frac{N^{k-1}}{k\cdot(512k)^{4k}}$, then the algorithm's output is $(T^{\mathrm{del}},E^{\mathrm{del}})$, so the cardinality of the set F is unimportant in that case).

We use a parameter $z = (2m)^{32\delta} \cdot k \cdot d \cdot 2^{c/\delta^6}$, where c is a large enough constant, for example that used in the condition on η in the definition of the problem. The algorithm consists of a number of phases. We now provide a high-level description of a single phase.

High-Level Description of a Single Phase

At the beginning of the ith phase, we select an arbitrary subset $\hat{E}_i \subseteq E(W_k) \setminus F$ of $\left\lfloor \frac{|E(W^k)|}{z} \right\rfloor$ edges that were not embedded yet into G, and whose endpoints do not currently lie in T^{del} (if $E(W_k) \setminus F$ contains fewer such edges, then we let $\hat{E}_i$ denote the set of all such remaining edges). Then, over the course of the ith phase, we will embed a subset $\hat{E}'_i \subseteq \hat{E}_i$ of the edges into G. We denote by $A'_i \subseteq E(G)$ the set of edges $e \in E(G)$, such that, prior to the ith phase, e participated in fewer than $\eta/2$ embedding paths, but after the ith phase is completed, e participates in at least $\eta/2$ embedding paths (we will ensure that no edge participates in more than η of the embedding paths). We will also compute another subset $A''_i \subseteq E(G)$ of edges, and we will identify a subset $T_i^{\text{del}} \subseteq T \setminus T^{\text{del}}$ of terminals, such that, for all $t \in T_i^{\text{del}}$, $B_{G \setminus (E^{\text{del}} \cup A'_i \cup A''_i)}(t, d)$ contains at most $N^k/2$ vertices of T. We will ensure that the following hold:

- P1. either $|T_i^{\text{del}}| \geq \frac{N^{k-1}}{m^{O(\delta^3)}}$ and $|A''_i| \leq |T_i^{\text{del}}| \cdot N \cdot \Delta \cdot \frac{32d}{\eta}$, in which case we say that the ith phase is *bad*; or
- P2. $A''_i = \emptyset$, and every edge in $\hat{E}_i \setminus \hat{E}'_i$ has an endpoint in T_i^{del}, in which case we say that the ith phase is *good*.

In either case, we add the edges of A'_i to E_1^{del}, and the edges of A''_i to E_2^{del}. We then update $E^{\text{del}} = E_1^{\text{del}} \cup E_2^{\text{del}}$, and add the vertices of T_i^{del} to T^{del}. Notice that if, before the current phase, Inequality 4 held, then it continues to hold at the end of the current phase. Lastly, we add the embeddings of the edges in $\hat{E}'_i$ that our algorithm computed to $\mathcal{Q}$. If the ith phase is good, then the edges of $\hat{E}_i \setminus \hat{E}'_i$ are added to the set F of fake edges. Additionally, every edge of W_k with at least one endpoint in T^{del} is added to F. If $|T^{\text{del}}| \geq \frac{N^k}{k \cdot (512k)^{4k}}$, then we terminate the algorithm and return $(T^{\text{del}}, E^{\text{del}})$. If all edges of $W_k \setminus F$ are currently embedded, then we terminate the algorithm, and return the current set $F \subseteq E(W^k)$, and the current embedding $\mathcal{Q}$ of $W^k \setminus F$, that is guaranteed to have all required properties. Otherwise, we continue to the next iteration.

This completes the high-level description of a single phase. Before we provide an algorithm to implement a single phase, we bound the number of phases in the following simple observation.

Observation 5.7 *The number of phases in the algorithm is bounded by* $O\left(m^{O(\delta)} \cdot k \cdot d \cdot 2^{O(1/\delta^6)}\right)$.

Proof: Since every bad phase adds at least $\frac{N^{k-1}}{m^{O(\delta^3)}}$ new terminals to set T^{del}, the number of bad phases is bounded by $m^{O(\delta^3)}$. Notice that, if the ith phase is good, then, at the end of the phase, every edge of $\hat{E}_i$ is either embedded into G, or added to F. Since $|\hat{E}_i| = \left\lfloor \frac{|E(W^k)|}{z} \right\rfloor$ (except when i is the last phase), there are at most $2z \leq m^{O(\delta)} \cdot k \cdot d \cdot 2^{O(1/\delta^6)}$ good phases.

Therefore, the total number of phases is bounded by $O\left(m^{O(\delta)} \cdot k \cdot d \cdot 2^{O(1/\delta^6)}\right)$. □

We now turn to describe an algorithm for implementing the ith phase, for $i \geq 1$. We assume that we are given a set $\hat{E}_i \subseteq E(W_k) \setminus F$ of at most $\left\lfloor \frac{|E(W^k)|}{z} \right\rfloor$ edges that are not embedded yet into G, and whose endpoints do not currently lie in T^{del}. For convenience, we will denote $\hat{E}_i$ and $\hat{E}'_i$ by

$\hat{E}$ and $\hat{E}'$, respectively. The algorithm consists of two steps. In the first step, we compute a large well-connected graph X, and we attempt to embed it into G, using Theorem 2.7. If we fail to do so, then we will compute a subset $A_i'' \subseteq E(G)$ of edges, and a subset $T_i^{\mathrm{del}} \subseteq T \setminus T^{\mathrm{del}}$ of terminals, such that, for all $t \in T_i^{\mathrm{del}}$, $B_{G \setminus (E^{\mathrm{del}} \cup A_i'')}(t, d)$ contains at most $|T|/2$ terminals. In this case, we terminate the algorithm for the current phase with $\hat{E}' = \emptyset$ and $A_i' = \emptyset$. If we manage to successfully compute a well-connected graph X and embed it into G, then we continue to the second step, in which we will employ the algorithm for APSP in well-connected graphs from Theorem 2.8, together with the standard Even-Shiloach Tree data structure, in order to compute an embedding of a large subset of edges of $\hat{E}$. If we fail to do so, then we will again compute the set T_i^{del} of terminals and a set A_i'' of edges with the required properties. We now formally describe each of the two steps in turn.

5.3.1 Step 1: Embedidng a Well-Connected Graph into G

We assume that E^{del} contains every edge e that participates in at least $\eta/2$ paths of $\mathcal{Q}$ at the beginning of the phase. Let $G' = G \setminus E^{\mathrm{del}}$, and we let $T' \subseteq T \setminus T^{\mathrm{del}}$ contain all terminals $t \in T \setminus T^{\mathrm{del}}$, such that the degree of t in G' is at least $\Delta/2$. Recall that from Inequality 5, and since we have assumed that $\eta > N^4 \cdot k \cdot d \cdot 2^{c/\delta^6}$ for a large enough constant c, we get that:

$$|E^{\mathrm{del}}| \leq \frac{N^k \cdot k \cdot \Delta \cdot d \cdot 2^{O(1/\delta^6)}}{\eta} \leq \frac{N^k \cdot \Delta}{16}.$$

Since the algorithm did not terminate yet, $|T^{\mathrm{del}}| \leq \frac{N^{k-1}}{k \cdot (512k)^{4k}}$, and so $|T \setminus T^{\mathrm{del}}| \geq \frac{N^k}{2}$ holds. Since the degree of every terminal $t \in T$ in G is at least Δ, we then get that $|T'| \geq \frac{N^k}{4}$ must hold.

We start by slightly modifying the graph G', as follows. Consider an edge $e = (u, v) \in E(G')$. If both endpoints of e lie in T', then we replace e with a path (u, u', v', v), where u' and v' are new vertices that we call *copies* of u and v, respectively. We also refer to the edges $(u, u'), (u', v')$ and (v', v) as *artificial edges corresponding to edge* (u, v). If exactly one endpoint of e (say u) lies in T', then we replace e with a path (u, u', v), and we call u' a copy of u, and we call all edges on the new path artificial edges corresponding to (u, v). Otherwise, we do not perform any transformation of e.

Let G'' denote the resulting graph, and let $\tilde{T} \subseteq V(G'')$ be the set of vertices obtained by adding, for every terminal $t \in T'$, exactly $\lceil \Delta/2 \rceil$ copies of t into $\tilde{T}$. We refer to the vertices in $\tilde{T}$ as *pseudo-terminals*, and we denote $q = |\tilde{T}|$. Notice that $q = |T'| \cdot \left\lceil \frac{\Delta}{2} \right\rceil \geq \frac{N^k \cdot \Delta}{8}$. Observe also that $|E(G'')| \leq O(m)$.

Let $\hat{N} = \left\lfloor q^{\delta} \right\rfloor$, so $\frac{q^{\delta}}{2} \leq \hat{N} \leq q^{\delta}$. Notice that:

$$\hat{N}^{1/\delta} \geq \frac{q}{2^{1/\delta}} \geq \frac{|T| \cdot \Delta}{8 \cdot 2^{1/\delta}}.$$

From the problem definition, $\delta > \frac{2}{(\log(|T|))^{1/12}} > \frac{2}{(\log q)^{1/12}}$, and so $\frac{2}{(\log q)^{1/12}} < \delta < \frac{1}{400}$ holds.

We apply the algorithm from Theorem 2.7 to graph G'', the set $\tilde{T}$ of its vertices, with parameter d replaced by $\hat{d} = 16d$, parameter η remaining unchanged, and parameter ϵ replaced by δ.

We now consider two cases. In the first case, the algorithm returns a graph X with $V(X) \subseteq \tilde{T}$, and $|V(X)| = \hat{N}^{1/\delta} \geq \frac{|T| \cdot \Delta}{8 \cdot 2^{1/\delta}}$, whose maximum degree is at most $q^{32\delta^3}$, together with an embedding $\mathcal{P}$ of X into G'' via paths of length at most $\hat{d} = 16d$, that cause congestion at most $\eta \cdot q^{32\delta^3} \leq \eta \cdot \left(N^k \cdot \Delta\right)^{32\delta^3} \leq 2\eta \cdot m^{32\delta^3}$, and a level-$(1/\delta)$ Hierarchical Support Structure for X, such that X is $(\tilde{d}, \eta')$-well-connected

with respect to the set $S(X)$ of vertices defined by the support structure, where $\eta' = \hat{N}^{6+256\delta}$, and $\tilde{d} = 2^{\tilde{c}/\delta^5}$, and $\tilde{c}$ is the constant used in the definition of the Hierarchical Support Structure. In this case, we say that the first step has successfully embedded a well-connected graph into G', and continue to the second step.

In the second case, the algorithm from Theorem 2.7 returns a pair $\tilde{T}_1, \tilde{T}_2 \subseteq \tilde{T}$ of disjoint subsets of pseudo-terminals of equal cardinality, and a set E' of edges of G'' of cardinality at most:

$$|E'| \leq |\tilde{T}_1| \cdot \frac{16d}{\eta} \leq \frac{|\tilde{T}_1|}{2},$$

since, from the problem definition, $\eta > N^4 \cdot k \cdot d \cdot 2^{c/\delta^6}$ holds, for a large enough constant c.

The algorithm also ensures that for every pair $t \in \tilde{T}_1, t' \in \tilde{T}_2$ of pseudo-terminals, $\mathsf{dist}_{G'' \setminus E'}(t, t') > \hat{d} = 16d$. Finally, the algorithm ensures that:

$$|\tilde{T}_1| \geq \frac{q^{1-4\delta^3}}{4} \geq \frac{(|T'| \cdot \Delta/2)^{1-4\delta^3}}{4} \geq \frac{N^k \cdot \Delta}{32 \cdot (N^k \cdot \Delta)^{4\delta^3}} \geq \frac{N^k \cdot \Delta}{64 \cdot m^{4\delta^3}}.$$

We construct a set E'' of edges in G as follows: for every edge $e \in E'$, if $e \in E(G)$, then we add e to E'. Otherwise, e is an artificial edge, corresponding to some edge $e' \in E(G)$. We then add e' to E''. It is easy to verify that $|E''| \leq |E'|$.

Next, we construct two sets T_1, T_2 of terminals, as follows. First, we let $\tilde{T}'_1 \subseteq \tilde{T}_1$ be a set of pseudo-terminals obtained from $\tilde{T}_1$ by deleting all pseudo-terminals that are incident to the edges of E'. We similarly define a set $\tilde{T}'_2 \subseteq \tilde{T}_2$ of pseudo-terminals. Since $|E'| < \frac{|\tilde{T}_1|}{2}$ and $|\tilde{T}_1| = |\tilde{T}_2|$, we get that $|\tilde{T}'_1|, |\tilde{T}'_2| \geq \frac{|\tilde{T}_1|}{2}$. For every pseudo-terminal $\tilde{t} \in \tilde{T}'_1$, we add the corresponding terminal t to T_1. Similarly, for every pseudo-terminal $\tilde{t}' \in \tilde{T}'_2$, we add the corresponding terminal t' to T_2. Clearly:

$$|T_1| \geq \frac{|\tilde{T}'_1|}{\Delta} \geq \frac{|\tilde{T}_1|}{2\Delta} \geq \frac{N^k}{128m^{4\delta^3}}.$$

Using the same reasoning, $|T_2| \geq \frac{N^k}{128m^{4\delta^3}}$. Assume w.l.o.g. that $|T_2| \geq |T_1|$. We delete vertices from T_2, until the cardinalities of both sets become equal. We also get that:

$$|E''| \leq |E'| \leq |\tilde{T}_1| \cdot \frac{16d}{\eta} \leq |\tilde{T}'| \cdot \frac{32d}{\eta} \leq |T_1| \cdot \Delta \cdot \frac{32d}{\eta}.$$

We need the following simple observation.

Observation 5.8 $\mathsf{dist}_{G' \setminus E''}(T_1, T_2) > 2d$.

Proof: Assume for contradiction that the claim is false. Then there are terminals $t \in T_1$, $t' \in T_2$, and a path P in $G' \setminus E''$, connecting t to t', such that the length of P is at most d.

Let P' be the path corresponding to P in graph G''. Since graph G'' is obtained from G' by subdividing each of its edges by at most 2 new vertices, the length of P in G'' is at most $8d$, and P connects t to t' in G''. We claim that P' must avoid the edges of E'. Indeed, consider any edge $e \in E'$, and assume for contradiction that P' contains e. If $e \in E(G')$, then e is an edge of E'' that lies on P' and hence on P. Since P is disjoint from E'', this is impossible. Therefore, e is an artificial edge that corresponds

to some edge $e' \in E(G')$. But then $e' \in E''$, and e' does not lie on P. It is then impossible that e' lies on P'. We conclude that P' is a path of length at most $8d$ in $G' \setminus E'$, connecting t to t'.

Lastly, let $\tilde{t}$ be a pseudo-terminal in $\tilde{T}_1'$ that corresponds to t, and let $\tilde{t}'$ be a pseudo-terminal in $\tilde{T}_2'$ that corresponds to t'. From the definition of the sets $\tilde{T}_1', \tilde{T}_2'$ of pseudo-terminals, edges $(\tilde{t}, t)$ and $(t', \tilde{t}')$ are disjoint from set E'. Therefore, if we let P'' be the path obtained from P' by appending $(\tilde{t}, t)$ at the beginning and $(t', \tilde{t}')$ at the end of the path, then P'' is a path of length $8d + 2 \leq 10d < \hat{d}$ in $G'' \setminus E'$, connecting a pair $\tilde{t} \in \tilde{T}_1$, $\tilde{t}' \in \tilde{T}_2$ of terminals, a contradiction. □

Let $S \subseteq T$ be the set of all terminals lying in $B_{G' \setminus E''}(T_1, d)$, and let $S' \subseteq T$ be the set of all terminals lying in $B_{G' \setminus E''}(T_2, d)$. Since $\mathsf{dist}_{G' \setminus E''}(T_1, T_2) > 2d$, it must be the case that either $|S| \leq \frac{N^k}{2}$, or $|S'| \leq \frac{N^k}{2}$. Assume w.l.o.g. that it is the former. Then we let $T_i^{\mathrm{del}} = T_1$, and $A_i'' = E''$. We also let $\hat{E}_i' = \emptyset$, and $A_i' = \emptyset$. We return $T_i^{\mathrm{del}}, A_i', A_i''$ and $\hat{E}_i'$ as the outcome of the phase, and terminate the phase as a bad phase. Since $|S| \leq \frac{N^k}{2}$, it is easy to verify that, for every vertex $t \in T_i^{\mathrm{del}}$, $B_{G \setminus (E^{\mathrm{del}} \cup A_i'')}(t, d) = B_{G' \setminus A_i''}(t, d)$ contains at most $N^k/2$ vertices of T. As we have shown, $|A_i''| = |E''| \leq |T_1| \cdot \Delta \cdot \frac{32d}{\eta} = |T_i^{\mathrm{del}}| \cdot \Delta \cdot \frac{32d}{\eta}$, and $|T_i^{\mathrm{del}}| = |T_1| \geq \frac{N^k}{m^{O(\delta^3)}}$.

We now bound the running time of the first step. The running time of the algorithm from Theorem 2.7 is bounded by:

$$\begin{aligned} O\left(q^{1+O(\delta)} + |E(G)| \cdot q^{O(\delta^3)} \cdot (\eta + d \log n)\right) &\leq O\left((N^k \cdot \Delta)^{1+O(\delta)} + |E(G)| \cdot (N^k \cdot \Delta)^{O(\delta^3)} \cdot (\eta + d \log n)\right) \\ &\leq O\left(m^{1+O(\delta)} \cdot \eta\right). \end{aligned}$$

The remaining time required to compute the set E'' of edges and the sets T_1, T_2 of terminals, together with the time required to compute the graph G'' and the sets S and S' of terminals, is asymptotically bounded by the above running time. Therefore, the total running time of the first step of the algorithm is at most: $O\left(m^{1+O(\delta)} \cdot \eta\right)$.

5.3.2 Step 2: Computing the Embedding of the Edges of $\hat{E}$

We assume from now on that the algorithm from Step 1 computed a graph X with $V(X) \subseteq \tilde{T}$, and $|V(X)| = \hat{N}^{1/\delta} \geq \frac{|T| \cdot \Delta}{8 \cdot 2^{1/\delta}}$, whose maximum degree is at most $q^{32\delta^3}$, together with an embedding $\mathcal{P}$ of X into G' via paths of length at most $16d$ that cause congestion at most $2\eta \cdot m^{32\delta^3}$, and a level-$(1/\delta)$ Hierarchical Support Structure for X, such that X is $(\tilde{d}, \eta')$-well-connected with respect to the set $S(X)$ of vertices defined by the support structure, where $\eta' = \hat{N}^{6+256\delta}$, and $\tilde{d} = 2^{\tilde{c}/\delta^5}$, and $\tilde{c}$ is the constant used in the definition of the Hierarchical Support Structure. Our algorithm maintains two main data structures.

First Data Structure: APSP in X. For our fist data structure, we use the algorithm for APSP in well-connected graphs from Theorem 2.8, that is applied to graph X, with parameter N replaced by $\hat{N}$ and parameted ϵ replaced by δ. We exploit the level-$(1/\delta)$ Hierarchical Support Structure for X that was computed in Step 1 of the algorithm. Recall that we are guaranteed that X is $(\tilde{d}, \eta')$-well-connected with respect to the set $S(X)$ of vertices defined by the Hierarchical Support Structure. In order to be able to use the algorithm from Theorem 2.8, we need to verify that $\frac{\hat{N}^{\delta^4}}{\log \hat{N}} \geq 2^{128/\delta^6}$ holds. Indeed, recall that $\frac{q^\delta}{2} \leq \hat{N} \leq q^\delta$, and that $q = |T'| \cdot \lceil \Delta/2 \rceil \geq \frac{|T| \cdot \Delta}{8}$. Therefore:

$$\frac{\hat{N}^{\delta^4}}{\log \hat{N}} \geq \frac{(|T|\cdot\Delta)^{\delta^5}}{8\delta\log(|T|\cdot\Delta)} \geq (|T|\cdot\Delta)^{\delta^5/2} \geq 2^{128/\delta^6}, \tag{7}$$

since $(|T|\cdot\Delta)^{\delta^5/2} \geq 2\log(|T|\cdot\Delta)$ from the problem definition. We have also used the fact that $\frac{2}{\delta} \leq (\log(|T|))^{1/12}$, so, in particular, $\log|T| \geq \frac{2^{12}}{\delta^{12}}$, and $|T| \geq 2^{2^{12}/\delta^{12}}$.

Let $\Lambda = |V(X)|^{1-10\delta}$, and recall that $|V(X)| \geq \frac{|T|\cdot\Delta}{8\cdot 2^{1/\delta}}$, so that:

$$\Lambda \geq \frac{(|T|\cdot\Delta)^{1-10\delta}}{8\cdot 2^{1/\delta}} \geq (|T|\cdot\Delta)^{1-11\delta}, \tag{8}$$

since $8\cdot 2^{1/\delta} \leq 2^{256/\delta^{10}} \leq (|T|\cdot\Delta)^{\delta}$, where the last inequality follows from Equation 7.

Recall that the algorithm from Theorem 2.8 maintains a set $S'(X)$ of vertices of X, as graph X undergoes a sequence of at most Λ edge deletions, such that set $S'(X)$ is decremental: vertices may leave it but not join it. Moreover, the algorithm ensures that, at all times:

$$|S'(X)| \geq \frac{|V(X)|}{2^{4/\delta}} \geq \frac{|T|\cdot|\Delta|}{2^{6/\delta}} \tag{9}$$

holds (we have used the fact that $|V(X)| \geq \frac{|T|\cdot\Delta}{8\cdot 2^{1/\delta}}$). We let $S''(X) \subseteq T$ be a set that contains every terminal $t \in T$, such that at least one pseudo-terminal representing t lies in $S'(X)$. It is easy to verify that $S''(X)$ is a decremental set, and, since every terminal has at most Δ corresponding pseudo-terminals, throughout the algorithm:

$$|S''(X)| \geq \frac{|T|}{2^{6/\delta}} \tag{10}$$

holds. We may sometimes refer to the vertices of $S''(X)$ as the *core*. The algorithm from Theorem 2.8 supports short-path queries between vertices of $S'(X)$: given a pair $x, y \in S'(X)$ of vertices, it returns a path P connecting x to y in the current graph X, whose length is at most $2^{O(1/\delta^6)}$, in time $O(|E(P)|)$. We denote the data structure maintain by the algorithm from Theorem 2.8 by DS_1. The total update time of the algorithm from Theorem 2.8 is bounded by:

$$O(|E(X)|^{1+O(\delta)}) \leq O\left(\left(|E(G)|^{1+O(\delta^3)}\right)^{1+O(\delta)}\right) \leq O\left(|E(G)|^{1+O(\delta)}\right),$$

since maximum degree in X is at most $q^{32\delta^3} \leq (|T|\cdot\Delta)^{O(\delta^3)} \leq |E(G)|^{O(\delta^3)}$, and $|V(X)| \leq |T|\cdot\Delta \leq |E(G)|$.

Second Data Structure: ES-Tree. For our second data structure, we construct a dynamic graph H, as follows. We start with the graph G', and we add a source vertex s to it, that connects with an edge to every vertex in set $S''(X)$ – the core set maintained by DS_1. As the algorithm progresses, whenever a vertex v is deleted from set $S''(X)$, we delete the edge (s, v) from H. Additionally, we may delete some edges from G' and from H directly – these are edges that participate in too many of the embedding paths that we construct. Our second data structure, that we denote by DS_2, maintains an ES-Tree in graph H, rooted at the source vertex s, up to depth $2d+1$. Recall that the total update time of this data structure is $O(|E(H)|\cdot d) = O(|E(G)|\cdot d)$. The data structure also maintains a set U of vertices of G, that contains every vertex u with $\mathsf{dist}_H(s,u) > 2d+1$.

We are now ready to describe our algorithm for the second step. We start by initializing data structures DS_1 and DS_2. For every edge $e \in E(G')$, we initialize a list $L(e)$ of all edges $e' \in E(X)$, whose embedding path $P(e') \in \mathcal{P}$ contains either e or one of the artificial edges that subdivide e. The time that is required to initialize all these lists is asymptotically bounded by the running time of Step 1, that computed the embedding $\mathcal{P}$ of X into G''.

Recall that $G' = G \setminus E^{\mathrm{del}}$. For every edge $e \in E(G')$, we let n_e the number of paths in the current set $\mathcal{Q}$ in which e participates. We initialize $E_i^{\mathrm{del}} = \emptyset$. For every edge $e \in E(G')$, once the number of embedding paths in $\mathcal{Q}$ that use e reaches η, we add e to E_i^{del}. We also initialize the set F_i of new fake edges to $\emptyset$.

Next, we process the edges $\hat{e} \in \hat{E}$ one by one. Consider any such edge $\hat{e} = (u, v)$. If either of the vertices u or v lies in U, then we add e to the set F_i of the fake edges. Otherwise, we use the $\mathsf{ES\text{-}Tree}$ data structure to compute a path P_1 of length at most $2d$ connecting u to some vertex $u' \in S''(X)$ (by computing a path connecting u to s, whose penultimate vertex must lie in $S''(X)$), and another path P_2 of length at most $2d$ connecting v to some vertex $v' \in S''(X)$. Both paths can be computed in time $O(d)$. Recall that some pseudoterminal $\tilde{u}'$ correpsonding to u', and some pseudoterminal $\tilde{v}'$ correpsonding to v' must lie in $S'(X)$. Next, we use the algorithm from Theorem 2.8 to compute a path $\hat{P}$ in graph X connecting $\tilde{v}'$ to $\tilde{u}'$, such that the length of $\hat{P}$ is bounded by $2^{O(1/\delta^6)}$, in time $O(|E(\hat{P})|)$. We then use the embedding of X into G'' to transform path $\hat{P}$ into a path P_3', connecting $\tilde{u}'$ to $\tilde{v}'$ in G''. Since the embedding of X into G'' is via paths of length at most $O(d)$, the length of P_3' is bounded by $d \cdot 2^{O(1/\delta^6)}$. Since graph G'' is obtained from G' by subdividing some of its edges, it is immediate to convert P_3' into a path P_3 in G connecting v' to u', whose length is at most $d \cdot 2^{O(1/\delta^6)}$. By concatenating P_1, P_3, and the reversed path P_2, we obtain a path connecting u to v in G, of length at most $d \cdot 2^{O(1/\delta^6)}$. The time required to compute this path is $O\left(d \cdot 2^{O(1/\delta^6)}\right)$, and we can turn the resulting path into a simple one, in time $O\left(d \cdot 2^{O(1/\delta^6)}\right)$. Let $Q(\hat{e})$ denote the resulting u-v path in G, whose length is bounded by $d \cdot 2^{O(1/\delta^6)}$. We add $Q(\hat{e})$ to set $\mathcal{Q}$ as the embedding of the edge $\hat{e}$. For every edge $e \in Q(\hat{e})$, we increase the counter n_e, and, if it reaches η, we delete e from graph H, and add it to E_i^{del}. In the latter case, for every edge $e' \in L(e)$ (that is, every edge $e' \in E(X)$ whose embedding path $P(e')$ uses e or one of the artificial edges subdividing e), we delete e' from X, and we update data structure DS_1 with this deletion.

Before we provide the remainder of the algorithm, we bound the total number of edges that are deleted from H, and the total number of edges that are deleted from X over the course of the second step of the phase, in the following observation and its corollary.

Observation 5.9 *Let E' be the set of all edges that are deleted from H over the course of Step 2 of the current phase, excluding the edges incident to s. Then $|E'| \leq \frac{|T| \cdot \Delta}{8 \cdot (2m)^{32\delta} \cdot \eta}$.*

Proof: Recall that:

$$|\hat{E}| \leq \frac{|E(W^k)|}{z} \leq \frac{|T| \cdot \Delta}{(2m)^{32\delta} \cdot d \cdot 2^{c/\delta^6}}$$

since $|E(W_k)| = N^k \cdot k \cdot \Delta = |T| \cdot k \cdot \Delta$, and $z = (2m)^{32\delta} \cdot k \cdot d \cdot 2^{c/\delta^6}$, for a large enough constant c. The length of the embedding path $Q(\hat{e})$ of each edge $e \in \hat{E} \setminus F_i$ is bounded by $d \cdot 2^{O(1/\delta^6)}$. An edge e' lies in E' iff it participates in at least $\eta/2$ embedding paths that have been added to $\mathcal{Q}$ over the course of Step 2 of the current phase, since, at the beginning of the phase, for every edge $e' \in E(G')$, $n_{e'} \leq \eta/2$ held. Therefore:

$$|E'| \leq \frac{|T| \cdot \Delta}{(2m)^{32\delta} \cdot d \cdot 2^{c/\delta^6}} \cdot \frac{d \cdot 2^{O(1/\delta^6)}}{\eta} \leq \frac{|T| \cdot \Delta}{8 \cdot (2m)^{32\delta} \cdot \eta}.$$

□

Corollary 5.10 *The total number of edges ever deleted from X over the course of the algorithm is at most Λ.*

Proof: Let E^* be the set of all edges deleted from X over the course of the algorithm. From Equation 8, it is enough to show that $|E^*| \leq (|T| \cdot \Delta)^{1-11\delta}$. Recall that the embedding of X into G causes congestion at most $2\eta \cdot m^{32\delta^3}$ in G''. Since every edge of G' has at most 3 edges of G'' subdividing it, every edge deleted from H (except for those incident to s) over the course of the current step may lead to the deletion of up to $6\eta \cdot m^{32\delta^3}$ edges of X. Altogether:

$$\begin{aligned}
|E^*| &\leq 6\eta \cdot m^{32\delta^3} \cdot |E'| \\
&\leq 6m^{32\delta^3} \cdot \frac{|T| \cdot \Delta}{(2m)^{32\delta}} \\
&\leq \frac{6|T| \cdot \Delta}{(|T| \cdot \Delta)^{16\delta}} \\
&\leq (|T| \cdot \Delta)^{1-11\delta}.
\end{aligned}$$

□

We are now ready to complete the algorithm. Let $A'_i \subseteq E(G)$ be the set of edges $e \in E(G)$, such that, prior to the current phase, e participated in fewer than $\eta/2$ embedding paths, but at the end of the current phase, e participates in at least $\eta/2$ embedding paths. Notice that, in particular $E' \subseteq A'_i$. We also let $A''_i = \emptyset$, and $\hat{E}'_i \subseteq \hat{E}$ the set of edges $\hat{e} \in \hat{E}$ for which an embedding path $Q(\hat{e})$ was computed in the current phase; in other words, $\hat{E}'_i = \hat{E} \setminus F_i$.

Consider the set $F_i \subseteq \hat{E}$ of the fake edges that the algorithm constructed. Then, at the end of the algorithm, for every edge $\hat{e} \in F_i$, at least one endpoint of e lies in U. Recall that, from the definition of the set U, $\mathsf{dist}_{G' \setminus E'}(U, S''(X)) > 2d$ holds at the end of the algorithm. Since $G' = G \setminus E^{\mathrm{del}}$, and since $E' \subseteq A'_i$, we get that $\mathsf{dist}_{G \setminus (E^{\mathrm{del}} \cup A'_i \cup A''_i)}(U, S''(X)) > 2d$.

Let S_1 be the set of all terminals lying in $B_{G' \setminus E'}(U, d)$, and let S_2 be the set of all terminals lying in $B_{G' \setminus E'}(S''(X), d)$ at the end of the algorithm. Clearly, either $|S_1| \leq |T|/2$, or $|S_2| \leq |T|/2$ must hold. Assume first that $|S_2| \leq |T|/2$. Recall that, from Inequality 10, $|S''(X)| \geq \frac{|T|}{2^{6/\delta}}$. We then define $T_i^{\mathrm{del}} = S''(X)$. Clearly, for every vertex $t \in T_i^{\mathrm{del}}$, $B_{G \setminus (E^{\mathrm{del}} \cup A'_i \cup A''_i)}(t, d)$ contains at most $|T|/2 = N^k/2$ vertices of T. We then output T_i^{del}, A'_i, A''_i, $\hat{E}'_i$, and the embedding of the edges in $\hat{E}'_i$, as the outcome of the current phase, and the current phase is considered a bad phase. Notice that $|T_i^{\mathrm{del}}| = |S''(X)| \geq \frac{|T|}{2^{6/\delta}} \geq \frac{N^k}{m^{O(\delta^3)}}$ as required.

Lastly, we assume that $|S_1| \leq |T|/2$. In this case, we let $T_i^{\mathrm{del}} = U$. As before, we are guaranteed that, for every vertex $t \in T_i^{\mathrm{del}}$, $B_{G \setminus (E^{\mathrm{del}} \cup A'_i \cup A''_i)}(t, d)$ contains at most $|T|/2 = N^k/2$ vertices of T. Moreover, every edge in F_i now has an endpoint in T_i^{del}. We then output T_i^{del}, A'_i, A''_i, and $\hat{E}'_i$, together with the embedding of the edges in $\hat{E}'_i$, and the current phase is considered good.

We now bound the running time of Step 2. First, the total update time of data structure DS_1, from Theorem 2.8 is bounded by $O\left(|E(G)|^{1+O(\delta)}\right)$, as discussed above. The time required to maintain the ES-Tree data structure is bounded by $O(md)$. For every edge $e \in \hat{E}'_i$, we may spend time $O(d \cdot 2^{O(1/\delta^6)})$

in order to compute its embedding. The time spent on inspecting the edges of $\hat{E}$ that are eventually added to F_i, and the time spent on computing the sets S_1, S_2 of vertices is bounded by $O(m)$. Therefore, the total running time of Step 1 is bounded by:

$$O\left(m^{1+O(\delta)}\right) + O\left(|\hat{E}'_i| \cdot d \cdot 2^{O(1/\delta^6)}\right).$$

Since the running time of Step 1 is bounded by $O\left(m^{1+O(\delta)} \cdot \eta\right)$, we get that the total running time of the ith phase is bounded by:

$$O\left(m^{1+O(\delta)} \cdot \eta\right) + O\left(|\hat{E}'_i| \cdot d \cdot 2^{O(1/\delta^6)}\right).$$

Lastly, since, from Observation 5.7, the number of phases is bounded by $O\left(m^{O(\delta)} \cdot k \cdot d \cdot 2^{O(1/\delta^6)}\right)$, and since $\sum_i |\hat{E}'_i| \leq |E(W_k)| \leq N^k \cdot \Delta \cdot k \leq O(mk)$, we get that the total running time of the algorithm is bounded by:

$$O\left(m^{1+O(\delta)} \cdot k \cdot d \cdot 2^{O(1/\delta^6)} \cdot \eta\right) + O\left(mk \cdot d \cdot 2^{O(1/\delta^6)}\right) \leq O\left(m^{1+O(\delta)} \cdot \eta^2\right).$$

since, from the problem definition, $\eta > k \cdot d \cdot 2^{c/\delta^6}$, for a large enough constant c.

6 Third Tool: Router Witness

In this section we show that an embedding of a nice router $W_k^{N,\Delta}$ into a graph C can be used to certify that the graph C is a strong router, and also provide an algorithm to compute a sparse subgraph of C that is a strong enough router. We start with the following lemma.

Lemma 6.1 *Let C be a graph, and let $\alpha \geq 1, \Delta \geq 1$ be parameters such that, for every vertex $v \in V(C)$, $\deg_C(v) \leq \alpha \cdot \Delta$. Assume that we are given a graph W, that is a properly pruned subgraph of $W_k^{N,\Delta}$, for some parameters N and k, and an embedding $\mathcal{P}$ of W into C via paths of length at most d^*, that cause congestion at most η^*. Assume further that every vertex $v \in V(C)$ lies on at least $\frac{\Delta}{\beta}$ paths in $\mathcal{P}$. Then C is a $(\overline{\deg}_C, \tilde{d}, \tilde{\eta})$-router with respect to the set $V(C)$ of its vertices, where $\tilde{d} = 22d^* \cdot k^2$, and $\tilde{\eta} = 2\alpha \cdot \beta \cdot k^{4k+1} \cdot d^* \cdot \eta^*$.*

Proof: Consider any $\overline{\deg}_C$-restricted demand $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ on $V(C)$. It is enough to show that this demand can be routed in C via paths of length at most $\tilde{d}$, with congestion at most $\tilde{\eta}$. Our algorithm uses a parameter $q = \left\lceil \frac{\Delta}{\beta} \right\rceil$. For brevity, we will denote the graph $W_k^{N,\Delta}$ by W_k in this proof. We will define a new demand $\mathcal{D}' = \left(\Pi', \{D(a',b')\}_{(a',b')\in\Pi'}\right)$ over the vertices of W. In order to do so, we will define, for every pair $(a,b) \in \Pi$, a collection $M(a,b)$ of q pairs of vertices in $V(W)$, called *proxy pairs for* (a,b). For every pair $(a',b') \in M(a,b)$, we will set its demand $D'(a',b') = \frac{D(a,b)}{q}$. We will also compute a partial routing $F(a,b)$ of the demand (a,b), where, for every pair $(a',b') \in M(a,b)$, vertex a sends $\frac{D(a,b)}{q}$ flow units to a' and vertex b sends $\frac{D(a,b)}{q}$ flow units to b'. Demand $\mathcal{D}'$ is then obtained by setting $\Pi' = \bigcup_{(a,b)\in\Pi} M(a,b)$. We will show that demand $\mathcal{D}'$ is properly restricted, and then use Theorem 4.3 to compute its routing f' in W. By using the embedding of W into C, we can then obtain a routing f'' of $\mathcal{D}'$ in C. Lastly, by combining f'' with the partial routings $F(a,b)$ that

we computed for all pairs $(a,b) \in \Pi$, we will obtain the final routing of $\mathcal{D}$. We now define the demand $\mathcal{D}'$, the proxy pairs, and the partial routings $F(a,b)$ for pairs $(a,b) \in \Pi$ formally.

Consider a vertex $v \in V(C)$, and let $\mathcal{P}(v) \subseteq \mathcal{P}$ be the collection of paths that contain v, so $|\mathcal{P}(v)| \geq \frac{\Delta}{\beta}$. We discard paths from $\mathcal{P}(v)$ until $|\mathcal{P}(v)| = \left\lceil \frac{\Delta}{\beta} \right\rceil$ holds. Next, we construct a multiset $S(v)$ of q vertices of $V(W)$, and, for every vertex $u \in S(v)$, a path $R_v(u)$, as follows. Consider some path $P \in \mathcal{P}(v)$, and let $e \in E(W)$ be the edge that is embedded into P. Recall that exactly one endpoint of e is a leaf vertex of W_k; denote this vertex by u. We add u to the multiset $S(v)$, and we set the corresponding path $R_v(u)$ to be the subpath of P between v and u. Note that u may serve as an endpoint of several paths in $\mathcal{P}(v)$. In such cases, we add several copies of u to the set $S(v)$, and for each copy we include a corresponding path $R_v(u)$ that is a subpath of a distinct path in $\mathcal{P}(v)$. To avoid clutter, abusing the notation, we denote each such path by $R_v(u)$, and we assume that the corresponding copy of u in $S(v)$ is implicit in this definition. We denote $\mathcal{R}(v) = \{R_v(u) \mid u \in S(v)\}$. Observe that $|\mathcal{R}(v)| = q$, and every path in $\mathcal{R}(v)$ is a subpath of a distinct path in $\mathcal{P}(v)$. Let $\mathcal{R} = \bigcup_{v \in V(C)} \mathcal{R}(v)$.

Since the set $\mathcal{P}$ of paths causes congestion at most η^* in C, and the length of every path in $\mathcal{P}$ is at most d^*, we get that the total congestion caused by the multiset of paths $\mathcal{P}' = \bigcup_{v \in V(C)} \mathcal{P}(v)$ in C is at most $d^* \cdot \eta^*$. Therefore, the congestion caused by $\mathcal{R}$ in C is also at most $d^* \cdot \eta^*$.

Next, we define a new demand $\mathcal{D}' = \left(\Pi', \{D'(a,b)\}_{(a,b) \in \Pi'}\right)$ over the vertices of $V(W)$, as follows. We start with $\Pi' = \emptyset$, so $\mathcal{D}'$ is empty, and then process the pairs $(a,b) \in \Pi$ one by one. When a pair (a,b) is processed, we define an arbitrary perfect matching $M(a,b)$ between vertices of $S(a)$ and vertices of $S(b)$; recall that $|S(a)| = |S(b)| = q$. We then add every pair $(a',b') \in M(a,b)$ to Π', and set its demand $D'(a',b') = \frac{D(a,b)}{q}$. Additionally, we define a *partial routing* $F(a,b)$ of the demand (a,b), in which a sends $\frac{D(a,b)}{q}$ flow units to every vertex $a' \in S(a)$, and b sends $\frac{D(a,b)}{q}$ flow units to every vertex $b' \in S(b)$, as follows: for every vertex $a' \in S(a)$, we send $\frac{D(a,b)}{q}$ flow units from a via the path $R_a(a') \in \mathcal{R}(a)$, and for every vertex $b' \in S(b)$, we sent $\frac{D(a,b)}{q}$ flow units from b via the path $R_b(b') \in \mathcal{R}(b)$. This completes the description of the demand $\mathcal{D}'$, and of the partial routing $F = \bigcup_{(a,b) \in \Pi} F(a,b)$.

Consider any vertex $v \in V(C)$, and let $\Pi(v) \subseteq \Pi$ be the collection of all demand pairs in wich v participates in $\mathcal{D}$. Since demand $\mathcal{D}$ is $\overline{\deg}_C$-restricted, $\sum_{(v,u) \in \Pi(v)} D(v,u) \leq \deg_C(v) \leq \alpha \cdot \Delta$. For every demand pair $(v,u) \in \Pi(v)$, the partial routing $F(v,u)$ sends $\frac{D(v,u)}{q}$ flow units on every path in $\mathcal{R}(v)$. Therefore, overall, the partial flow F sends at most $\frac{\alpha\Delta}{q}$ flow units on every path in $\mathcal{R}(v)$. Since the paths in $\mathcal{R} = \bigcup_{v \in V(C)} \mathcal{R}(v)$ cause congestion at most $d^* \cdot \eta^*$, the congestion of the partial flow F in C is bounded by:

$$d^* \cdot \eta^* \cdot \frac{\alpha\Delta}{q} \leq d^* \cdot \eta^* \cdot \alpha \cdot \beta.$$

since $q = \left\lceil \frac{\Delta}{\beta} \right\rceil$.

Consider now some leaf vertex $u \in V(W)$. From the definition of graph W_k, the degree of u in W is at most $\Delta \cdot k$. Therefore, u serves as an endpoint of at most $\Delta \cdot k$ paths in $\mathcal{P}$. Since the length of every path $P \in \mathcal{P}$ is at most d^*, we get that u serves as an endpoint of at most $\Delta \cdot k \cdot d^*$ paths in the multiset $\mathcal{P}' = \bigcup_{v \in V(C)} \mathcal{P}(v)$, and hence u serves as an endpoint of at most $\Delta \cdot k \cdot d^*$ paths in $\mathcal{R}$. Consider now the total demand $D^*(u) = \sum_{(u,x) \in \Pi'} D'(u,x)$ on the vertex u in $\mathcal{D}'$. For every pair $(u,x) \in \Pi'$, there is some path $R \in \mathcal{R}$, such that the partial flow F sends $D'(u,x)$ flow units along R to u. Since u serves as endpont of at most $\Delta \cdot k \cdot d^*$ paths in $\mathcal{R}$, and since the total flow along each

such path, as we have shown above, is bounded by $\frac{\alpha\Delta}{q} \leq \alpha\beta$, we get that $D^*(u) \leq \alpha \cdot \beta \cdot \Delta \cdot k \cdot d^*$. We conclude that demand $\mathcal{D}'$ is $(\alpha \cdot \beta \cdot \Delta \cdot k \cdot d^*)$-restricted.

Let $\mathcal{D}''$ be the demand obtained from $\mathcal{D}'$, after we scale each demand $D'(a,b)$ down by factor $\alpha \cdot \beta \cdot k^{4k+1} \cdot d^*$. Then it is immediate to verify that $\mathcal{D}''$ is a $\frac{\Delta}{k^{4k}}$-restricted demand for W. From Theorem 4.3, there is a routing f'' of $\mathcal{D}''$ in W via paths of length at most $20k^2$, with no edge-congestion. By scaling the flow f'' up by factor $\alpha \cdot \beta \cdot k^{4k+1} \cdot d^*$, we obtain a routing f' of the demand $\mathcal{D}'$ in W via paths of length at most $20k^2$, with congestion at most $\alpha \cdot \beta \cdot k^{4k+1} \cdot d^*$.

Consider now any flow-path Q that is used in the routing f'. Let Q' be a path in graph C, that is obtained from Q, by replacing every edge $e \in E(Q)$ by its corresponding embedding path $P(e) \in \mathcal{P}$. It is then easy to see that the length of Q' is bounded by $20k^2 \cdot d^*$. By setting the flow $f(Q')$ on each such path Q' to be $f'(Q)$, we obtain a routing f of $\mathcal{D}'$ in graph C, via paths of length at most $20k^2 \cdot d^*$. Since the congestion of the routing f' in W is at most $\alpha \cdot \beta \cdot k^{4k+1} \cdot d^*$, and the congestion caused by the set $\mathcal{P}$ of embedding paths in C is at most η^*, it is easy to verify that the congestion of f is at most $\alpha \cdot \beta \cdot k^{4k+1} \cdot d^* \cdot \eta^*$.

Lastly, by combining the partial routing F with the routing f, we obtain a final routing f^* of the demand $\mathcal{D}$ in C, via paths of length at most $22k^2 \cdot d^*$, with congestion at most:

$$\alpha \cdot \beta \cdot k^{4k+1} \cdot d^* \cdot \eta^* + \alpha \cdot \beta \cdot d^* \cdot \eta^* \leq 2\alpha \cdot \beta \cdot k^{4k+1} \cdot d^* \cdot \eta^*.$$

□

Recall that a Router Decomposition requires that, for every cluster $C \in \mathcal{C}$, we provide a subgraph $C' \subseteq C$ with $V(C') = V(C)$ and $|E(C')| \leq \Delta^* \cdot |V(C)|$, such that C' is a $(\overline{\Delta}^*, \tilde{d}, \tilde{\eta})$-router for the set $V(C)$ of supported vertices. We next describe how C' is constructed, and how we certify that it is indeed a router with required properties. We will call C' a *sparsified router*.

6.1 A Sparsified Router

In this subsection we assume that we are given a simple graph C, together with parameters $1 \leq \Delta^* \leq \Delta$. Assume further that we are given another graph W, that is a properly pruned subgraph of $W_k^{N,\Delta}$, for some parameters N and k, together with an embedding $\mathcal{P}$ of W into C via paths of length at most d^*, that cause congestion at most η^*, such that every vertex $v \in V(C)$ lies on at least $\left\lfloor \frac{\Delta^*}{2kd^*} \right\rfloor$ paths in $\mathcal{P}$. Finally, assume that $\Delta^* \geq 2kd^*$.

We use a parameter $\Delta' = \left\lfloor \frac{\Delta^*}{2kd^*} \right\rfloor$. In order to construct a sparsified router C' for C, we select, for every superedge (u,v) of W, an arbitrary collection $E'(u,v) \subseteq E(W)$ of Δ' parallel edges (recall that, from Definition 4.1, graph W must contain at least $\frac{\Delta}{2} \geq \frac{\Delta^*}{2} \geq \Delta'$ such parallel edges). Let $W' \subseteq W$ be the graph obtained from W by including, for every superedge (u,v) of W, only the edges of $E'(u,v)$. Additionally, for every vertex $v \in V(C)$, we select an arbitrary subset $\mathcal{P}(v) \subseteq \mathcal{P}$ of Δ' paths that contain v, such that every leaf vertex $u \in V(W)$ serves as an endpoint in at most $\gamma \cdot \Delta'$ paths in the multiset $\bigcup_{v \in V(C)} \mathcal{P}(v)$, for some parameter γ. (Whenever using this tool, we will provide an explicit algorithm for constructing and maintaining the sets $\mathcal{P}(v)$ of paths for vertices $v \in V(C)$).

We let $C' \subseteq C$ be the simple graph obtained by taking the union of all paths in $\bigcup_{v \in V(C)} \mathcal{P}(v)$, and all paths of $\mathcal{P}$ that serve as embedding paths for the edges in $E(W')$. The following observation summarizes the properties of C'.

Observation 6.2 *Graph C' is a $(\overline{\Delta}^*, \tilde{d}, \tilde{\eta}')$-router with respect to the set $V(C')$ of supported vertices, where $\tilde{d} = 22d^* \cdot k^2$, and $\tilde{\eta}' = 8\gamma(d^*)^2 \cdot \eta^* \cdot k^{4k+1}$. Moreover, $|E(C')| \leq |V(C)| \cdot \Delta^*$.*

Proof: Observe first that, since graph W' is embedded into C, $|V(W')| \leq |V(C)|$. Therefore, $|E(W')| \leq |V(W')| \cdot k \cdot \Delta' \leq |V(C)| \cdot k \cdot \Delta'$. Consider the collection $\mathcal{Q}$ of paths that contains the embedding paths $P(e) \in \mathcal{P}$ for every edge $e \in E(W')$, and, for every vertex $v \in V(C)$, all paths in set $\mathcal{P}(v)$. Then $|\mathcal{Q}| \leq |E(W')| + |V(C)| \cdot \Delta' \leq 2|V(C)| \cdot k \cdot \Delta'$, and the length of every path in $\mathcal{Q}$ is bounded by d^*. Therefore, $|E(C')| \leq \sum_{P \in \mathcal{Q}} |E(P)| \leq 2|V(C)| \cdot k \cdot \Delta' \cdot d^* \leq |V(C)| \cdot \Delta^*$, since $\Delta' = \left\lfloor \frac{\Delta^*}{2k\Delta d^*} \right\rfloor$.

We now turn to prove that graph C' is a $(\overline{\Delta}^*, \tilde{d}, \tilde{\eta}')$-router with respect to the set $V(C')$ of supported vertices. Consider any $\overline{\Delta}^*$-restricted demand $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b) \in \Pi}\right)$ on $V(C')$. We now show that this demand can be routed in C' via paths of length at most $\tilde{d}$, with congestion at most $\tilde{\eta}'$. Note that, from the definition of a properly pruned subgraph of $W_k^{N,\Delta}$ (see Definition 4.1), graph W' is a properly pruned subgraph of $W_k^{N,\Delta'}$. The remainder of the proof is almost identical to the proof of Lemma 6.1.

For every vertex $v \in V(C)$, we define a multiset $S(v)$ of Δ' leaf vertices of W' as follows. Consider some path $P \in \mathcal{P}(v)$, and let $e \in E(W')$ be the edge that is embedded into P. Recall that exactly one endpoint of e is a leaf vertex of $W_k^{N,\Delta'}$; denote this vertex by u. We add u to the multiset $S(v)$, and we set the corresponding path $R_v(u)$ to be the subpath of P between v and u. As before, u may serve as an endpoint of several paths in $\mathcal{P}(v)$. In such cases, we add several copies of u to the set $S(v)$, and for each copy we include a corresponding path $R_v(u)$ that is a subpath of a distinct path in $\mathcal{P}(v)$. We denote $\mathcal{R}(v) = \{R_v(u) \mid u \in S(v)\}$. Observe that $|\mathcal{R}(v)| = \Delta'$, and every path in $\mathcal{R}(v)$ is a subpath of a distinct path in $\mathcal{P}(v)$. Let $\mathcal{R} = \bigcup_{v \in V(C)} \mathcal{R}(v)$.

Since the set $\mathcal{P}$ of paths causes congestion at most η^* in C, and the length of every path in $\mathcal{P}$ is at most d^*, we get that the total congestion caused by the multiset of paths $\mathcal{P}' = \bigcup_{v \in V(C)} \mathcal{P}(v)$ in C is at most $d^* \cdot \eta^*$. Therefore, the congestion caused by $\mathcal{R}$ in C is also at most $d^* \cdot \eta^*$.

Next, we define a new demand $\mathcal{D}' = \left(\Pi', \{D'(a,b)\}_{(a,b) \in \Pi'}\right)$ over the vertices of $V(W')$, as follows. We start with $\Pi' = \emptyset$, so $\mathcal{D}'$ is empty, and then process the pairs $(a,b) \in \Pi$ one by one. When a pair (a,b) is processed, we define an arbitrary perfect matching $M(a,b)$ between vertices of $S(a)$ and vertices of $S(b)$; recall that $|S(a)| = |S(b)| = \Delta'$. We then add every pair $(a',b') \in M(a,b)$ to Π', and set its demand $D'(a',b') = \frac{D(a,b)}{\Delta'}$. Additionally, we define a *partial routing* $F(a,b)$ of the demand (a,b), in which a sends $\frac{D(a,b)}{\Delta'}$ flow units to every vertex $a' \in S(a)$, and b sends $\frac{D(a,b)}{\Delta'}$ flow units to every vertex $b' \in S(b)$, as follows: for every vertex $a' \in S(a)$, we send $\frac{D(a,b)}{\Delta'}$ flow units from a via the path $R_a(a') \in \mathcal{R}(a)$, and for every vertex $b' \in S(b)$, we sent $\frac{D(a,b)}{\Delta'}$ flow units from b via the path $R_b(b') \in \mathcal{R}(b)$. This completes the description of the demand $\mathcal{D}'$, and of the partial routing $F = \bigcup_{(a,b) \in \Pi} F(a,b)$.

Consider any vertex $v \in V(C)$, and let $\Pi(v) \subseteq \Pi$ be the collection of all demand pairs in which v participates in $\mathcal{D}$. Since demand $\mathcal{D}$ is $\overline{\Delta}^*$-restricted, $\sum_{(v,u) \in \Pi(v)} D(v,u) \leq \Delta^*$. For every demand pair $(v,u) \in \Pi(v)$, the partial routing $F(v,u)$ sends $\frac{D(v,u)}{\Delta'}$ flow units on every path in $\mathcal{R}(v)$. Therefore, overall, the partial flow F sends at most $\frac{\Delta^*}{\Delta'} \leq 4kd^*$ flow units on every path in $\mathcal{R}(v)$ (we have used the fact that $\Delta' = \left\lfloor \frac{\Delta^*}{2kd^*} \right\rfloor$). Since the paths in $\mathcal{R} = \bigcup_{v \in V(C)} \mathcal{R}(v)$ cause congestion at most $d^* \cdot \eta^*$, the congestion of the partial flow F in C' is bounded by $4k(d^*)^2 \cdot \eta^*$.

Consider now some leaf vertex $u \in V(W')$, and recall that it may serve as an endpoint of at most $\gamma \cdot \Delta'$ paths in the multiset $\bigcup_{v \in V(C)} \mathcal{P}(v)$. Consider the total demand $D^*(u) = \sum_{(u,x) \in \Pi'} D'(u,x)$ on the vertex u in $\mathcal{D}'$. For every pair $(u,x) \in \Pi'$, there is some path $R \in \mathcal{R}$, such that the partial flow F sends $D'(u,x)$ flow units along R to u. Since u serves as endpont of at most $\gamma \cdot \Delta'$ paths in $\mathcal{R}$, and since the total flow along each such path, as we have shown above, is bounded by $4kd^*$, we get that $D^*(u) \leq 4\gamma k \cdot d^* \cdot \Delta'$. We conclude that demand $\mathcal{D}'$ is $(4\gamma k \cdot d^* \cdot \Delta')$-restricted.

Let $\mathcal{D}''$ be the demand obtained from $\mathcal{D}'$, after we scale each demand $D'(a,b)$ down by factor $4\gamma d^* \cdot k^{4k+1}$. Then it is immediate to verify that $\mathcal{D}''$ is a $\frac{\Delta'}{k^{4k}}$-restricted demand for W'. From Theorem 4.3, there is a routing f'' of $\mathcal{D}''$ in W via paths of length at most $20k^2$, with no edge-congestion. By scaling the flow f'' up by factor $4\gamma d^* \cdot k^{4k+1}$, we obtain a routing f' of the demand $\mathcal{D}'$ in W' via paths of length at most $20k^2$, with congestion at most $4\gamma d^* \cdot k^{4k+1}$.

Consider now any flow-path Q that is used in the routing f'. Let Q' be a path in graph C', that is obtained from Q, by replacing every edge $e \in E(Q)$ by its corresponding embedding path $P(e) \in \mathcal{P}$. It is then easy to see that the length of Q' is bounded by $20k^2 \cdot d^*$. By setting the flow $f(Q')$ on each such path Q' to be $f'(Q)$, we obtain a routing f of $\mathcal{D}'$ in graph C, via paths of length at most $20k^2 \cdot d^*$. Since the congestion of the routing f' in W' is at most $4\gamma d^* \cdot k^{4k+1}$, and the congestion caused by the set $\mathcal{P}$ of embedding paths in C is at most η^*, it is easy to verify that the congestion of f is at most $4\gamma d^* \cdot \eta^* \cdot k^{4k+1}$.

Lastly, by combining the partial routing F with the routing f, we obtain a final routing f^* of the demand $\mathcal{D}$ in C, via paths of length at most $22k^2 \cdot d^*$, with congestion at most:

$$4\gamma d^* \cdot \eta^* \cdot k^{4k+1} + 4k(d^*)^2 \cdot \eta^* \leq 8\gamma (d^*)^2 \cdot \eta^* \cdot k^{4k+1}.$$

□

7 Fourth Tool: Decremental Low-Diameter Clustering

7.1 Problem Definition

We assume that we are given an undirected n-vertex graph G that undergoes an online adversarial sequence of edge deletions. We are also given a parameter $1 < k < \log n$, and we use another parameter $d = 4k^3$. We will sometimes refer to subgraphs of G as *clusters*. Next, we need the definition of a settled cluster, and of a valid clustering.

Definition 7.1 (Settled Cluster) *We say that a cluster $C \subseteq G$ is* settled*, if either $|V(C)| \leq n^{1/k}$; or there is a vertex $v \in V(C)$, such that $|B_C(v,d)| \geq |V(C)|^{1-1/k}$. A cluster that is not settled is called* unsettled*.*

Definition 7.2 (Valid Clustering) *A valid clustering is a collection $\mathcal{C}$ of clusters of G, such that:*

- *every cluster $C \in \mathcal{C}$ is settled; and*
- *every edge in the current graph G lies in exactly one of the sets $\{E(C)\}_{C \in \mathcal{C}}$.*

We are now ready to define the Decremental Low-Diameter Clustering problem.

Definition 7.3 (Low-Diameter Clustering problem) *In the Decremental Low-Diameter Clustering problem, the input is a simple n-vertex graph G, and a parameter $1 < k < \log n$. Let $d = 4k^3$. At the beginning of the algorithm, an initial valid clustering $\mathcal{C}$ of the initial graph G must be produced by the algorithm. Over the course of the algorithm, the adversary maintains a partition of the set $\mathcal{C}$ of clusters into two subsets: set $\mathcal{C}^I$ of inactive clusters, and set $\mathcal{C}^A$ of active clusters. Once a cluster $C \in \mathcal{C}$ is added to $\mathcal{C}^I$, it may no longer be modified, and it remains in $\mathcal{C}^I$ until the end of the algorithm. Immediately after the initialization, $\mathcal{C}^A = \mathcal{C}$ and $\mathcal{C}^I = \emptyset$ holds.*

The algorithm proceeds over a number of phases. At the beginning of every phase, the adversary may delete some edges and vertices from some of the clusters in $\mathcal{C}^A$, and move some clusters from $\mathcal{C}^A$ to $\mathcal{C}^I$, after which every cluster that remains in $\mathcal{C}^A$ is guaranteed to become unsettled.

The algorithm is then required to update the current clustering, via a sequence of cluster splitting operations*, defined as follows. In every cluster splitting operation, we start with a cluster $C \in \mathcal{C}^A$, and a vertex-induced subgraph $C' \subseteq C$ with $|V(C')| \leq |V(C)|^{1-1/k}$. Cluster C' is then added to $\mathcal{C}$ and $\mathcal{C}^A$ as a new cluster, and the edges of $E(C')$ are deleted from C. Additionally, some vertices that have become isolated in C can be deleted from C. At the end of the phase, all clusters in $\mathcal{C}^A$ must be settled.*

For every vertex v, let n_v be the total number of clusters to which v ever belonged over the course of the algorithm. Then we additionally require that $\sum_{v \in V(G)} n_v \leq 2n^{1+1/k}$ holds.

The main result of the current section is an algorithm for the Low-Diameter Clustering problem, that is summarized in the following theorem; we note that the algorithm relies on the standard ball-growing technique, and a standard approach for computing neighborhood covers.

Theorem 7.4 *There is a deterministic algorithm for the Decremental Low-Diameter Clustering problem, such that, if m is the number of edges in the initial graph G, then the running time of the initialization, and the running time of each of the phases, is bounded by $O(m^{1+1/k^3})$.*

In the remainder of this section, we prove the above theorem. We start by describing our main subroutine that is based on the Ball Growing technique, whose purpose is to perform a single split of a cluster. We then provide our second main subroutine, that processes a single cluster, via a sequence of such split operations. Finally, we provide the description of the entire algorithm.

7.2 First Main Subroutine: Ball Growing Procedure

Our first main subroutine is given as input a cluster $C \in \mathcal{C}$ (which, presumably, is currently unsettled). The subroutine selects a vertex $v \in V(C)$, and computes an *eligible index* $1 \leq i \leq d$, that is defined below. If $|B_C(v,i)| \geq |V(C)|^{1-1/k}$, then the subroutine correctly determines that C is settled and terminates. Otherwise, it creates a new cluster C', that is a subgraph of C induced by the set $B_C(v,i)$ of vertices. Our algorithm will then split C' off from the cluster C. Eventually, the same subroutine will be repeatedly applied to the cluster C until it becomes is settled.

From now on we fix a cluster $C \in \mathcal{C}$, and a vertex $v \in V(C)$, and describe our first main subroutine, whose purpose is to compute an *eligible index* $1 \leq i \leq d$, that we now define.

For all $i > 0$, let L_i be the ith layer of the Breadth First Search in C starting from v; in other words, $L_i = B_C(v,i) \setminus B_C(v,i-1)$. Let $L_0 = \{v\}$. For convenience, we denote by $\hat{N} = |V(C)|$, and, for $i \geq 0$, we denote by $N_i = |L_0| + |L_1| + \cdots + |L_i|$. We also denote by $\hat{E}_i$ the set of all edges of C with both endpoints in $L_0 \cup \cdots \cup L_i$. We are now ready to define an eligible index.

Definition 7.5 (Eligible Index) *For $i \geq 1$, we say that index i is* eligible*, if the following three conditions hold:*

- $N_i \leq N_{i-1} \cdot \hat{N}^{1/k^3}$;
- $N_i < \hat{N}^{1-1/k}$; *and*
- $|\hat{E}_i| \leq |\hat{E}_{i-1}| \cdot \hat{N}^{1/k^3}$.

The following claim summarizes our first main subroutine.

Claim 7.6 *There is an algorithm, that, given a cluster C in the adjacency list representation, and a vertex $v \in V(C)$ that is not isolated in C, either computes an eligible index $1 \leq i \leq d$ for v, or certifies that $|B_C(v,d)| \geq \hat{N}^{1-1/k}$, so the cluster is settled. The running time of the algorithm is $O(|\hat{E}_i|) \leq O\left(|\hat{E}_{i-1}| \cdot \hat{N}^{1/k^3}\right)$ if an eligible index i is returned, and $O(|E(C)|)$ otherwise.*

Proof: The proof of the claim easily follows from the following observation.

Observation 7.7 *If $|B_C(v,d)| < \hat{N}^{1-1/k}$, then some index $1 \leq i \leq d$ is eligible.*

Proof: Assume otherwise, that is, $|B_C(v,d)| < \hat{N}^{1-1/k}$, but no index $1 \leq i \leq d$ is eligible. We say that index i is *type-1 ineligible* if $N_i > N_{i-1} \cdot \hat{N}^{1/k^3}$, and we say that it is *type-2 ineligible* if $|\hat{E}_i| > |\hat{E}_{i-1}| \cdot \hat{N}^{1/k^3}$. Since $|V(C)| = \hat{N}$ and $N_1 = 1$, it is easy to verify that at most k^3 indices may be type-1 ineligible. Since $|E(C)| \leq \hat{N}^2$, and vertex v is not isolated, it is easy to verify that at most $2k^3$ indices may be type-2 ineligible. But $d = 4k^3$, and every index $1 \leq i \leq d$ must be either type-1 or type-2 ineligible, a contradiction. □

We are now ready to describe our algorithm. The algorithm performs a BFS in C, starting from vertex v, until it either encounters an eligible index $1 \leq i \leq d$, or establishes that no such index is eligible. In the former case, it terminates the search once it reaches the ith layer of the BFS, and returns the index i. In this case, the running time of the algorithm is bounded by $O(|\hat{E}_i|) \leq O\left(|\hat{E}_{i-1}| \cdot N^{1/k^3}\right)$. In the latter case, the running time of the algorithm is bounded by $O(|E(C)|)$, and we are guaranteed that $|B_C(v,d)| > \hat{N}^{1-1/k}$. □

7.3 Second Main Subroutine: Processing a Cluster

Our second main subroutine is responsible for processing a single cluster. The input to the subroutine is a single cluster $C \in \mathcal{C}^A$. The subroutine must compute a collection $\mathcal{H} = \{C_0, C_1, \ldots, C_r\}$ of clusters, where $r \geq 0$, that will replace C in $\mathcal{C}^A$. We require that each of the resulting clusters is a subgraph of C and is a settled cluster. Additionally, we require that, for all $1 \leq j \leq r$, $|V(C_j)| \leq |V(C)|^{1-1/k}$ hold, and that every edge of C lies in exactly one of the clusters $C_0, \ldots, C_r$. Generally, we would also like to ensure that $\sum_{j=1}^{r} |V(C_j)| \leq |V(C)|^{1+1/k^3}$ holds. The problem is that we may need to apply this subroutine repeatedly to cluster C (where we will view the cluster C_0 as replacing C after each such application, while clusters $C_1, \ldots, C_r$ are viewed as newly created clusters). Eventually, we will need to ensure that the total number of vertices in all resulting new clusters is bounded by $|V(C)|^{1+1/k^3}$. In order to overcome this difficulty, we require that, for every cluster $1 \leq j \leq r$, the algorithm computes a "core" U_j, containing a large fraction of vertices of C_j. The cores $U_1, \ldots, U_r$ must be disjoint from each other, and disjoint from $V(C_0)$. We can then "charge" each such core U_j for the vertices of $V(C_j) \setminus U_j$, which may be duplicated between different clusters. We provide an algorithm for computing such a decomposition of C in the following lemma, which is a straightforward application of Claim 7.6.

Claim 7.8 *There is a deterministic algorithm, that is given as an input a graph C in the adjacency list representation with $|V(C)| \geq n^{1/k}$. The algorithm produces a collection $\mathcal{H} = \{C_0, C_1, \ldots, C_r\}$ of subgraphs of C, and, for all $1 \leq j \leq r$, a subset $U_j \subseteq V(C_j)$ of vertices, such that the following hold:*

- *every edge of $E(C)$ lies in exactly one cluster of $\mathcal{H}$;*

- *for all* $1 \leq j \leq r$, $|V(C_j)| \leq |V(C)|^{1-1/k}$;
- *every vertex* $v \in V(C)$ *lies in at most one of the sets* $U_1, \ldots, U_r, V(C_0)$;
- *for all* $1 \leq j \leq r$, $|U_j| \geq \frac{|V(C_j)|}{|V(C)|^{1/k^3}}$; *and*
- *Every cluster* $C_i \in \mathcal{H}$ *is settled.*

The running time of the algorithm is $O\left(|E(C)| \cdot |V(C)|^{1/k^3}\right)$.

Proof: At the beginning of the algorithm, we set $C_0 = C$ and $\mathcal{H} = \{C_0\}$, and then perform iterations. The jth iteration, for $j \geq 1$, is executed as follows. Let $v \in V(C_0)$ be an arbitrary vertex. If vertex v is isolated in C_0, then we set $C_j = \{v\}$ and $U_j = \{v\}$, delete v from C_0, and continue to the next iteration. From now on we assume that v is not isolated in C_0.

We apply the algorithm from Claim 7.6 to cluster C_0, and vertex v. If the algorithm establishes that $|B_{C_0}(v,d)| \geq |V(C_0)|^{1-1/k}$, then we terminate the algorithm, and return the current set $\mathcal{H}$. In this case, we say that the current iteration is good. The running time of the iteration in this case is bounded by $O(|E(C)|)$. Assume now that the algorithm from Claim 7.6 computes an eligible index $1 \leq i \leq d$. In this case, we let C_j be the subgraph of C_0 induced by the vertices of $B_{C_0}(v,i)$. From the definition of the eligible index, $|V(C_j)| \leq |V(C_0)|^{1-1/k} \leq |V(C)|^{1-1/k}$. We also set $U_j = B_{C_0}(v, i-1)$. We then delete from C_0 all vertices of U_j, and all edges that lie in $E(C_j)$. If the algorithm from Claim 7.6 returned an eligible index i, we say that the jth iteration is bad. In this case, the running time of the iteration is guaranteed to be at most $O(|\hat{E}^j|) \cdot O(|V(C)|^{1/k^3})$, where $\hat{E}_j$ is the set of all edges that were deleted from C_0 in the current iteration. Observe also that, from the definition of the eligible layer, $|U_j| \geq \frac{|V(C_j)|}{|V(C_0)|^{1/k^3}} \geq \frac{|V(C_j)|}{|V(C)|^{1/k^3}}$. Lastly, since $i \leq d$, $B_{C_j}(v,d) = V(C_j)$, so the cluster C_j is settled.

It is easy to verify that the algorithm guarantees that every edge of $E(C)$ always lies in exactly one cluster of $\mathcal{H}$, and that vertex sets U_j for clusters $C_j \in \mathcal{H} \setminus \{C_0\}$ are disjoint from each other and from C_0. It is also immediate to verify from our discussion that, for all $1 \leq j \leq r$, $|C_j| \leq |V(C)|^{1-1/k}$, and $|U_j| \geq \frac{|V(C_j)|}{|V(C)|^{1/k^3}}$. We have shown already that all clusters in $\mathcal{H} \setminus \{C_0\}$ are settled. It is easy to verify that cluster C_0 is also settled at the end of the algorithm, since the algorithm may only terminate following a good iteration, that establishes that C_0 is a settled cluster.

The algorithm has at most one good iteration, whose running time is at most $O(|E(C)|)$. If the jth iteration is bad, then its running time is at most $O(|\hat{E}^j|) \cdot O(|V(C)|^{1/k^3})$, where $\hat{E}^j$ is the set of all edges that were deleted from C_0 in iteration j. Therefore, the total running time of the algorithm is $O(|E(C)| \cdot |V(C)|^{1/k^3}$, as required. □

7.4 The Remainder of the Algorithm and its Analysis

We are now ready to complete the description of our algorithm for the Low-Diameter Clustering problem. At the beginning of the algorithm, we let the set $\mathcal{C}^A$ of active clusters contain a single cluster – the entire graph G. For simplicity, we consider the process of constructing the initial clustering $\mathcal{C}$ as the 0th phase.

In every phase, we consider every cluster $C \in \mathcal{C}^A$ that contains more than $n^{1/k}$ vertices one by one. For each cluster $C \in \mathcal{C}^A$, we apply the algorithm from Claim 7.8 to it. Let $\mathcal{H} = \{C_0, C_1, \ldots, C_r\}$ be the outcome of the algorithm. We then add $C_1, \ldots, C_r$ to $\mathcal{C}$ and to $\mathcal{C}^A$ as new clusters, and we replace

C with C_0 in both sets, by deleting edges and vertices from C that do not lie in C_0. We notice that this update can be implemented via a sequence of r cluster splitting operations, where, for $1 \leq j \leq r$, the jth such operation splits cluster C_j from C. This completes the description of the algorithm.

In order to analyze this algorithm, it will be convenient to define a *partitioning tree*. For every cluster C that ever lied in $\mathcal{C}$, let $\hat{C}$ denote the same cluster when it was first added to $\mathcal{C}$, and let $\hat{\mathcal{C}}$ denote the collection of all graphs $\hat{C}$, where C is a cluster that ever lied in $\mathcal{C}$. Let T be a tree, whose vertex set is $\left\{v_{\hat{C}} \mid \hat{C} \in \hat{\mathcal{C}}\right\}$. Consider now any graph $\hat{C} \in \hat{\mathcal{C}}$, and any application of the algorithm from Claim 7.8 to the corresponding cluster C. Let $\mathcal{H} = \{C_0, C_1, \ldots, C_r\}$ be the collection of clusters computed by Claim 7.8. Then we say that $C_1, \ldots, C_r$ are *child clusters* of $\hat{C}$, and, for all $1 \leq j \leq r$, we add an edge $(v_{\hat{C}}, v_{\hat{C}_j})$ to the tree. We also denote the set U_j of vertices corresponding to $\hat{C}_j$ by $U(\hat{C}_j)$. This completes the description of the partitioning tree. We say that a cluster $\hat{C} \in \hat{\mathcal{C}}$ is *small* if $|V(\hat{C})| \leq n^{1/k}$, and we say that it is *large* otherwise.

For every cluster $\hat{C} \in \hat{\mathcal{C}}$, we denote by $\mathcal{W}(\hat{C})$ the collection of its child clusters. Note that, if $\hat{C}$ is a small cluster, then $\mathcal{W}(\hat{C}) = \emptyset$. Otherwise, from Claim 7.8, we are guaranteed that, for each child cluster $\hat{C}'$ of $\hat{C}$, $|V(\hat{C}')| \leq |V(C)|^{1-1/k} \leq \frac{|V(\hat{C})|}{n^{1/k^2}}$. Additionally, we are guaranteed that, for each such cluster $\hat{C}'$, $|U(\hat{C}')| \geq \frac{|V(\hat{C}')|}{|V(\hat{C})|^{1/k^3}}$. Moreover, the sets $\left\{U(\hat{C}') \mid \hat{C}' \in \mathcal{W}(\hat{C})\right\}$ of vertices are mutually disjoint. It is then easy to verify that:

$$\sum_{\hat{C}' \in \mathcal{W}(C)} |V(\hat{C}')| \leq \sum_{\hat{C}' \in \mathcal{W}(C)} |U(\hat{C}')| \cdot |V(\hat{C})|^{1/k^3} \leq |V(\hat{C})|^{1+1/k^3} \tag{11}$$

For all $i \geq 0$, let S_i denote the set of vertices of the tree T that lie at distance exactly i from the root, and let $\mathcal{C}^i = \left\{\hat{C} \in \hat{\mathcal{C}} \mid v_{\hat{C}} \in S_i\right\}$ be the corresponding collection of clusters. Then, from the above discussion, for every index $i \geq 0$, every cluster $\hat{C} \in \mathcal{C}^i$ contains at most n^{1-i/k^2} vertices, and so the height of the partitioning tree is at most k^2.

From Inequality 11, it is easy to verify that, for all $1 \leq i \leq k^2$:

$$\sum_{\hat{C} \in \mathcal{C}^i} |V(\hat{C})| \leq \sum_{\hat{C} \in \mathcal{C}^{i-1}} |V(\hat{C})|^{1+1/k^3} \leq \left(\sum_{\hat{C} \in \mathcal{C}^{i-1}} |V(\hat{C})|\right) \cdot n^{1/k^3}.$$

Overall, we get that $\sum_{v \in V(G)} n_v = \sum_{\hat{C} \in \hat{\mathcal{C}}} |V(\hat{C})| \leq 2n^{1+1/k}$, as required.

It now remains to bound the running time of each phase. Recall that the time required to process a single cluster C by the algorithm from Claim 7.8 is bounded by $O(|E(C)| \cdot |V(C)|^{1/k^3})$. At the beginning of every phase, all clusters of $\mathcal{C}^A$ that need to be processed are disjoint in their edges, and these are the only clusters processed in that phase. Therefore, the running time of each phase (including the initialization phase) is bounded by $O(\hat{m}^{1+1/k^3})$.

8 Combining All Tools: the Proof of Theorem 3.2

In this section we complete the prooof of Theorem 3.2 by combining the tools presented in Sections 4–7.

In order to simplify the notation in this subsection, we call a graph G and parameters n, k, δ, Δ and Δ^* that satisfy the conditions of Theorem 3.2 (except for the constraint on vertex degrees in G) as

"valid input graph and parameters". For completeness we define them below.

Definition 8.1 (Valid input graph and parameters) *Assume that we are given a parameter n, that is greater than a large enough constant, and additional parameters $512 \leq k \leq (\log n)^{1/49}$, $\frac{1}{k} \leq \delta \leq \frac{1}{400}$, Δ^*, and Δ, such that $n^{18/k} \leq \Delta^* \leq \frac{\Delta}{n^{8/k^2}} \leq n$ holds, and k, Δ, Δ^* and $1/\delta$ are integers. Lastly, assume that we are given a simple graph G with $|V(G)| \leq n$. We refer to a graph G and parameters $n, k, \delta, \Delta, \Delta^*$ that satisfy these constraints as a* valid input graph G with valid input parameters n, k, δ, Δ and Δ^*.

Throughout, we use a large enough constant c from from the definition of the EmbedOrScatter problem (see Definition 5.3).

Given a valid input graph G with valid input parameters n, k, δ, Δ and Δ^*, we define additional parameters $d^* = k^{10} \cdot 2^{c/\delta^6}$, $\eta^* = 2^{c/\delta^6} \cdot k^{4ck^2} \cdot n^{4/k^2}$, and $\hat{k} = k^2$. Note that, if n, k, δ, Δ and Δ^* are valid input parameters, then, since $k \leq (\log n)^{1/49}$ holds, we get that $k^{29} \leq \frac{\log n}{k^{20}}$, and so:

$$2^{k^{29}} \leq n^{1/k^{20}}. \tag{12}$$

Additionally, since n is sufficiently large, and $k^{49} \leq \log n$ holds, we can assume that $ck^{40} \leq c(\log n)^{40/49} \leq \log n$ holds, and so $2^{ck^{40}} \leq n$. Since $\delta \geq 1/k$, we get that:

$$2^{c/\delta^6} \leq 2^{ck^{10}} \leq n^{1/k^{10}}. \tag{13}$$

From the definition of d^* and inequality 13, we get that:

$$d^* = 2^{c/\delta^6} \cdot k^{10k} \leq 2^{ck^6+11k} \leq n^{1/k^6}. \tag{14}$$

Lastly, since $\eta^* = 2^{c/\delta^6} \cdot k^{4ck^2} \cdot n^{4/k^2}$, we get that:

$$\eta^* \cdot d^* = 2^{2c/\delta^6} \cdot k^{8ck^2} \cdot n^{4/k^2} \leq 2^{4ck^6} \cdot n^{4/k^2} \leq n^{5/k^2}. \tag{15}$$

The proof of Theorem 3.2 consits of two parts. In the first part, we provide an algorithm that maintains the collection $\mathcal{C}$ of edge-disjoint clusters of G, such that every cluster $C \in \mathcal{C}$ is a $(\overline{\deg}_C, \tilde{d}, \tilde{\eta})$-router with respect to the set $V(C)$ of supported vertices. The second part provides an algorithm that, for every cluster $C \in \mathcal{C}$, maintains a subgraph $C' \subseteq C$ with $V(C') = V(C)$, such that C' is a $(\overline{\Delta}^*, \tilde{d}, \tilde{\eta})$-router for the set $V(C)$ of supported vertices, and $|E(C')| \leq \Delta^* \cdot |V(C)|$.

8.1 Part 1: Maintaining the Clustering $\mathcal{C}$

In this subsection we provide an algorithm that maintains a collection $\mathcal{C}$ of edge-disjoint clusters of G, such that every cluster $C \in \mathcal{C}$ is a $(\overline{\deg}_C, d^*, \eta^*)$-router with respect to the set $V(C)$ of supported vertices. A central notion that we use in this part is a *router certificate*, that we define next.

Definition 8.2 (Router Certificate) *Let G be a valid input graph, and let n, k, δ, Δ and Δ^* be the corresponding valid input parameters. Let $d^* = k^{10} \cdot 2^{c/\delta^6}$, $\eta^* = 2^{c/\delta^6} \cdot k^{4ck^2} \cdot n^{4/k^2}$, and $\hat{k} = k^2$. For a given subgraph (cluster) $C \subseteq G$, a* router certificate *for C consists of:*

- *a graph W_C, that is a properly pruned subgraph of $W_{\hat{k}}^{N_C,\Delta}$ for some parameter $\frac{1}{2} \cdot |V(C)|^{1/\hat{k}-1/\hat{k}^2} \leq N_C \leq (2|V(C)|)^{1/\hat{k}}$, and*

- *an embedding $\mathcal{P}_C$ of W_C into C via paths of length at most d^* that cause congestion at most η^*, such that every edge $e \in E(C)$ lies on some path in $\mathcal{P}_C$, and every vertex $v \in V(C)$ participates in at least $\frac{\Delta}{n^{4/k^2}}$ paths in $\mathcal{P}_C$.*

Moreover, graph W_C must be obtained from $W_{\hat{k}}^{N_C,\Delta}$ by applying the algorithm from Theorem 4.8 for the NiceRouterPruning *problem to it, with a single phase Φ_0 of updates, consisting of at most $\frac{N_C^{\hat{k}} \cdot \Delta}{2\hat{k}^{4\hat{k}}}$ edge deletions.*

Next, we describe the initialization step, in which we construct an initial clustering $\mathcal{C}$, and, for every cluster $C \in \mathcal{C}$, we construct a router certificate $\left(W_{\hat{k}}^{N_C,\Delta}, W_C, \mathcal{P}_C\right)$. Intuitively, from Lemma 6.1, the existence of the router certificate for each such cluster C proves that C is a $(\overline{\deg}_C, \tilde{d}, \tilde{\eta})$-router, for appropriately chosen parameters $\tilde{d}$ and $\tilde{\eta}$. Later, we describe an algorithm that maintains the clusters in $\mathcal{C}$ as the graph G undergoes k batches of edge deletions. For each cluster $C \in \mathcal{C}$, the algorithm will also maintain a subgraph $\hat{W}_C \subseteq W_C$, such that, for every vertex v that remains in C, the number of the embedding paths paths in $\left\{P(e) \in \mathcal{P}_C \mid e \in E(\hat{W}_C)\right\}$ that contain v is sufficiently large. Again, from Lemma 6.1, this is sufficient in order to ensure that each cluster $C \in \mathcal{C}$ remains a $(\overline{\deg}_C, \tilde{d}, \tilde{\eta})$-router, for appropriately chosen parameters $\tilde{d}$ and $\tilde{\eta}$. We now provide the algorithm to initialize the clustering $\mathcal{C}$, and to compute an initial router certificate for every cluster $C \in \mathcal{C}$.

8.1.1 Initialization: Constructing the Clustering and Router Certificates

The algorithm for the first part of the initialization is summarized in the following theorem.

Theorem 8.3 *There is a deterministic algorithm, that receives as input a valid input graph G together with valid input parameters n, k, δ, Δ and Δ^*, such that, for every vertex $v \in V(G)$, $\deg_G(v) \leq \Delta \cdot n^{16/k}$. The algorithm computes a collection $\mathcal{C}$ of subgraphs of G called* clusters *that are disjoint in their edges but may share vertices, with $\sum_{C \in \mathcal{C}} |V(C)| \leq n^{1+20/k}$. For every cluster $C \in \mathcal{C}$, it also computes a router certificate $\left(W_{k^2}^{N_C,\Delta}, W_C, \mathcal{P}_C\right)$. If we denote by $E^{\text{del}} = E(G) \setminus \left(\bigcup_{C \in \mathcal{C}} E(C)\right)$, then the algorithm guarantees that $|E^{\text{del}}| \leq |V(G)| \cdot \Delta \cdot n^{2/k^2}$. The running time of the algorithm is $O\left((n\Delta)^{1+O(\delta)}\right)$.*

Recall that, from the definition of a router certificate, for every cluster $C \in \mathcal{C}$, our algorithm initializes the algorithm from Theorem 4.8 for the NiceRouterPruning problem on graph $W_C = W_{\hat{k}}^{N_C,\Delta}$, that we denote by $\mathcal{A}_C$. During the phase Φ_0, Algorithm $\mathcal{A}_C$ performs an initial pruning of W_C, and the resulting graph serves as the router certificate for C. After the initialization step, as our algorithm receives batches $\pi_1, \ldots, \pi_k$ of edge deletions for graph G, we may delete additional edges from W_C, over the course of k phases $\Phi_1, \ldots, \Phi_k$, that correspond to the above batches of updates to G. We will employ the same algorithm $\mathcal{A}_C$, that was initialized as part of the algorithm in Theorem 8.3, in order to implement the additional pruning of graph W_C over the course of the phases $\Phi_1, \ldots, \Phi_k$.

The proof of Theorem 8.3 easily follows from the following lemma, that is almost identical to Theorem 8.3. The main difference is that the lemma only ensures that $\sum_{C \in \mathcal{C}} |E(C)| \geq \frac{|E(G)|}{2n^{18/k}}$, while Theorem 8.3 requires that all but at most $|V(G)| \cdot \Delta \cdot n^{2/k^2}$ edges of G lie in $\bigcup_{C \in \mathcal{C}} E(C)$. The lemma also requires that all vertex degrees in G are at least $4\Delta \cdot n^{1/k^2}$.

Lemma 8.4 *There is a deterministic algorithm, that receives as input a valid input graph G together with valid input parameters n, k, δ, Δ and Δ^*, such that, for every vertex $v \in V(G)$, $4\Delta \cdot n^{1/k^2} \leq \deg_G(v) \leq \Delta \cdot n^{16/k}$. The algorithm computes a collection $\mathcal{C}$ of subgraphs of G called* clusters *that are*

disjoint in their edges but may share vertices, with $\sum_{C\in\mathcal{C}}|V(C)|\leq 2n^{1+1/(4k^2)}$. *For every cluster* $C\in\mathcal{C}$, *it also computes a router certificate* $\left(W_{k^2}^{N_C,\Delta},W_C,\mathcal{P}_C\right)$. *The algorithm ensures that* $\sum_{C\in\mathcal{C}}|E(C)|\geq\frac{|E(G)|}{4n^{18/k}}$. *The running time of the algorithm is* $O\left((n\Delta)^{1+O(\delta)}\right)$.

We prove the lemma below, after we complete the proof of Theorem 8.3 using it.

Proof of Theorem 8.3. Our algorithm consists of at most $n^{19/k}$ iterations. We start with $G'=G$, $\mathcal{C}=\emptyset$, and $E^{\text{del}}=\emptyset$.

We now describe the ith iteration, for $i\geq 1$. We start by iteratively removing from G' all vertices whose current degree is below $4\Delta\cdot n^{1/k^2}$. All edges that were removed from G' in this step are added to set E^{del}. If no vertices remain in G', then we terminate the algorithm. Otherwise, we apply the algorithm from Lemma 8.4 to the resulting graph G', and we let $\mathcal{C}_i$ be the collection of clusters that the algorithm returns. We add the clusters in $\mathcal{C}_i$ to $\mathcal{C}$, and we delete from G' all edges that lie in $\bigcup_{C\in\mathcal{C}_i}E(C)$. Let $E_i=\bigcup_{C\in\mathcal{C}'}E(C)$, and recall that, from Lemma 8.4, $|E_i|\geq\frac{|E(G')|}{4n^{18/k}}$. We then continue to the next iteration.

We now bound the number of iterations. At the beginning of the algorithm, $|E(G)|\leq n^{1+16/k}\cdot\Delta$ holds, since all vertex degrees are bounded by $\Delta\cdot n^{16/k}$. As long as the algorithm continues, G' contains at least one vertex of degree at least $4\Delta\cdot n^{1/k^2}$. From the above calculations, after the ith iteration:

$$|E(G')|\leq|E(G)|\cdot\left(1-\frac{1}{4n^{18/k}}\right)^i\leq n^{1+16/k}\cdot\Delta\cdot\left(1-\frac{1}{4n^{18/k}}\right)^i$$

holds. Clearly, after at most $8\cdot n^{18/k}\cdot\log n$ iterations, the algorithm must terminate. Since $k\leq(\log n)^{1/10}$, we get that $n^{1/k}\geq 2^{(\log n)^{9/10}}\geq 16\log n$ (since we have assumed that n is sufficiently large). Therefore, altogether, the number of iterations is bounded by $n^{19/k}$.

It is also easy to verify that the total number of edges that were ever added to E^{del} is bounded by $4|V(G)|\Delta\cdot n^{1/k^2}\leq|V(G)|\cdot\Delta\cdot n^{2/k^2}$, and every edge of G that does not lie in E^{del} must lie in one of the clusters of $\mathcal{C}$; moreover, the clusters in $\mathcal{C}$ are disjoint in their edges. Lastly recall that, from Lemma 8.4, for all $i\geq 1$, $\sum_{C\in\mathcal{C}_i}|V(C)|\leq 2n^{1+1/(4k^2)}$. Since the number of iterations is bounded by $n^{19/k}$, we get that:

$$\sum_{C\in\mathcal{C}}|V(C)|\leq 2n^{1+1/(4k^2)}\cdot n^{19/k}\leq n^{1+20/k}.$$

It now only remains to bound the running time of the algorithm. The running time of the algorithm from Lemma 8.4 is $O\left((n\Delta)^{1+O(\delta)}\right)$, and the number of iterations is at most $n^{19/k}$. In every iteration, we also need to spend $O(|E(G)|)\leq O(n^{1+16/k}\cdot\Delta)$ time to remove low-degree vertices from G'. The total running time of the algorithm is then bounded by $O\left((n\Delta)^{1+O(\delta+1/k)}\right)\leq O\left((n\Delta)^{1+O(\delta)}\right)$, since $\delta\geq 1/k$. □

Next, we prove Lemma 8.4.

Proof of Lemma 8.4. The algorithm consists of at most $z=16k^3$ phases. Throughout the algorithm, we maintain a collection $\mathcal{C}$ of clusters that are disjoint in their edges but may share vertices. The set $\mathcal{C}$ of clusters is partitioned into two subsets: set $\mathcal{C}^I$ of *inactive* clusters, and set $\mathcal{C}^A$ of *active* clusters. For every cluster $C\in\mathcal{C}$, if $|V(C)|\leq\Delta$, then we say that it is a *small cluster*, and we say that it is a *large cluster* otherwise. Throughout the algorithm, we use a parameter $d=256k^6=4(4\hat{k})^3$. Our algorithm also maintains two sets $E_1^{\text{del}},E_2^{\text{del}}$ of edges. We ensure that the following invariants hold throughout the algorithm.

I1. If $C \in \mathcal{C}^I$ is an inactive cluster that is large, then the algorithm computes a router certificate $\left(W_{\hat{k}}^{N_C,\Delta}, W_C, \mathcal{P}_C\right)$ for C, when C is first added to $\mathcal{C}^I$. After that time, cluster C and its corresponding router certificate remain unchanged;

I2. every edge of G lies in exactly one of the sets E_1^{del}, E_2^{del}, and $\{E(C)\}_{C\in\mathcal{C}}$ at all times;

I3. $\sum_{C\in\mathcal{C}} |V(C)| \leq 2n^{1+1/(4k^2)}$ holds at all times;

I4. for all $j > 0$, after the jth phase, $|E_1^{\text{del}}| \leq 4j \cdot n^{1+1/(4k^2)} \cdot \Delta$ holds; and

I5. $|E_2^{\text{del}}| \leq n^{18/k} \cdot \sum_{C\in\hat{\mathcal{C}}^I} |E(C)|$ hold at all times, where $\hat{\mathcal{C}}^I$ is the set of all large inactive clusters.

At the beginning of the algorithm, we initialize the algorithm from Theorem 7.4 for the Low-Diameter Clustering problem on graph G, with the parameter k replaced by $4\hat{k} = 4k^2$, and obtain an initial clustering $\mathcal{C}$, such that every cluster $C \in \mathcal{C}$ is settled. For every cluster $C \in \mathcal{C}$, if $|V(C)| \leq n^{1/(4k^2)}$, then C is small, since $\Delta \geq n^{1/(4k^2)}$ holds from the definition of valid input parameters. We add each cluster C with $|V(C)| \leq n^{1/(4k^2)}$ to $\mathcal{C}^I$; and we add all other clusters to $\mathcal{C}^A$. We also initialize $E_1^{\text{del}} = \emptyset$ and $E_2^{\text{del}} = \emptyset$. Notice that Invariants I1–I5 hold at the beginning of the algorithm. We perform phases as long as $\mathcal{C}^A \neq \emptyset$ holds; we will show later that the total number of phases is bounded by $z = 16k^3$. We now describe the jth phase, for some $j \geq 1$. We assume that, at the beginning of the phase, $\mathcal{C}^A \neq \emptyset$, and Invariants I1–I5 hold.

Execution of the jth phase. In order to execute the jth phase, we process every cluster $C \in \mathcal{C}^A$ one by one. We now provide an algorithm for processing the cluster C. First, we delete all isolated vertices from C, and, if C becomes a small cluster, then we move it to $\mathcal{C}^I$. In this case, no further processing of C is needed. From now on we assume that C remains a large cluter, so $|V(C)| \geq \Delta$ holds.

Our next step is to apply the algorithm from Theorem 5.4 for the EmbedOrScatter problem to cluster C, with parameters Δ, δ and $d = 256k^6$ remaining unchanged, and parameter k replaced with $\hat{k}$. For convenience, we denote $n' = |V(C)|$. In the following simple claim, whose proof is deferred to Section D.1 of Appendix, we verify that this is indeed a valid input to the EmbedOrScatter problem.

Claim 8.5 *Graph C together with parameters Δ, δ, d and $\hat{k}$ as defined above are a valid input to the* EmbedOrScatter *problem.*

We now consider two cases. The first case happens if the algorithm for the EmbedOrScatter problem, when appied to cluster C, computed a subset $E_C \subseteq E(C)$ of at most $2|V(C)|\cdot\Delta$ edges, such that $C\setminus E_C$ is $(d, 1/\hat{k})$-scattered. In this case, we delete the edges of E_C from cluster C, and then delete isolated vertices from C. The edges of E_C are then added to set E_1^{del}. If cluster C becomes small, then we move it from $\mathcal{C}^A$ to $\mathcal{C}^I$. Otherwise, it remains in $\mathcal{C}_A$. Note that, from the definition of a scattered graph, we are now guaranteed that, for every vertex $v \in V(C)$, $|B_C(v,d)| \leq |V(C)|^{1-1/k^2} \leq |V(C)|^{1-1/(4k^2)}$ holds. In particular, cluster C is now unsettled.

Consider now the second case, when the algorithm for the EmbedOrScatter problem computes, for a parameter $N_C = \left\lfloor |V(C)|^{1/\hat{k}-1/\hat{k}^2} \right\rfloor$, a collection $F \subseteq E(W_{\hat{k}}^{N_C,\Delta})$ of at most $\frac{N_C^{\hat{k}}\cdot\Delta}{(512\hat{k})^{4\hat{k}}}$ fake edges of $W_{\hat{k}}^{N_C,\Delta}$, together with an embedding $\mathcal{P}_C$ of $W_{\hat{k}}^{N_C,\Delta}\setminus F$ into C via paths of length at most $d\cdot\hat{k}\cdot 2^{O(1/\delta^6)} = 256k^8\cdot 2^{O(1/\delta^6)} \leq d^*$ that cause congestion at most $n^{4/\hat{k}}\cdot d\cdot\hat{k}^{c\hat{k}}\cdot 2^{c/\delta^6} \leq \eta^*$ (recall that $d = 256k^6$, $\hat{k} = k^2$, $d^* = k^{10}\cdot 2^{c/\delta^6}$ and $\eta^* = 2^{c/\delta^6}\cdot k^{4ck^2}\cdot n^{4/k^2}$). Next, we will compute a router certificate for C, after possibly deleting some edges and vertices from it.

In order to do so, for every vertex $v \in V(C)$, we initialize a counter n_v to be the number of edges in graph $W_{\hat{k}}^{N_C,\Delta} \setminus F$ whose embedding path in $\mathcal{P}_C$ contains v, and we initialize a list $L(v)$ that contains all such edges. Next, we initialize the algorithm for the NiceRouterPruning problem from Theorem 4.8 on graph $W_{\hat{k}}^{N_C,\Delta}$, with parameter k replaced by $\hat{k}$ and parameter N replaced by N_C. Recall that $\hat{k} = k^2 \geq 256$ and $\Delta \geq n^{18/k} \geq 4k^4 = 4\hat{k}^2$ holds from the definition of valid input parameters (see Definition 8.1) and Inequality 12. Additionally:

$$N_C = \left\lfloor |V(C)|^{1/\hat{k}-1/\hat{k}^2} \right\rfloor \geq (n')^{1/(2k^2)} \geq n^{9/k^3} \geq 2^{k^4} \geq k^{6k^2} = \hat{k}^{3\hat{k}},$$

since C is a large cluster (so $n' = |V(C)| \geq \Delta \geq n^{18/k}$), and from Inequality 12. Therefore, we obtain a valid input to the NiceRouterPruning problem. For convenience, in the following discussion, we denote parameter N_C by N.

We denote the algorithm for the NiceRouterPruning problem from Theorem 4.8 that we initialized on graph C with the above parameters by $\mathcal{A}_C$. We now describe the online sequence of edge deletions that serves as input to $\mathcal{A}_C$. All these edge deletions are executed as part of Phase Φ_0. Initially, algorithm $\mathcal{A}_C$ receives the set F of fake edges as part of the deletion sequence. Recall that algorithm $\mathcal{A}_C$ maintains a graph W_C, that is a properly pruned subgraph of $W_{\hat{k}}^{N,\Delta}$, together with the corresponding sets $U_1, \ldots, U_{\hat{k}}$ of its vertices. As the algorithm progresses, some edges are deleted from W_C, either as part of the input sequence of edge deletions, or by algorithm $\mathcal{A}_C$ itself. Whenever an edge e is deleted from W_C, we consider its corresponding embedding path $P(e) \in \mathcal{P}_C$. For every vertex $v \in P(e)$, we decrease the corresponding counter n_v by 1, and we delete e from list $L(v)$. Whenever, for any vertex $v \in V(C)$, the counter n_v falls below $\frac{\Delta}{n^{4/k^2}}$, all the edges of $L(v)$ are deleted from W_C, by providing them to Algorithm $\mathcal{A}_C$ as part of its online sequence of edge deletions.

Recall that $|F| \leq \frac{N^{\hat{k}} \cdot \Delta}{(512\hat{k})^{4\hat{k}}}$, and, for every vertex $v \in V(C)$, when the counter n_v falls below $\frac{\Delta}{n^{4/k^2}}$, we may delete at most $\frac{\Delta}{n^{4/k^2}}$ edges from W_C.

We denote by E_0' the online sequence of edge deletions that serves as input to Algorithm $\mathcal{A}_C$ so far. Then $|E_0'|$ is bounded by:

$$\begin{aligned}
|F| + |V(C)| \cdot \frac{\Delta}{n^{4/k^2}} &\leq \frac{N^{\hat{k}} \cdot \Delta}{(512\hat{k})^{4\hat{k}}} + |V(C)| \cdot \frac{\Delta}{n^{4/k^2}} \\
&\leq \frac{N^{\hat{k}} \cdot \Delta}{(512\hat{k})^{4\hat{k}}} + N^{\hat{k}} \cdot n^{2/k^2} \cdot \frac{\Delta}{n^{4/k^2}} \\
&\leq \frac{N^{\hat{k}} \cdot \Delta}{2\hat{k}^{4\hat{k}}}.
\end{aligned}$$

We have used the fact that $N = \left\lfloor |V(C)|^{1/\hat{k}-1/\hat{k}^2} \right\rfloor$, and so $N^{\hat{k}} \geq \frac{|V(C)|}{2^{k^2} \cdot n^{1/k^2}} \geq \frac{|V(C)|}{n^{2/k^2}}$. We have also used Inequality 12.

We update the set $\mathcal{P}_C$ of embedding paths, so that it only contains paths corresponding to the edges that currently lie in W_C. Let $V' \subseteq V(C)$ be the set of vertices $v \in V'$, such that v participates in fewer than $\frac{\Delta}{n^{4/k^2}}$ of the paths in $\mathcal{P}_C$. Note that our algorithm ensures that every vertex $v \in V'$ in fact does not participate in any paths in $\mathcal{P}_C$. Consider now the graph $C' \subseteq C$ that contains all edges and vertices of C that lie on the paths in $\mathcal{P}_C$. In other words, graph C' is obtained from $C \setminus V'$, by deleting from it all edges that do not lie on the paths in $\mathcal{P}_C$. In the next simple claim, whose proof is

deferred to Section D.2 of Appendix, we prove that $\left(W_{\hat{k}}^{N,\Delta}, W_C, \mathcal{P}_C\right)$ is a valid router certificate for C', and that $|E(C')| \geq \frac{|E(C)|}{n^{18/k}}$.

Claim 8.6 *$\left(W_{\hat{k}}^{N,\Delta}, W_C, \mathcal{P}_C\right)$ is a valid router certificate for C', and $|E(C')| \geq \frac{|E(C)|}{n^{18/k}}$.*

We add the edges of $E(C) \setminus E(C')$ to the set E_2^{del}, and we *charge* these edges to the edges of C', where the charge to every edge is at most $n^{18/k}$. Next, we delete from C all edges and vertices, except those lying in C', and we move the resulting cluster C from $\mathcal{C}^A$ to $\mathcal{C}^I$. This completes the description of the procedure for processing a cluster $C \in \mathcal{C}^A$. Next, we analyze its running time.

The running time of the algorithm from Theorem 5.4 for the EmbedOrScatter problem is bounded by:

$$O\left(|E(C)|^{1+O(\delta+1/\hat{k})} \cdot d^2 \cdot 2^{O(1/\delta^6)}\right) \leq O\left(|E(C)|^{1+O(\delta+1/k^2)} \cdot k^6 \cdot 2^{O(1/\delta^6)}\right).$$

The time required to initialize the counters n_v and the lists L_v for every vertex $v \in V(C)$, and to maintain them during the pruning procedure is subsumed by the above running time. The running time of the algorithm from Theorem 4.8 for the NiceRouterPruning problem is bounded by:

$$O\left(\hat{k}^2 \cdot N^{\hat{k}} + |E_0'| \cdot \hat{k}^{O(\hat{k})}\right) \leq O\left(N^{\hat{k}} \cdot \Delta \cdot k^{O(k^2)}\right) \leq O\left(|V(C)| \cdot \Delta \cdot k^{O(k^2)}\right).$$

Altogether, the time required to process a cluster C is bounded by:

$$O\left(|E(C)|^{1+O(\delta+1/k^2)} \cdot k^6 \cdot 2^{O(1/\delta^6)}\right) + O\left(|V(C)| \cdot \Delta \cdot k^{O(k^2)}\right).$$

Recall that, from Invariant I2, the clusters in $\mathcal{C}$ do not share edges, and, from Invariant I3, $\sum_{C \in \mathcal{C}} |V(C)| \leq 2n^{1+1/(4k^2)}$ holds. Therefore, the time required to process all clusters in a single phase is bounded by:

$$\begin{aligned} O\left(|E(G)|^{1+O(\delta+1/k^2)} \cdot k^6 \cdot 2^{O(1/\delta^6)}\right) + O\left(n^{1+1/(4k^2)} \cdot \Delta \cdot k^{O(k^2)}\right) \\ \leq O\left((n\Delta)^{1+O(\delta+1/k)} \cdot 2^{O(1/\delta^6)}\right) \\ \leq \left((n\Delta)^{1+O(\delta)}\right), \end{aligned}$$

since $|E(G)| \leq n \cdot \Delta \cdot n^{O(1/k)}$, that $\delta \geq 1/k$, and Inequality 13.

Next, we verify that all invariants continue to hold after we finish processing all clusters in $\mathcal{C}^A$. Invariants I1 and I2 are immediate from our algorithm, and Invariant I3 continues to hold since the only changes to the clusters in $\mathcal{C}$ are edge- and vertex-deletions. Invariant I5 follows immediately from our charging scheme. In order to show that Invariant I4 continues to hold, it is enough to show that the total number of edges added to E_1^{del} in the current iteration is bounded by $4n^{1+1/(4k^2)} \cdot \Delta$. Recall that, for every cluster $C \in \mathcal{C}^A$ that the algorithm processes, at most $2|V(C)| \cdot \Delta$ edges are added to E_1^{del}. Since, from I3, $\sum_{C \in \mathcal{C}} |V(C)| \leq 2n^{1+1/(4k^2)}$ holds, we get that the total number of edges added to E_1^{del} in the current phase is bounded by:

$$2\Delta \cdot \sum_{C \in \mathcal{C}} |V(C)| \leq 4n^{1+1/(4k^2)} \cdot \Delta.$$

Once all clusters in $\mathcal{C}^A$ are processed, the algorithm for the Low-Diameter Clustering problem is notified of the changes to the clusters in $\mathcal{C}^A$, and to the sets $\mathcal{C}^I, \mathcal{C}^A$ of clusters. Recall that the algorithm is then required to amend the current clustering in $\mathcal{C}$, by performing cluster splitting operations to the clusters in $\mathcal{C}^A$, to ensure that all clusters in $\mathcal{C}^A$, including the newly added ones, are settled. The algorithm does not add any clusters to $\mathcal{C}^I$, and it does not add any edges to E_1^{del} or E_2^{del}. Moreover, it ensures that $\sum_{C\in\mathcal{C}} |V(C)| \leq 2n^{1+1/(4\hat{k})} = 2n^{1+1/(4k^2)}$ continues to hold. Therefore, all invariants continue to hold at the end of the phase. This completes the description of the algorithm. In the next claim we bound the number of phases in the algorithm.

Claim 8.7 *After $16k^3$ phases, $\mathcal{C}^A = \emptyset$ must hold.*

Proof: We partition the algorithm's execution $\lceil \log k \rceil$ stages. For $1 \leq j \leq \lceil \log k \rceil$, the jth stage terminates when, for every cluster $C \in \mathcal{C}^A$, $|V(C)| \leq n^{1/2^j}$ holds. At the end of the last stage, for every cluster $C \in \mathcal{C}^A$, $|V(C)| \leq n^{1/k} < \Delta$ must hold, so $\mathcal{C}^A = \emptyset$.

Observation 8.8 *For all $1 \leq j \leq \lceil \log k \rceil$, the number of phases in Stage j of the algorithm is bounded by $16k^2$.*

Note that the above observation implies that the number of phases in the algorithm is bounded by $16k^2 \cdot \lceil \log k \rceil \leq 16k^3$. It now remains to prove the observation.

Proof: In order to prove the observation it is enough to show that, for all $1 \leq i \leq 16k^2$, if C is a cluster that lies in $\mathcal{C}^A$ at the end of the ith phase of Stage j, then $|V(C)| \leq n^{\frac{2}{2^j} - \frac{i-1}{2^{j+3}\cdot k^2}}$.

The proof is by induction on i. For $i = 1$, the claim clearly holds. Consider now some integer $i > 1$, and assume that the claim holds at the end of phase $i-1$ of stage j. Let C be any cluster that lies in $\mathcal{C}^A$ at the beginning of phase i of stage j, and assume that, once the cluster is processed by the algorithm for the EmbedOrScatter problem, it remains in $\mathcal{C}^A$. In other words, the algorithm for the EmbedOrScatter problem from Theorem 5.4, when applied to cluster C, computed a subset $E_C \subseteq E(C)$ of edges, that were subsequently deleted from C, such that, after the deletion of these edges, for every vertex $v \in V(C)$, $|B_C(v,d)| \leq |V(C)|^{1-1/k^2}$ holds, and cluster C becomes unsettled.

The algorithm for the Decremental Low-Diameter Clustering problem then needs to "fix" this cluster, by splitting some clusters $C' \subseteq C$ from it; we call each such cluster C' a *child-cluster* of C. Recall that, from the definition of the Low-Diameter Clustering problem, for each child-cluster C' of C:

$$|V(C')| \leq |V(C)|^{1-\frac{1}{4k^2}}.$$

Recall that, from the induction hypothesis:

$$|V(C)| \leq n^{\frac{2}{2^j} - \frac{i-2}{2^{j+3}\cdot k^2}}.$$

At the same time, since the algorithm is still at stage j, $|V(C)| \geq n^{\frac{1}{2^j}}$. Therefore:

$$|V(C)|^{\frac{1}{4k^2}} \geq \left(n^{\frac{1}{2^j}}\right)^{\frac{1}{4k^2}} \geq n^{\frac{1}{2^{j+3}\cdot k^2}}.$$

Altogether, we get that:

$$|V(C)|^{1-1/(4k^2)} \leq n^{\frac{2}{2^j} - \frac{i-1}{2^{j+3}\cdot k^2}}, \tag{16}$$

and so $|V(C')| \leq n^{\frac{2}{2^j}-\frac{i-1}{2^{j+3}\cdot k^2}}$, as required.

Next, we denote by $\hat{C}$ the cluster C that is obtained at the end of the phase, that is, after all its child clusters were split off from it in the current phase. For convenience, we continue to use the notation C for the cluster C before the splitting operations. Then cluster $\hat{C}$ is settled, and so there is some vertex $v \in V(\hat{C})$, with $|B_{\hat{C}}(v,d)| \geq |V(\hat{C})|^{1-1/(4k^2)}$. Recall however that, after Algorithm EmbedOrScatter was applied to the cluster, $|B_C(v,d)| \leq |V(C)|^{1-1/k^2}$ held. Since $|B_{\hat{C}}(v,d)| \leq |B_C(v,d)|$ must hold, we get that:

$$|V(\hat{C})|^{1-1/(4k^2)} \leq |V(C)|^{1-1/k^2},$$

and so:

$$|V(\hat{C})| \leq |V(C)|^{1-1/k^2} \cdot |V(\hat{C})|^{1/(4k^2)} \leq |V(C)|^{1-1/(4k^2)} \leq n^{\frac{2}{2^j}-\frac{i-1}{2^{j+3}\cdot k^2}},$$

from Inequality 16. □ □

We are now ready to bound the running time of the algorithm so far. The running time of the algorithm from Theorem 7.4 for the low-diameter clustering problem is bounded by $O(|E(G)|^{1+64/k^6}) \leq O\left((n\Delta)^{1+O(1/k)}\right)$. The additional time required to execute each phase is bounded by: $O\left((n\Delta)^{1+O(\delta)}\right)$.

Therefore, the total running time of the algorithm so far is $O\left((n\Delta)^{1+O(\delta+1/k)} \cdot k^3\right) \leq O\left((n\Delta)^{1+O(\delta)}\right)$, from Inequality 12, and since $\delta \geq 1/k$.

Next, we bound the cardinality of the set E_1^{del} of edges.

Observation 8.9 *At the end of the algorithm,* $|E_1^{\text{del}}| \leq \frac{|E(G)|}{8}$.

Proof: From Claim 8.7, the number of phases is bounded by $z = 16k^3$. From Invariant I4, at the end of the last phase:

$$|E_1^{\text{del}}| \leq 4z \cdot n^{1+1/(4k^2)} \cdot \Delta \leq 64k^3 \cdot n^{1+1/(4k^2)} \cdot \Delta \leq \frac{\Delta \cdot n^{1+1/k^2}}{16} \leq \frac{|E(G)|}{8}$$

(we have used Inequality 12, and the fact that all vertex degrees in G are at least $4\Delta \cdot n^{1/k^2}$). □

Consider the set $\mathcal{C} = \mathcal{C}^I$ of clusters obtained at the end of the algorithm, and let $\hat{\mathcal{C}} \subseteq \mathcal{C}$ be the subset containing all large clusters. From Invariant I3, $\sum_{C\in\mathcal{C}} |V(C)| \leq 2n^{1+1/(4k^2)}$ holds. From Invariant I1, for every cluster $C \in \hat{\mathcal{C}}$, our algorithm has computed a router certificate $\left(W_{\hat{k}}^{N_C,\Delta}, W_C, \mathcal{P}_C\right)$ for C. Additionally, from Invariant I5, $|E_2^{\text{del}}| \leq n^{18/k} \cdot \sum_{C\in\hat{\mathcal{C}}} |E(C)|$ holds. Moreover, our charging scheme charges every edge of E_2^{del} to some edge of $\bigcup_{C\in\hat{\mathcal{C}}} E(C)$, such that the charge to every edge in $\bigcup_{C\in\hat{\mathcal{C}}} E(C)$ is at most $n^{18/k}$. It now remains to process small clusters that remain in $\mathcal{C}$. We use the following observation in order to bound the number of edges in the small clusters.

Observation 8.10 *The total number of edges lying in the small clusters of* $\mathcal{C}^I$ *is bounded by* $\frac{|E(G)|}{4}$.

Proof: Recall that, if C is a small cluster, then $|V(C)| \leq \Delta$. Since graph G is simple, for every vertex $v \in V(C)$, $\deg_C(v) \leq \Delta$. Let $\mathcal{C}' \subseteq \mathcal{C}^I$ denote the collection of all small clusters. Then, from Invariant I3:

$$\begin{aligned}\sum_{C\in\mathcal{C}'}|E(C)| &\leq \Delta\cdot\sum_{C\in\mathcal{C}}|V(C)|\\ &\leq 2\Delta\cdot n^{1+1/(4k^2)}\\ &\leq \frac{|E(G)|}{4},\end{aligned}$$

since all vertex degrees in G are at least $4\Delta\cdot n^{1/k^2}$. □

We simply delete all small clusters from $\mathcal{C}^I$, and we add all their edges to set E_1^{del}. It is immediate to verify that $\sum_{C\in\mathcal{C}}|V(C)|\leq 2n^{1+1/(4k^2)}$ continues to hold, and every edge of G belongs to exactly one of the sets $E_1^{\mathrm{del}}, E_2^{\mathrm{del}}$, and $E(C)$ for $C\in\mathcal{C}$. From our discussion, $|E_1^{\mathrm{del}}|\leq\frac{|E(G)|}{2}$, and, from our charging scheme:

$$\sum_{C\in\mathcal{C}}|E(C)|\geq\frac{|E_2^{\mathrm{del}}|}{n^{18/k}}.$$

Therefore:

$$\sum_{C\in\mathcal{C}}|E(C)|\geq\frac{\sum_{C\in\mathcal{C}}|E(C)|}{2}+\frac{|E_2^{\mathrm{del}}|}{2n^{18/k}}\geq\frac{|E(G)\setminus E_1^{\mathrm{del}}|}{2n^{18/k}}\geq\frac{|E(G)|}{4n^{18/k}}.$$

This completes the proof of Lemma 8.4. □

8.1.2 Updating the Clusters and the Router Certificates

In this subsection we provide an algorithm for maintaining the clustering $\mathcal{C}$. For every cluster $C\in\mathcal{C}$, for all $1\leq i\leq k$, the algorithm updates the cluster C and the corresponding graph W_C following the batch $\pi_i^C=\pi_i\cap E(C)$ of edge deletions from C. The algorithm for updating a single cluster $C\in\mathcal{C}$ is summarized in the following theorem. For simplicity, in the theorem, we denote π_i^C by π_i. We note that the theorem statement requires that $|\pi_i^C|$ is sufficiently small; whenever this is not the case, we will destroy the cluster completely, and move all its edges to E^{del}.

Theorem 8.11 *There is a deterministic algorithm, whose input consists of:*

- *a valid input graph G together with valid input parameters n,k,δ,Δ and Δ^*, such that all vertex degrees in G are at most $\Delta\cdot n^{16/k}$;*
- *a subgraph $C\subseteq G$ called a* cluster*, together with a router certificate $\left(W_{\hat{k}}^{N_C,\Delta},W_C,\mathcal{P}_C\right)$ for C;*
- *for every edge $e\in E(C)$, a list $L(e)$ of all edges $\hat{e}\in E(W_C)$, whose embedding path $P(\hat{e})\in\mathcal{P}_C$ contains e; and*
- *for every vertex $v\in V(C)$, the number n_v of paths in $\mathcal{P}_C$ containing v.*

Recall that, from the definition of the router certiciate, graph W_C must be obtained from $W_{\hat{k}}^{N_C,\Delta}$ by applying the algorithm $\mathcal{A}_C$ from Theorem 4.8 for the NiceRouterPruning *problem to it, with a single phase Φ_0 of updates, consisting of at most $\frac{N_C^{\hat{k}}\cdot\Delta}{2\hat{k}^{4\hat{k}}}$ edge deletions. We assume that our algorithm is*

also given access to Algorithm $\mathcal{A}_C$, including its inner state, at the end of Phase Φ_0. The algorithm also receives as input k online batches $\pi_1, \ldots, \pi_k$ of edge deletions from C, where, for all $1 \leq i \leq k$, $|\pi_i| \leq \frac{|V(C)| \cdot \Delta}{n^{11/k^2}}$.

The algorithm maintains, at all times, a graph $\hat{C} \subseteq C$ and a graph $\hat{W}_C \subseteq W_C$, that is a properly pruned subgraph of $W_{\hat{k}}^{N_C,\Delta}$. Initially, $\hat{C} = C$ and $\hat{W}_C = W_C$ hold, and the only changes that graphs $\hat{C}$ and $\hat{W}_C$ undergo is the deletion of some of their edges and vertices over time. Throughout, we denote by $\mathcal{P}_C = \left\{P(e) \mid e \in E(\hat{W}_C)\right\}$ the embedding of the current graph $\hat{W}_C$ into C, where the embedding paths $P(e)$ do not change over the course of the algorithm. The algorithm guarantees that, at all times, for every edge $e \in E(\hat{W}_C)$, its embedding path $P(e) \in \mathcal{P}_C$ is contained in the current graph $\hat{C}$; for every edge $e' \in E(\hat{C})$, some path in $\mathcal{P}_C$ contains e; and for every vertex $v \in V(\hat{C})$, at least $\frac{\Delta}{n^{6/k^2}}$ paths in $\mathcal{P}_C$ contain v. Lastly, the algorithm ensures that, for all $1 \leq i \leq k$, at most $|\pi_i| \cdot n^{5/k^2}$ edges are deleted from $\hat{C}$, and at most $|\pi_i| \cdot n^{3/k^2}$ edges are deleted from $\hat{W}_C$ while processing batch π_i. For all $1 \leq i \leq k$, the time required to process batch π_i is at most $O\left(|\pi_i| \cdot n^{O(1/k^2)}\right)$.

The remainder of this subsection is dedicated to the proof of Theorem 8.11. At the beginning of the algorithm, we let $\hat{C} = C$ and $\hat{W}_C = W_C$. Throughout the algorithm, we denote by $\mathcal{P}_C = \left\{P(e) \mid e \in E(\hat{W}_C)\right\}$, where for each edge $e \in E(\hat{W}_C)$, its embedding path $P(e)$ remains unchanged throughout the algorithm. We will ensure that, throughout the algorithm, the following three conditions hold:

C1. for all $1 \leq i \leq k$, if an edge e lies in $\hat{W}_C$ after batch π_i is processed, then its embedding path $P(e) \in \mathcal{P}_C$ is contained in the current graph $\hat{C}$;

C2. for all $1 \leq i \leq k$, if an edge e lies in $\hat{C}$ after batch π_i is processed, then some path in $\mathcal{P}_C$ contains e; and

C3. for all $1 \leq i \leq k$, if a vertex v lies in $\hat{C}$ after batch π_i is processed, then v belongs to at least $\frac{\Delta}{n^{4/k^2} \cdot \left(4k^{16k^2} \cdot d^*\right)^i}$ paths in $\mathcal{P}_C$.

Clearly, Condition C1 holds at the beginning of the algorithm, and, from the definition of a Router Certificate (see Definition 8.2), conditions C2 and C3 hold as well. Recall that, over the course of the algorithm, some vertices and edges may be deleted from $\hat{W}_C$. Whenever an edge e is deleted from $\hat{W}_C$, we delete its embedding path $P(e)$ from $\mathcal{P}_C$. For every edge $e' \in E(P(e))$, we also update the list $L(e')$, by deleting edge e from the list, and, if $L(e')$ becomes empty, we delete e' from C. Additionally, for every vertex $v \in V(P(e))$, we decrease the counter n_v by 1. This ensures that, throughout the algorithm, for every edge $e' \in E(\hat{C})$, list $L(e')$ contains all paths in the current set $\mathcal{P}_C$ to which e' belongs, and for every vertex $v \in V(\hat{C})$, counter n_v is equal to the number of paths in the current set $\mathcal{P}_C$ that contain v. This also ensures that Condition C2 always holds.

We now fix an integer $1 \leq i \leq k$, and describe the processing of the ith batch π_i of edge deletions from $\hat{C}$. Over the course of this algorithm, we will construct a sequence E_i of edges to be deleted from graph $\hat{W}_C$; all these edges serve as an input to Algorithm $\mathcal{A}_C$, as part of Phase Φ_i of the algorithm, and we call all such deletions from $\hat{W}_C$ *direct edge deletions*.

Initially, for every edge $e \in \pi_i$, we add all edges in $L(e)$ to E_i. Since the paths in $\mathcal{P}_C$ cause congestion at most η^*, this initial collection of edge deletions contains at most $|\pi_i| \cdot \eta^*$ edges. Recall that Algorithm $\mathcal{A}_C$ for the NiceRouterPruning problem may also delete some edges from $\hat{W}_C$, that do not lie in E_i. We call such edge deletions *indirect*. It may also delete some vertices from $\hat{W}_C$. Whenever an edge

$e \in E(\hat{W}_C)$ is deleted by Algorithm $\mathcal{A}_C$ indirectly, we inspect the corresponding embedding path $P(e) \in \mathcal{P}_C$. For every edge $e' \in E(P(e))$, we delete e from the list $L(e')$, and, if the list becomes empty, we delete e' from $\hat{C}$. Next, we consider every vertex $v \in V(P(e))$. If e is the first edge that is deleted from $\hat{W}_C$ in this phase, whose embedding path contains v, then we mark v to indicate that some edge whose embedding path contains v was deleted from $\hat{W}_C$ in this phase, and we also record the number n_v of the embedding paths that contained v at the beginning of the phase; this number is denoted by $\lambda_i(v)$. In any case, we decrease n_v by 1, and, if it falls below $\frac{\lambda_i(v)}{4k^{16k^2} \cdot d^*}$, then we delete v and all its incident edges from $\hat{C}$. For every edge $\hat{e}$ that we just deleted from $\hat{C}$, we add all edges of $\hat{W}_C$ that lie in the list $L(\hat{e})$ to set E_i, which are then deleted from $\hat{W}_C$ as direct deletions, and become part of the input sequence of edge deletions for $\mathcal{A}_C$. This completes the description of the algorithm for processing the ith batch π_i of edge deletions. It is immediate to verify that, if Conditions C1–C3 held before batch π_i was processed, then they continue to hold after it is processed. The key in the analysis of the algorithm is the following claim, the bounds the total number of edges that are deleted from $\hat{W}_C$ during the processing algorithm.

Claim 8.12 *The total number of edges deleted from $\hat{W}_C$, either directly or indirectly, while processing batch π_i, is bounded by $|\pi_i| \cdot n^{9/k^2}$.*

Proof: For simplicity, in this proof we denote $\hat{W}_C$ by W. Recall that an edge e is deleted from W *directly*, if it lies in the input sequence E_i of edge deletions, and it is deleted from W *indirectly* otherwise. We denote by $W^{(0)}$ the graph W just before the batch π_i of edge deletions from C is processed. In order to analyze the algorithm, we will maintain, for every edge $e \in E(W^{(0)})$, a *budget*, as follows. Whenever an edge e is deleted from W, for every vertex $v \in V(P(e))$, we initially set the value $\gamma(v, e)$ to be 1, and, if vertex v is ever deleted from $\hat{C}$, value $\gamma(v, e)$ is set to 0.

Whenever an edge e is added to E_i, its budget $\beta(e)$ is set to be $2k^{16k^2} \cdot d^*$, and following that, it may only decrease, but it must always remain at least $1 + \sum_{v \in P(e)} \gamma(v, e)$. The budget $\beta(e)$ of an edge e that was deleted from W indirectly is always set to be $\beta(e) = 1 + \sum_{v \in P(e)} \gamma(v, e)$. So initially, immediately after e is deleted from W, $\beta(e)$ is set to be $|V(P(e))| + 1$, and afterwords it may only decrease, but it always remains at least 1. The budgets of all other edges in $W^{(0)}$ are 0. We denote by $\beta^* = \sum_{e \in E(W^{(0)})} \beta(e)$ the total budget of all edges of $W^{(0)}$.

Recall that at the beginning of the algorithm, we add to E_i all edges that lie in lists $L(e)$ for edges $e \in \pi_i$. Since each such list may contain at most η^* edges, initially, $|E_i| \leq |\pi_i| \cdot \eta^*$ holds. Since the initial budget of every edge in E_i is set to be $2k^{16k^2} \cdot d^*$, we get that initially:

$$\beta^* \leq 2|\pi_i| \cdot k^{16k^2} \cdot d^* \cdot \eta^* \leq |\pi_i| \cdot n^{9/k^2}.$$

(We have used the fact that, from Inequality 12, $2k^{16k^2} \leq 2^{k^6} \leq n^{1/k^4}$ holds, and, From Inequality 15, $\eta^* \cdot d^* \leq n^{5/k^2}$.)

Observe that, throughout the algorithm, the budget of every edge in $E(W^{(0)}) \setminus E(W)$ is at least 1, and so the number of edges deleted from W is bounded by the final budget β^*. Therefore, it is now enough to show that we can maintain the budgets of all edges of $W^{(0)}$, so that they obey the above rules, and the total budget β^* never increases.

Recall that, when an edge e is added to set E_i, we set its initial budget to be $2k^{16k^2} \cdot d^*$. The budgets of the edges in E_i change over the course of the algorithm as follows. Whenever the algorithm $\mathcal{A}_C$ chooses to delete some edge e' from W indirectly, its initial budget is set to $1 + |V(P(e'))| \leq 2 + d^*$. In order to ensure that the total budget β^* does not grow, we will *move* the budget from some edges in E_i to edge e'. In other words, we will reduce the total budget of the edges in the current set E_i by (additive)

d^*+2, while ensuring that the budget of every edge in E_i remains at least $1+\sum_{v\in P(e)}\gamma(v,e)$. Recall that Algorithm $\mathcal{A}_C$ for the NiceRouterPruning problem must ensure that the total number of edges deleted from W over the course of phase Φ_i is bounded by $|E_i|\cdot k^{8k^2}$ (recall that we use the algorithm with parameter k replaced by k^2). Since the algorithm does not know when Phase Φ_i is going to end, it must guarantee that, at any time τ during the phase, the total number of edges deleted from W so far is bounded by $|E_i^{(\tau)}|\cdot k^{8k^2}$, where $E_i^{(\tau)}$ is the set of all edges that were added to E_i by time τ. Since, whenever an edge is added to E_i, its initial budget is $2k^{16k^2}\cdot d^*$, while the initial budget of an edge that is deleted from W indirectly is set to be at most $2+d^*$, it is easy to verify that, whenever a new edge e is deleted from W indirectly, we can move $1+|V(P(e))|$ units of budget from the edges that are currently in E_i to e, so that the budget of every edge in E_i remains at least 1, and the total budget β^* does not increase.

Consider now some time τ during the algorithm's execution, where a vertex v is deleted from C. Recall that we have denoted by $\lambda_i(v)$ the initial number n_v of paths $P(e)\in\mathcal{P}_C$ that contained v before batch π_i was processed. Let $\hat{E}(v)\subseteq E(W^{(0)})$ be the set of all edges $e\in E(W^{(0)})$, whose embedding path $P(e)$ contained v, so that $|\hat{E}(v)|=\lambda_i(v)$. Let $\hat{E}'(v)\subseteq E(W)$ be the set of edges e that remain in graph W at time τ, whose embedding path $P(e)$ contains v. Since the algorithm decided to delete vertex v from C at time τ, we get that $|\hat{E}'(v)|\leq\frac{|\hat{E}(v)|}{4k^{16k^2}\cdot d^*}$.

Note that, before vertex v is deleted from C, for every edge $e\in\hat{E}(v)\setminus\hat{E}'(v)$, $\gamma(v,e)=1$ holds, and immediately after v is deleted from C, $\gamma(v,e)=0$. Therefore, following the deletion of v from C, the budget $\beta(e)$ of every edge $e\in\hat{E}(v)\setminus\hat{E}'(v)$ decreases by 1, and the total decrease in the budgets of all such edges is at least $|\hat{E}(v)\setminus\hat{E}'(v)|\geq\frac{\lambda_i}{2}$. For every edge $e\in\hat{E}'(v)$, following the deletion of v from C, its budget $\beta(e)$ grows from 0 to $2k^{16k^2}\cdot d^*$. But since $|\hat{E}'(v)|\leq\frac{\lambda_i}{4k^{16k^2}\cdot d^*}$, the total budget β^* does not grow. □

Recall that the algorithm $\mathcal{A}_C$ for the NiceRouterPruning problem requires that the total number of edges deleted from $\hat{W}_C$ directly during phase Φ_i is bounded by $\frac{N_C^{k^2}\cdot\Delta}{k^{16k^2}}$. Recall that, from the definition of the router certificate (see Definition 8.2), $N_C\geq\frac{|V(C)|^{1/k^2}}{2|V(C)|^{1/k^4}}\geq\frac{|V(C)|^{1/k^2}}{2n^{1/k^4}}$ holds. Since $|\pi_i|\leq\frac{|V(C)|\cdot\Delta}{n^{11/k^2}}$, from Claim 8.12, we get that:

$$\begin{aligned}|E_i|&\leq|\pi_i|\cdot n^{9/k^2}\\&\leq\frac{|V(C)|\cdot\Delta}{n^{2/k^2}}\\&\leq\frac{|V(C)|}{2^{k^2}\cdot n^{1/k^2}}\cdot\frac{\Delta}{k^{16k^2}}\\&\leq\frac{N_C^{k^2}\cdot\Delta}{k^{16k^2}},\end{aligned}$$

as required (we have used the fact that, from Inequality 12, $n^{1/k^2}\geq k^{16k^2}\cdot 2^{k^2}$).

In the next simple corollary of Claim 8.12, we bound the total number of edges deleted from $\hat{C}$ when processing the batch π_i, for all $1\leq i\leq k$.

Corollary 8.13 *The total number of edges deleted from $\hat{C}$ when processing batch π_i of edge deletions is bounded by $|\pi_i|\cdot n^{10/k^2}$.*

Proof: Let $\hat{E}$ be the set of all edges deleted from $\hat{W}_C$ while processing batch π_i. From Claim 8.12, $|\hat{E}|\leq|\pi_i|\cdot n^{9/k^2}$.

Let E' be the set of all edges that were deleted from $\hat{C}$ while processing batch π_i. Recall that, from Condition C2, before the algorithm started processing batch π_i, for every edge $e \in E(\hat{C})$, some path in $\mathcal{P}_C$ contained e. Since Condition C1 holds when the algorithm finishes processing π_i, we get that, for every edge $e \in E'$, there is an edge $e' \in \hat{E}$ with $e \in P(e')$. Since the length of each embedding path in $\mathcal{P}_C$ is at most $d^* \leq n^{1/k^6}$ from Inequality 14, it is easy to see that:

$$|E'| \leq d^* \cdot |\hat{E}| \leq |\pi_i| \cdot n^{10/k^2}.$$

□

Recall that, from Property C3, if a vertex v lies in $\hat{C}$ at any time τ, then, at time τ, the number of paths in $\mathcal{P}_C$ that contain v is at least:

$$\begin{aligned}\frac{\Delta}{n^{4/k^2} \cdot \left(4k^{16k^2} \cdot d^*\right)^k} &\geq \frac{\Delta}{n^{4/k^2} \cdot 2^{2ck^7}} \\ &\geq \frac{\Delta}{n^{6/k^2}}.\end{aligned}$$

since $d^* = k^{10} \cdot 2^{c/\delta^6} \leq k^{10} \cdot 2^{ck^6}$, as $\delta \geq 1/k$; we also used Inequality 12.

It now remains to bound the time required to process a batch π_i. The running time of the algorithm $\mathcal{A}_C$ from Theorem 4.8 for the NiceRouterPruning problem during phase Φ_i is bounded by:

$$O\left(|E_i| \cdot k^{O(k)}\right) \leq O\left(|\pi_i| \cdot n^{O(1/k^2)}\right),$$

from Claim 8.12 and Inequality 12. Additionally, for every edge e that is deleted from $\hat{W}_C$, we need to spend time $O(|E(P(e))|) \leq O(d^*)$ in order to process all edges and vertices on path $P(e)$. Since, from Claim 8.12, the total number of edges deleted from $\hat{W}_C$ while processing batch π_i is bounded by $|\pi_i| \cdot n^{9/k^2}$, and since, from Inequality 14, $d^* \leq n^{1/k^6}$, the total time required to process all such paths is bounded by $O\left(|\pi_i| \cdot n^{O(1/k^2)}\right)$. Therefore, the total time the algorithm spends to process batch π_i is bounded by $O\left(|\pi_i| \cdot n^{O(1/k^2)}\right)$.

8.1.3 Part 1: Summary

The following corollary easily follows from Theorem 8.3 and Theorem 8.11, and it summarizes our algorithm for maintaining the clustering $\mathcal{C}$, and, for every cluster $C \in \mathcal{C}$, the corresponding graph W_C, along with its embedding into C.

Corollary 8.14 *There is a deterministic algorithm, that receives as input a valid input graph G together with valid input parameters n, k, δ, Δ and Δ^*, such that, for every vertex $v \in V(G)$, $\deg_G(v) \leq \Delta \cdot n^{16/k}$. The algorithm also receives k online batches $\pi_1, \ldots, \pi_k$ of updates to graph G, where for $1 \leq i \leq k$, the ith batch π_i is a collection of at most $\Delta \cdot n$ edges that are deleted from G.*

The algorithm maintains a collection $\mathcal{C}$ of subgraphs of G called clusters*, with $\sum_{C \in \mathcal{C}} |V(C)| \leq n^{1+20/k}$, such that every cluster $C \in \mathcal{C}$ is a $\left(\overline{\deg}_C, \tilde{d}, \tilde{\eta}\right)$-router with respect to the set $V(C)$ of supported vertices, for $\tilde{d} \leq 2k^{18} \cdot 2^{c/\delta^6}$ and $\tilde{\eta} \leq n^{17/k}$, and every edge of G lies in at most one cluster $C \in \mathcal{C}$. The algorithm also maintains, for every cluster $C \in \mathcal{C}$, a graph W_C, that is a properly pruned subgraph of $W_{k^2}^{N_C, \Delta}$, for some parameter N_C, and an embedding $\mathcal{P}_C$ of W_C into C via paths of length at most d^* that cause*

congestion at most η^*, such that every vertex of C belongs to at least $\frac{\Delta}{n^{6/k^2}}$ of the paths in $\mathcal{P}_C$ at all times. After the clustering $\mathcal{C}$ is initialized, for every cluster $C \in \mathcal{C}$, the only changes that C and W_C may undergo are the deletions of some of their edges and vertices; for every edge e that remains in W_C, its embedding path $P(e) \in \mathcal{P}_C$ cannot change after the initialization. Moreover, for all $1 \leq i \leq k$, if we denote by $\pi_i^C = \pi_i \cap E(C)$, then the number of edges deleted from C while processing the batch π_i is at most $|\pi_i^C| \cdot n^{13/k^2}$. The only additional change that the clustering $\mathcal{C}$ may undergo after its initializaition is the deletion of empty clusters from $\mathcal{C}$.

Throughout, we denote by $E^{\text{del}} = E(G) \setminus \left(\bigcup_{C \in \mathcal{C}} E(C)\right)$. At the beginning of the algorithm, after the initialiation, $|E^{\text{del}}| \leq |V(G)| \cdot \Delta \cdot n^{2/k^2}$ must hold, and after that edges may only be added to E^{del}. For all $1 \leq i \leq k$, the number of edges added to E^{del} while processing batch π_i is bounded by $|\pi_i| \cdot n^{13/k^2}$.

The initialization time of the algorithm is $O\left((n\Delta)^{1+O(\delta)}\right)$, and for all $1 \leq i \leq k$, the time required to process the ith update batch π_i is bounded by $O\left(|\pi_i| \cdot n^{O(1/k^2)}\right)$.

Proof: We start by employing the algorithm from Theorem 8.3 to compute an initial collection $\mathcal{C}$ of clusters of G, that are disjoint in their edges, such that $\sum_{C \in \mathcal{C}} |V(C)| \leq n^{1+20/k}$. Recall that, for every cluster $C \in \mathcal{C}$, the algorithm also computes a router certificate $\left(W_{k^2}^{N_C,\Delta}, W_C, \mathcal{P}_C\right)$, where graph W_C is obtained from $W_{k^2}^{N_C,\Delta}$ by applying the algorithm from Theorem 4.8 for the NiceRouterPruning problem to it, with a single phase Φ_0 of updates, consisting of at most $\frac{N_C^{\hat{k}} \cdot \Delta}{2\hat{k}^{4\hat{k}}}$ edge deletions; we denote this algorithm for $\mathcal{A}_C$. The algorithm from Theorem 8.3 also ensures that the cardinality of the set $E^{\text{del}} = E(G) \setminus \left(\bigcup_{C \in \mathcal{C}} E(C)\right)$ of edges is at most $|V(G)| \cdot \Delta \cdot n^{2/k^2}$, and its running time is $O\left((n\Delta)^{1+O(\delta)}\right)$. We also need the following simple observation.

Observation 8.15 *For all $C \in \mathcal{C}$, $|E(C)| \leq |V(C)| \cdot \Delta \cdot n^{2/k^2}$ holds.*

Proof: Consider the router certificate $\left(W_{k^2}^{N_C,\Delta}, W_C, \mathcal{P}_C\right)$ for C. Clearly, $|V(C)| \geq |V(W_C)|$ must hold. From the definition of the graph $W_{k^2}^{N_C,\Delta}$, for every leaf vertex $v \in W_C$, the degree of v in W_C is at most $k^2 \cdot \Delta$, and every edge of W_C is incident to at least one leaf vertex. Therefore, $|E(W_C)| \leq |V(W_C)| \cdot k^2 \cdot \Delta \leq |V(C)| \cdot k^2 \cdot \Delta$.

Recall that, from the definition of a router certificate, every edge of C must lie on some path in $\mathcal{P}_C$, and moreover, the length of every path in $\mathcal{P}_C$ is at most d^*. Therefore:

$$|E(C)| \leq d^* \cdot |\mathcal{P}_C| = d^* \cdot |E(W_C)| \leq |V(C)| \cdot d^* \cdot k^2 \cdot \Delta \leq |V(C)| \cdot \Delta \cdot n^{2/k^2}$$

from Inequalies 14 and 13. □

Consider now some cluster $C \in \mathcal{C}$, and the router certificate $\left(W_{k^2}^{N_C,\Delta}, W_C, \mathcal{P}_C\right)$ that we have computed for C. For every edge $e \in E(C)$, we compute a list $L(e)$ of all edges $\hat{e} \in E(W_C)$, whose embedding path $P(\hat{e}) \in \mathcal{P}_C$ contains e. Additionally, for every vertex $v \in V(C)$, we the number n_v of paths in $\mathcal{P}_C$ containing v. The time required to compute all such lists $L(e)$ for $e \in E(C)$ and integers n_v for $v \in V(C)$, for all clusters $C \in \mathcal{C}$, is asymptotically bounded by the time required to compute the router certificates $\left(W_{k^2}^{N_C,\Delta}, W_C, \mathcal{P}_C\right)$ for all clusters, which is in turn asymptotically bounded by running time of the algorithm from Theorem 8.3.

For every cluster $C \in \mathcal{C}$, we initialize the algorithm from Theorem 8.11 for C, that we denote by $\mathcal{A}'_C$. The algorithm receives as input the router certificate $\left(W_{k^2}^{N_C,\Delta}, W_C, \mathcal{P}_C\right)$, the lists $\{L(e)\}_{e \in E(C)}$, and

the values $\{n_v \mid v \in V(C)\}$ that we have computed. This finishes the description of the initialization algorithm. We will later prove that the every cluster $C \in \mathcal{C}$ is a $\left(\overline{\deg}_C, \tilde{d}, \tilde{\eta}\right)$-router. Next, we provide an algorithm for processing the update batches $\pi_1, \ldots, \pi_k$.

Consider some index $1 \leq i \leq k$, and the batch π_i of updates. For every cluster $C \in \mathcal{C}$, let $\pi_i^C = \pi_i \cap E(C)$. Assume first that $|\pi_i^C| > \frac{|V(C)| \cdot \Delta}{n^{11/k^2}}$. Using the same reasoning as in Observation 8.15, it is easy to see that $|E(C)| \leq |V(C)| \cdot \Delta \cdot n^{2/k^2} \leq |\pi_i^C| \cdot n^{13/k^2}$. We then delete all edges and all vertices from C, and delete C from $\mathcal{C}$. Assume now that $|\pi_i^C| \leq \frac{|V(C)| \cdot \Delta}{n^{11/k^2}}$. We then provide the batch π_i^C, as an input batch of edge deletions, to Algorithm $\mathcal{A}'_C$. Recall that the algorithm deletes at most $|\pi_i^C| \cdot n^{10/k^2}$ edges, and possibly some vertices, from cluster C. It may also delete some edges and vertices from graph W_C. The algorithm ensures that W_C remains a properly pruned subgraph of $W_{\hat{k}}^{N_C,\Delta}$; that every edge $e \in E(C)$ lies on some embedding path in $\{P(e') \mid e' \in E(W_C)\}$, and every vertex $v \in V(C)$ lies on at least $\frac{\Delta}{n^{6/k^2}}$ such paths. The algorithm also ensures that every embedding path in $\{P(e') \mid e' \in E(W_C)\}$ is contained in the current cluster C. The running time required to process batch π_i^C is bounded by $O\left(|\pi_i^C| \cdot n^{O(1/k^2)}\right)$.

This completes the description of the algorithm for processing the batch π_i of edge deletions from G. All edges that have been deleted from the clusters in $\mathcal{C}$ while processing π_i are added to set E^{del}. From our discussion, for every cluster $C \in \mathcal{C}$, the number of edges deleted from C while processing π_i is bounded by $|\pi_i^C| \cdot n^{13/k^2}$, and so the total number of edges added to E^{del} is bounded by $\sum_{C \in \mathcal{C}} |\pi_i^C| \cdot n^{13/k^2} \leq |\pi_i| \cdot n^{13/k^2}$. The time that is required to process the batch π_i is bounded by:

$$\sum_{C \in \mathcal{C}} O\left(|\pi_i^C| \cdot n^{O(1/k^2)}\right) \leq O\left(|\pi_i| \cdot n^{O(1/k^2)}\right).$$

Finally, we show that, throughout the algorithm, every cluster $C \in \mathcal{C}$ remains a $\left(\overline{\deg}_C, \tilde{d}, \tilde{\eta}\right)$-router with respect to the set $V(C)$ of supported vertices, for $\tilde{d} \leq 2k^{18} \cdot 2^{c/\delta^6}$ and $\tilde{\eta} \leq n^{17/k}$.

Consider some cluster C, and any time τ at which C lied in $\mathcal{C}$, so $C \neq \emptyset$ holds. Then our algorithm maintains a graph W_C, that is a properly pruned subgraph of $W_{\hat{k}}^{N_C,\Delta}$, together with an embedding $\mathcal{P}_C$ of W_C into C via paths of length at most d^* that cause congestion at most η^*, such that every vertex $v \in V(C)$ lies on at least $\frac{\Delta}{n^{6/k^2}}$ paths in $\mathcal{P}_C$. Recall that the maximum vertex degree in C is bounded by $\Delta \cdot n^{16/k}$. By using Lemma 6.1 with parameters $\alpha = n^{16/k}$, $\beta = n^{6/k^2}$, and parameter k replaced with k^2, we get that C is a $\left(\overline{\deg}_C, \tilde{d}, \tilde{\eta}\right)$-router with respect to the set $V(C)$ of supported vertices, for $\tilde{d} = 22 \cdot d^* \cdot k^8 \leq 2k^{18} \cdot 2^{c/\delta^6}$, and $\tilde{\eta} = 2n^{16/k} \cdot n^{6/k^2} \cdot k^{8k^2+1} \cdot d^* \cdot \eta^* \leq n^{17/k}$, since $d^* = k^{10} \cdot 2^{c/\delta^6}$; we have also used Inequalities 12 and 15. □

8.2 Part 2 of the Algorithm: Maintaining the Subgraphs $C' \subseteq C$

In order to complete the proof of Theorem 3.2, it is enough to provide an algorithm that maintains, for every cluster $C \in \mathcal{C}$, a subgraph $C' \subseteq C$ with $V(C') = V(C)$ and $|E(C')| \leq \Delta^* \cdot |V(C)|$, such that C' is a $\left(\overline{\Delta}^*, \tilde{d}, \tilde{\eta}\right)$-router for the set $V(C)$ of supported edges.

Throughout, we use the parameter $\Delta' = \left\lfloor \frac{\Delta^*}{2k^2 d^*} \right\rfloor$. Note that, from the definition of valid input parameters, and Inequalities 12 and 14, $\Delta^* \geq n^{18/k} > 4k^2 d^*$ holds. Recall that the algorithm from Corollary 8.14 maintains, for every cluster $C \in \mathcal{C}$, a graph W_C, that is a properly pruned subgraph of $W_{k^2}^{N_C,\Delta}$, for some parameter N_C, and an embedding $\mathcal{P}_C$ of W_C into C via paths of length at most d^*

that cause congestion at most η^*, such that every vertex of C belongs to at least $\frac{\Delta}{n^{6/k^2}}$ of the paths in $\mathcal{P}_C$ at all times. Notice that:

$$\Delta' = \left\lfloor \frac{\Delta^*}{2k^2 d^*} \right\rfloor \leq \frac{\Delta}{n^{8/k^2}} \leq \frac{\Delta}{2^k \cdot n^{6/k^2}},$$

from the definition of valid input parameters, and since $2^k \leq n^{1/k^2}$ from Inequality 12.

Consider now some cluster $C \in \mathcal{C}$. For every vertex $v \in V(C)$, let $\mathcal{P}_C(v) \subseteq \mathcal{P}_C$ denote the set of all embedding paths that contain v. Consider now some path $P \in \mathcal{P}_C(v)$, and recall that P must be an embedding path of some edge $e \in E(W_C)$, so $P = P(e)$. Recall that one of the endpoints of e must be a leaf vertex of W_C; we call it the *distinguished endpoint of* P. We denote by $U_C(v)$ the multiset of leaf vertices of W_C that serve as distiguished endpoints of the paths in $\mathcal{P}_C(v)$. Lastly, consider the bipartite graph $R_C = (A, B, \tilde{E})$, where $A = V(C)$, and B contains all leaf vertices of W_C (so a vertex $v \in V(C)$ may lie in both A and B; we will refer to these vertices as "the copy of v in A" and "the copy of v in B"). For every vertex $v \in V(C)$, for each vertex u multiset $U_C(v)$, we include an edge connecting a copy of v in A to a copy of u in B in $\tilde{E}$. Recall that, for every vertex $u \in U_C(v)$, there is a distinct path $P \in \mathcal{P}_C(v)$, such that u is the distinguished endpoint of P. We will say that the new edge *represents* the path P.

Additional Data Structures. We fix again a cluster $C \in \mathcal{C}$. Throughout the algorithm, for every vertex $v \in V(C)$, we maintain the set $U_C(v)$ of leaf vertices of W_C explicitly. Additionally, we maintain the set $\mathcal{P}_C(v)$ of paths implicitly, as follows: for every vertex $u \in U_C(v)$, we maintain a pointer from the copy of u in $U_C(v)$ to the edge $e \in E(W_C)$, such that u serves as the distinguished endpoint of the path $P(e) \in \mathcal{P}_C(v)$ (if several copies of u lie in $U_C(v)$, then we associate each copy with a distinct path in $\mathcal{P}_C(v)$, and maintain a pointer from each copy of u in $U_C(v)$ to the corresponding edge of W_C). We also explicitly maintain the bipartite graph R_C.

Recall that our algorithm from Part 1 maintains, for every large cluster $C \in \mathcal{C}$ and vertex $v \in V(C)$, the counter $n_v = |\mathcal{P}_C(v)|$. The counter is maintained in a straightforward manner: during the initialization step, for every edge $e \in E(W_C)$, we inspect all vertices lying on path $P(e)$ and update their counters n_v. At this time, we can also construct the sets $U_C(v)$ of vertices for all $v \in V(C)$, together with the pointers from every vertex $u \in U_C(v)$ to the corresponding edge of W_C, without increasing the asymptotic running time of the initialization algorithm. We can also initialize the graph R_C within this running time.

During the update steps, whenever an edge e is deleted from graph W_C, we inspect every vertex $v \in V(P(e))$, and we decrease the corresponding counter n_v. During this time, we can also update the list $U_C(v)$ for each such vertex v with the deletion of this edge, and delete the edge connecting v to the distinguished endpoint of $P(e)$ from graph R_C. All this can be done without increasing the asymptotic running time of the algorithm for processing each batch π_i^C of updates to cluster C.

Next, we summarize some properties of the graph R_C, in the following simple observation.

Observation 8.16 *Upon the initialization of graph $R_C = (A, B, \tilde{E})$, every vertex $x \in B$ has degree at most $d^* \cdot k^2 \cdot \Delta \leq n^{2/k^2} \cdot \Delta$ in R_C. For all $1 \leq i \leq k$, after batch π_i of edge deletions is processed by the algorithm from Part 1, at most $|\pi_i^C| \cdot n^{10/k^2}$ edges are deleted from R_C, and all vertices that become isolated are deleted from R_C as well. These are the only updates to graph R_C. Moreover, for every vertex $v \in A$, the degree of v in R_C remains at least $\frac{\Delta}{n^{6/k^2}}$ at all times.*

Proof: Recall that, from the definition of graph $W_{k^2}^{N_C,\Delta}$, every leaf vertex of the graph has degree

at most $k^2 \cdot \Delta$. Consider now any leaf vertex $u \in B$, and let $E_u \subseteq E(W_C)$ be the set of edges that are incident to u in W_C after the initialiation step, so $|E_u| \leq k^2 \cdot \Delta$. For every edge $e \in E_u$, path $P(e) \in \mathcal{P}_C$ contains at most d^* vertices, and vertex x lies in the set $U_C(v)$ of each such vertex v. Therefore, the degree of x in R_C is bounded by $d^* \cdot k^2 \cdot \Delta \leq n^{2/k^2} \cdot \Delta$ (from Inequality 14).

Consider now some integer $1 \leq i \leq k$. From Theorem 8.11, while processing the batch π_i of updates, the algorithm from Part 1 may delete at most $|\pi_i^C| \cdot n^{9/k^2}$ edges from W_C. Since the embedding path of each such edge has length at most d^*, this can lead to the deletion of at most $|\pi_i^C| \cdot n^{9/k^2} \cdot d^* \leq |\pi_i^C| \cdot n^{9/k^2}$ edges from R_C (from Inequality 14). Since every vertex that remains in C must participate in at least $\frac{\Delta}{n^{10/k^2}}$ paths in $\mathcal{P}_C$, we get that the degree of every vertex in A remains at least $\frac{\Delta}{n^{6/k^2}}$ throughout the algorithm. □

The main result of the current subsection is summarized in the following claim.

Claim 8.17 *There is a deterministic algorithm, that, given an access to the adjacency-list representation of the graph R_C, maintains, for every vertex $v \in V(C)$, a subset $U'_C(v) \subseteq U_C(v)$ of vertices, whose cardinality is between Δ' and $\Delta' \cdot n^{11/k^2}$ at all times, such that, for every leaf vertex x of W_C, the total number of times that x appears in the sets $\{U'_C(v) \mid v \in V(C)\}$ is at most $\gamma \cdot \Delta'$, where $\gamma = n^{16/k}$. After processing all updates to graph R_C following each batch π_i^C, the algorithm outputs all vertices that were deleted from or inserted into the sets $\{U'_C(v)\}_{v \in V(C)}$. The initialization time of the algorithm is $O(|V(C)| \cdot \Delta \cdot n^{O(1/k)})$, and, for all $1 \leq i \leq k$, the time required for updates following the processing of batch π_i^C of edge deletions to graph C is bounded by $O\left(|\pi_i^C| \cdot n^{O(1/k^2)}\right)$.*

We provide the proof of Claim 8.17 below, after we comlete the proof of Theorem 3.2 using it.

We now fix a cluster $C \in \mathcal{C}$, and provide an algorithm for maintaining the corresponding graph $C' \subseteq C$. We first define the graph C' and prove that it must be a $\left(\overline{\Delta}^*, \tilde{d}, \tilde{\eta}\right)$-router for the set $V(C)$ of supported edges, and that $|E(C')| \leq \Delta^* \cdot |V(C)|$. Later, we provide an algorithm for maintaining C'.

Definition of Graph C'. Recall that the algorithm from Claim 8.17 maintains, for every vertex $v \in V(C)$, a subset $U'_C(v) \subseteq U_C(v)$ of vertices, whose cardinality is between Δ' and $\Delta' \cdot n^{11/k^2}$ at all times. We let $U''_C(v) \subseteq U'_C(v)$ be an arbitray subset of Δ' such vertices, and we let $\mathcal{Q}_C(v) \subseteq \mathcal{P}_C(v)$ be the set of paths that correspond to these vertices, so that $|\mathcal{Q}_C(v)| = \Delta'$. Let $\mathcal{Q}_C = \bigcup_{v \in V(C)} \mathcal{Q}_C(v)$. Notice that, from Claim 8.17, every leaf vertex of W_C serves as an endpoint in at most $\gamma \cdot \Delta'$ paths in $\mathcal{Q}_C$. Additionally, for every superedge $e = (a, b)$ of W_C, we select an arbitrary collection $E'(a, b)$ of Δ' parallel edges (a, b) (since W_C remains a properly pruned subgraph of $W_{k^2}^{N_C, \Delta}$, the number of parallel edges (a, b) must be at least $\left\lceil \frac{\Delta}{2} \right\rceil \geq \Delta'$). We let $\mathcal{Q}'_C$ be the set of paths that contains, for every superedge (a, b) of W_C, for every edge $e \in E'(a, b)$, the embedding path $P(e) \in \mathcal{P}_C$. We let C'' be the multigraph, obtained by taking the union of all paths in set $\mathcal{Q}_C \cup \mathcal{Q}'_C$, and we let $C' \subseteq C$ be the corresponding simple graph, obtained from C'' by deleting parallel edges. Recall that, from the definition of valid input parameters, $\Delta^* \geq n^{18/k} \geq 2k^2 d^*$ (from Inequalities 13 and 14) must hold. From Observation 6.2, we conclude that $|E(C')| \leq |V(C)| \cdot \Delta^*$, and that graph C' is a $(\overline{\Delta}^*, \tilde{d}, \tilde{\eta}')$-router with respect to the set $V(C')$ of supported vertices, where:

$$\tilde{d} = 22d^* \cdot k^8 \leq k^{19} \cdot 2^{c/\delta^6},$$

since $d^* = k^{10} \cdot 2^{c/\delta^6}$; and

$$\tilde{\eta} = 8\gamma (d^*)^2 \cdot \eta^* \cdot k^{4k+1} \leq n^{17/k},$$

since $\gamma = n^{16/k}$, $\eta^* \cdot d^* \leq n^{5/k^2}$ from Inequality 15; and $k^{6k} \leq n^{1/k^2}$ from Inequality 12. Since $\sum_{C\in\mathcal{C}} |V(C)| \leq n^{1+20/k}$, we get that $\sum_{C\in\mathcal{C}} |E(C')| \leq \sum_{C\in\mathcal{C}} |V(C)| \cdot \Delta^* \leq n^{1+20/k} \cdot \Delta^*$, as required. It is now enough to provide an algorithm for maintaining the graph C'.

Maintaining the Graph C'. Recall that the algorithm from Claim 8.17 maintains, for every vertex $v \in V(C)$, a subset $U'_C(v) \subseteq U_C(v)$ of vertices, whose cardinality is between Δ' and $\Delta' \cdot n^{11/k^2}$ at all times, such that, for every leaf vertex x of W_C, the total number of times that x appears in the sets $\{U'_C(v) \mid v \in V(C)\}$ is at most $\gamma \cdot \Delta'$. Once the subset $U'_C(v)$ is initialized, the algorithm may modify it arbitrarily, that is, vertices may be both added to, and deleted from the set $U'_C(v)$. We assume w.l.o.g. that the algorithm maintains an ordered list $L_C(v)$ of all vertices in $U'_C(v)$, so that, when a new vertex is added to $U'_C(v)$, it is added to the end of the list. Throughout the algorithm, we then let $U''_C(v)$ be the collection of the first Δ' vertices that lie in list $L_C(v)$. As before, we denote by $\mathcal{Q}_C(v) \subseteq \mathcal{P}_C(v)$ all paths corresponding to the vertices of $U''_C(v)$, and we denote by $\mathcal{Q}_C = \bigcup_{v\in V(C)} \mathcal{Q}_C(v)$. For every superedge (u, v) of W_C, we also assume that the parallel edges of W_C corresponding to (u, v) are stored in an ordered list $L'_C(u, v)$. Recall that, as the algorithm progresses, edges may only be deleted from $L'_C(u, v)$. Throughout the algorithm, we will let $E'(u, v)$ be the set of the first Δ' edges in list $L'_C(u, v)$. As before, we denote by $\mathcal{Q}'_C$ the set of paths that contains, for each superedge (a, b) of W_C, for every edge $e \in E'(a, b)$, the embedding path $\mathcal{P}(e) \in \mathcal{P}_C$. As before, multigraph graph C'' is obtained by taking the union of all paths in $\mathcal{Q}_C \cup \mathcal{Q}'_C$ while keeping parallel edges, and graph C' is the simple graph obtained from C'' by removing parallel edges. We define the multiset $U^*_C = \bigcup_{v\in V(C)} U''_C(v)$, and by E^*_C the union of the sets $E'(a, b)$ of edges of W_C, for all superedge (a, b) of W_C.

At the beginning of the algorithm, once the data structures from Part 1 and from Claim 8.17 are initialized, we can compute, for every cluster $C \in \mathcal{C}$, the initial sets $U''_C(v)$ of vertices for all $v \in V(C)$, and sets $E'(a, b)$ of edges for each superedge (a, b) of the initial graph W_C, in time that is asymptotically bounded by the time required to initialize the data structures from Part 1 (that include graph W_C and its embedding), and the data structures from Claim 8.17 (that include sets $U'_C(v)$ of vertices for all $v \in V(C)$). The time required for this computation is then bounded by:

$$T = O\left((n\Delta)^{1+O(\delta)} + \sum_{C\in\mathcal{C}} |V(C)| \cdot \Delta \cdot n^{O(1/k)}\right) \leq O\left((n\Delta)^{1+O(\delta+1/k)}\right) \leq O\left((n\Delta)^{1+O(\delta)}\right),$$

since $\sum_{C\in\mathcal{C}} |V(C)| \leq |V(G)|^{1+20/k}$ must hold, and $\delta \geq 1/k$. Since, for every cluster $C \in \mathcal{C}$, the length of every path in $\mathcal{P}_C$ is bounded by $d^* \leq n^{1/k^6}$ (from Inequality 14), the time required to compute the graphs C'' and C', for all $C \in \mathcal{C}$ is then bounded by:

$$O\left(d^* \cdot (|U^*(C)| + |E^*_C|)\right) \leq O\left(T \cdot d^*\right) \leq O\left((n\Delta)^{1+O(\delta+1/k)}\right) \leq O\left((n\Delta)^{1+O(\delta)}\right).$$

Therefore, the initialiation time of both Part 1 and Part 2 of our algorithm is bounded by $O\left((n\Delta)^{1+O(\delta)}\right)$.

Consider now some integer $1 \leq i \leq k$, and the batch π_i of edge deletions. Once the algorithm from Part 1 finishes processing the batch π_i, for some clusters $C \in \mathcal{C}$, some of the edges may be deleted from graph W_C. If (a, b) is a superedge of graph W_C, and if the number of parallel edges (a, b) deleted from W_C that lie in set $E'(a, b)$ is $m_{a,b}$, then we add $m_{a,b}$ new edges from the list $L'_C(a, b)$, to set $E'(a, b)$, so that $E'(a, b)$ contains the first Δ' edges in list $L'_C(a, b)$. For each edge e of the original set $E'(a, b)$ that was deleted from W_C, for every edge $e' = (x, y) \in E(P(e))$, we delete e' from C'', and, if no edges (x, y) remain in C'', then we delete edge (x, y) from C' as well. Next, for every edge e that was just added to $E'(a, b)$, we consider every edge $e' = (x, y) \in P(e)$. We add e' to graph C'', and,

if no other edges (x,y) currently lie in C'', then we add edge (x,y) to C' as well. It is easy to verify that, if T_i is the time that the algorithm from Part 1 spends on processing batch π_i, then all the above updates, for all clusters $C \in \mathcal{C}$, can be done in time $O(T_i \cdot d^*) \leq O(T_i \cdot n^{2/k^2})$.

Consider now a cluster $C \in \mathcal{C}$, and denote by $\pi_i^C = E(C) \cap \pi_i$. Once the algorithm from Claim 8.17 processes the batch π_i^C, we consider every vertex $v \in V(C)$, for which at least one vertex that lies in $U_C''(v)$ was deleted from $U_C'(v)$. Let $n_C(v)$ be the number of such vertices. First, for every vertex $u \in U_C''(v)$ that was deleted from $U_C'(v)$, we delete u from $U_C''(v)$ as well. We then consider the path Q of $\mathcal{Q}_C(v)$ that corresponds to vertex u. For every edge $e' = (x,y)$ of Q, we delete e' from C'', and, if no other edges (x,y) remain in C'', then we delete edge (x,y) from C' as well. Next, we add $n_C(v)$ vertices from list $L_C(v)$ to $U_C''(v)$, so that $U_C''(v)$ contains the first Δ' vertices from list $L_C(v)$. For each such newly added vertex u, we consider the path Q of $\mathcal{Q}_C(v)$ corresponding to u. For every edge $e' = (x,y)$ of Q, we add e' to C'', and, if no other edges (x,y) lie in C'', then we add edge (x,y) to C' as well. Note that the time required to perform these updates a cluster $C \in \mathcal{C}$ is bounded by:

$$O\left(\sum_{v \in V(C)} n_C(v) \cdot d^*\right).$$

For every cluster $C \in \mathcal{C}$, the algoritm from Claim 8.17 must spend at least $\Omega\left(\sum_{v \in V(C)} n_C(v)\right)$ time on updating the sets $U_C'(v)$ of vertices for all $v \in V(C)$. Therefore, if T_i^C denotes the time that the algorithm from Claim 8.17 spends on processing the batch π_i^C of updates, then the time required to update the sets $U_C''(v)$ of vertices for all $v \in V(C)$, and to update the graph C' accordingly, is bounded by $O\left(T_i^C \cdot d^*\right) \leq O\left(T_i^C \cdot n^{2/k^2}\right)$, from Inequality 14. Since, from Claim 8.17, for every cluster $C \in \mathcal{C}$, $T_i^C \leq O\left(|\pi_i^C| \cdot n^{O(1/k^2)}\right)$, and since the time required for the algorithm from Part 1 to process the batch π_i of updates is $O\left(|\pi_i| \cdot n^{O(1/k^2)}\right)$, we get that the total time required for processing the batch π_i of updates, for both parts of our algorithm, is bounded by:

$$O\left(|\pi_i| \cdot n^{O(1/k^2)}\right) + \sum_{C \in \mathcal{C}} O\left(|\pi_i^C| \cdot n^{O(1/k^2)}\right) \leq O\left(|\pi_i| \cdot n^{O(1/k^2)}\right).$$

It is easy to verify that the total number of all edges that were deleted from or inserted into the graphs in $\{C' \mid C \in \mathcal{C}\}$ is also bounded by $O\left(|\pi_i| \cdot n^{O(1/k^2)}\right)$.

In order to complete the proof of Theorem 3.2, it now remains to prove Claim 8.17, which we do next.

Proof of Claim 8.17. For simplicity, in this proof, we denote R_C by R, and, for every vertex $v \in V(C)$, we denote $U_C(v)$ and $U_C'(v)$ by $U(v)$ and $U'(v)$, respectively.

Let β be the largest integer, such that $2^k \cdot \Delta' \cdot n^{10\beta/k^2} \leq \frac{\Delta}{n^{6/k^2}}$. Since, from the definition of valid input parameters, $\Delta \leq n^2$ holds, we get that $\beta \leq k^2$. For convenience, we denote $\Delta_0 = \frac{\Delta}{n^{6/k^2} \cdot n^{10\beta/k^2}}$. Notice that:

$$2^k \cdot \Delta' \leq \Delta_0 \leq 2^k \cdot \Delta' \cdot n^{10/k^2}. \tag{17}$$

For $1 \leq j \leq \beta$, we denote by $\Delta_j = \Delta_0 \cdot n^{10j/k^2}$, so that $\Delta_\beta = \frac{\Delta}{n^{6/k^2}}$. We also denote by $\tilde{r} = n^{10/k^2}$. Notice that, from Observation 8.16, we are guaranteed that, throughout the algorithm, the degree of every vertex $v \in A$ in R is at least Δ_β, while the degree of every vertex $u \in B$ is at most $n^{2/k^2} \cdot \Delta < \Delta_\beta \cdot n^{8/k^2} < \Delta_\beta \cdot \tilde{r}$. Throughout the algorithm, for all $0 \leq j \leq \beta$, we will maintain a

subgraph $R_j \subseteq R$, such that, for all $1 \leq i \leq k$, just before update batch π_i of updates arrives, the following properties hold:

P1. $V(R_j) = V(R)$;

P2. for every vertex $v \in A$, the degree of v in R_j is at least $\frac{\Delta_j}{2^i}$ and at most Δ_j; and

P3. for every vertex $u \in B$, the degree of u in R_j is at most $\Delta_j \cdot \tilde{r} \cdot k^{4(i-1)} \cdot (16 \log n)^{\beta - j}$.

In order to initialize and to maintain the graphs $R_0, \ldots, R_\beta$, we use the following simple claim, whose proof is deferred to Section D.3 of Appendix.

Claim 8.18 *There is a deterministic algorithm, whose input is a connected bipartite multi-graph $H = (X, Y, E)$, and parameters $z \geq 2$, $\hat{\Delta}, \gamma' \geq 1$, and $\hat{r} \geq 8\gamma' \cdot \log(|X|)$, such that, for every vertex $x \in X$, $\deg_H(x) \geq z \cdot \hat{\Delta}$, and, for every vertex $y \in Y$, $\deg_H(y) \leq \gamma' \cdot z \cdot \hat{\Delta}$. The algorithm computes, for every vertex $x \in X$, a collection $E'(x) \subseteq \delta_H(x)$ of its incident edges with $|E'(x)| = \left\lceil \hat{\Delta} \right\rceil$, such that every vertex $y \in Y$ serves as an endpoint in at most $\hat{r} \cdot \hat{\Delta}$ edges in multiset $\bigcup_{x \in X} E'(x)$. The running time of the algorithm is $O(|E| \cdot \log(|X|))$.*

We are now ready to describe our algorithms for initializing the graphs $R_0, \ldots, R_\beta$, and for maintaining them as graph R undergoes k batches of updates.

Initialization. Recall that, from Observation 8.16, the degree of every vertex $v \in A$ in R is at least Δ_β, while the degree of every vertex $u \in B$ is at most $n^{2/k^2} \cdot \Delta < \Delta_\beta \cdot n^{8/k^2} < \Delta_\beta \cdot \tilde{r}$. In order to construct the graph R_β, we let $V(R_\beta) = V(R)$. For every vertex $v \in A$, we then include in R_β an arbitrary subset of $\lfloor \Delta_\beta \rfloor$ edges incdient to v in R. The resulting graph R_β is now guaranteed to have Properties P1–P3. Consider now some integer $0 \leq j < \beta$, and assume that we have computed the graph R_{j+1} for which Properties P1–P3. We now provide an algorithm for computing graph R_j. Let $z = n^{10/k^2}$, $\hat{r} = 2\tilde{r} \cdot (16 \log n)^{\beta - j}$, and $\gamma' = 2\tilde{r} \cdot (16 \log n)^{\beta - j - 1}$. Notice that $\hat{r} \geq 8\gamma' \cdot \log n$. Moreover, for every vertex $v \in A$, the degree of $v \in A$ in R_{j+1} is at least $\frac{\Delta_{j+1}}{2} = \frac{\Delta_j}{2} \cdot n^{10/k^2} = \frac{\Delta_j}{2} \cdot z$, while the degree of every vertex $u \in B$ in R_{j+1} is at most $\Delta_{j+1} \cdot \tilde{r} \cdot (16 \log n)^{\beta - j - 1} = \frac{\Delta_j}{2} \cdot z \cdot \gamma'$. We apply the algorithm from Claim 8.18 to graph R_{j+1}, with $\hat{\Delta} = \frac{\Delta_j}{2}$, and the parameters z, γ' and $\hat{r}$ as defined above. We construct the graph R_j by first setting $V(R_j) = V(R)$, and then adding, for every vertex $v \in A$, the collection $E'(v)$ of $\left\lceil \frac{\Delta_j}{2} \right\rceil$ edges incident to v in R_{j+1} computed by the algorithm. The algorithm guarantees that every vertex $u \in B$ is an endpoint of at most $\hat{r} \cdot \frac{\Delta_j}{2} = \tilde{r} \cdot (16 \log n)^{\beta - j} \cdot \Delta_j$ edges in R_j, as required. This completes the initalization algorithm. Notice that the algorithm from Claim 8.18 is executed $\beta \leq k^2$ times, and the time required for each such execution is $O(|E(R)| \cdot \log n)$. Therefore, the total time required to initialize the graphs $R_1, \ldots, R_\beta$ is at most $O(|E(R)| \cdot k^2 \cdot \log n) \leq O(|V(C)| \cdot \Delta \cdot n^{O(1/k)})$ (since $|E(R)| \leq O(|V(C)| \cdot \Delta \cdot n^{O(1/k^2)})$ from Observation 8.16, and $k^2 \log n \leq n^{O(1/k)}$, as $k \leq (\log n)^{1/49}$ and n is large enough.)

Updates. We now fix an index $1 \leq i \leq k$, and consider the ith batch π_i of updates. Recall that, from Observation 8.16, following the batch π_i of updates, at most $|\pi_i^C| \cdot n^{10/k^2}$ edges are deleted from R_C. We denote by E_i' this set of edges. We assume that Properties P1 – P3 hold before π_i is processed.

Next, we partition the vertices of $A = V(C)$ into groups $Z_0, \ldots, Z_\beta$, as follows. Group Z_0 contains all vertices $v \in A$, such that the number of edges incident to v that lie in E_i' is at most $\frac{\Delta_0}{2^{i+1}}$. For $1 \leq j < \beta$,

group Z_j contains all vertices $v \in A$, such that $\frac{\Delta_{j-1}}{2^{i+1}} < |\delta_R(v) \cap E'_i| \leq \frac{\Delta_j}{2^{i+1}}$. Lastly, set Z_β contains all vertices $v \in A$, such that more than $\frac{\Delta_{\beta-1}}{2^{i+1}}$ edges incident to v lie in E'_i. Our algorithm only explicitly computes the groups $Z_1, \ldots, Z_\beta$; notice that this can be done in time $O(|E'_i|) \leq O(|\pi_i^C| \cdot n^{10/k^2})$.

For all $0 \leq j \leq \beta$, we start by deleting from R_j all vertices and edges that were deleted from R when processing π_i.

Consider first the group Z_0, and let $v \in Z_0$ be any vertex in this group. Then at most $\frac{\Delta_0}{2^{i+1}}$ edges that are incident to v in R have been deleted from R when processing batch π_i. Therefore, for all $0 \leq j \leq \beta$, graph R_j still contains at least $\frac{\Delta_j}{2^i} - \frac{\Delta_0}{2^{i+1}} \geq \frac{\Delta_j}{2^{i+1}}$ edges that are incident to v, and so we do not need to process the vertices of Z_0 further.

Consider now some group Z_j, for $1 \leq j \leq \beta$. Since $|E'_i| \leq |\pi_i^C| \cdot n^{10/k^2}$, and every vertex in Z_j is incident to at least $\frac{\Delta_{j-1}}{2^{i+1}}$ edges of E'_i, we get that:

$$|Z_j| \leq \frac{|\pi_i^C| \cdot n^{10/k^2} \cdot 2^{i+1}}{\Delta_{j-1}}. \tag{18}$$

Consider now any vertex $v \in Z_j$, and recall that at most $\frac{\Delta_j}{2^{i+1}}$ edges incident to v lie in E'_i. If $j < \beta$, then, for every index $j' > j$, the number of edges incident to v in $R_{j'}$ that were deleted from $R_{j'}$ is bounded by $\frac{\Delta_j}{2^{i+1}}$, and so the degree of v in $R_{j'}$ remains at least $\frac{\Delta_{j'}}{2^i} - \frac{\Delta_j}{2^{i+1}} \geq \frac{\Delta_{j'}}{2^{i+1}}$, as required. Therefore, for indices $j' > j$, the vertices of Z_j satisfy Condition P2 in graph $R_{j'}$, and they require no further processing in $R_{j'}$. For indices $0 \leq j' \leq j$, however, it is possible that too many edges incident to some vertices of Z_j have been deleted from $R_{j'}$. Intuitively, for all $0 \leq j' \leq j$, our algorithm will compute a new collection $\tilde{E}^j_{j'}$ of edges, such that, on the one hand, every vertex of Z_j is incident to at least $\frac{\Delta_{j'}}{2^{i+1}}$ such edges, while, on the other hand, every vertex of B is incident to at most $\Delta_{j'} \cdot \tilde{r} \cdot k^{4(i-1)} \cdot (16 \log n)^{\beta - j'}$ such edges. For all $0 \leq j' \leq j$, we will then delete all edges incident to the vertices of Z_j from graph $R_{j'}$, and insert the edges of $\tilde{E}^j_{j'}$ instead. The sets $\left\{\tilde{E}^j_{j'}\right\}_{j'=0}^{j}$ of edges are computed by repeatedly applying the algorithm from Claim 8.18, from higher to lower values of j', similarly to the algorithm that we used in order to initialize the data structures. We summarize the algorithm for computing these edge sets in the following observation.

Observation 8.19 *There is a deterministic algorithm that, given an index $1 \leq j \leq \beta$, computes, for all $0 \leq j' \leq j$, a collection $\tilde{E}^j_{j'} \subseteq E(R)$ of edges, such that:*

- *every edge in $\tilde{E}^j_{j'}$ is incident to a vertex of Z_j;*
- *every vertex $v \in Z_j$ is incident to at least $\frac{\Delta_{j'}}{2^{i+1}}$ and at most $\Delta_{j'}$ edges of $\tilde{E}^j_{j'}$; and*
- *every vertex $u \in B$ is incident to at most $\Delta_{j'} \cdot \tilde{r} \cdot k^{4(i-1)} \cdot (16 \log n)^{\beta - j'}$ edges of $\tilde{E}^j_{j'}$.*

The running time of the algorithm is $O(|Z_j| \cdot \Delta_j \cdot \log n)$.

Proof: We start by describing the algorithm for computing the set $\tilde{E}^j_j$ of edges. Assume first that $j = \beta$. In order to construct set $\tilde{E}^\beta_\beta$ of edges, for every vertex $v \in Z_\beta$, we include in $\tilde{E}^\beta_\beta$ arbitrary $\lfloor \Delta_\beta \rfloor$ edges that are incident to v in R; recall that, from Observation 8.16, the degree of v in R must be at least Δ_β. Moreover, the observation guarantees that the degree of every vertex $u \in B$ is at most $n^{2/k^2} \cdot \Delta < \Delta_\beta \cdot n^{8/k^2} < \Delta_\beta \cdot \tilde{r}$. Assume now that $j < \beta$. Recall that, from Property P2, every vertex

$v \in Z_j$ had degree at least $\frac{\Delta_j}{2^i}$ and at most Δ_j in graph R_j prior to the arrival of batch π_i^C. From the definition of set Z_j of vertices, for every vertex $v \in Z_j$, at most $\frac{\Delta_j}{2^{i+1}}$ edges incident to v were deleted from R. Therefore, the degree of every vertex $v \in Z_j$ in graph R_j remains at least $\frac{\Delta_j}{2^i} - \frac{\Delta_j}{2^{i+1}} \geq \frac{\Delta_j}{2^{i+1}}$, as required. We let set $\tilde{E}_j^j$ contain, for every vertex $v \in Z_j$, all edges incident to v in graph R_j. Clearly, the time required to compute the set $\tilde{E}_j^j$ of edges is bounded by $|Z_j| \cdot \Delta_j$.

Consider now some index $0 \leq j' < j$, and assume that we have computed the set $\tilde{E}_{j'+1}^j$ of edges of R, such that every edge in $\tilde{E}_{j'+1}^j$ is incident to a vertex of Z_j; every vertex of Z_j is incident to at least $\frac{\Delta_{j'+1}}{2^{i+1}}$ and at most $\Delta_{j'}$ edges of $\tilde{E}_{j'+1}^j$; and every vertex $u \in B$ is incident to at most $\Delta_{j'+1} \cdot \tilde{r} \cdot k^{4(i-1)} \cdot (16 \log n)^{\beta - j' - 1}$ edges of $\tilde{E}_{j'+1}^j$.

We construct a graph $H_{j'}$, that contains all edges in $\tilde{E}_{j'+1}^j$, and all their endpoints. For convenience, we denote by $B_{j'} = V(H_j) \cap B$. We will compute the set $\tilde{E}_{j'}^j$ of edges by applying the algorithm from Claim 8.18 to graph $H_{j'}$. In order to do so, we set the parameters as follows. First, we let $\hat{\Delta} = \frac{\Delta_{j'}}{2^{i+1}}$, and $z = n^{10/k^2}$. Recall that every vertex $v \in Z_j$ is incident to at least $\frac{\Delta_{j'+1}}{2^{i+1}} = \frac{\Delta_{j'} \cdot n^{10/k^2}}{2^{i+1}} = \hat{\Delta} \cdot z$ edges in $H_{j'}$. We set $\gamma' = \tilde{r} \cdot k^{4(i-1)} \cdot 2^{i+1} \cdot (16 \log n)^{\beta - j' - 1}$; recall that every vertex $u \in B_j$ is incident to at most $\Delta_{j'+1} \cdot \tilde{r} \cdot k^{4(i-1)} \cdot (16 \log n)^{\beta - j' - 1} = \Delta_{j'} \cdot n^{10/k^2} \cdot \tilde{r} \cdot k^{4(i-1)} \cdot (16 \log n)^{\beta - j' - 1} \leq \hat{\Delta} \cdot z \cdot \gamma'$ edges of $\tilde{E}_{j'+1}^j$. Lastly, we set $\hat{r} = 8\gamma' \log n$. We can now use the algorithm from Claim 8.18 to compute, for every vertex $v \in Z_j$, a collection $E'(v) \subseteq \delta_{H_j}(v)$ of its incident edges. We then set $\tilde{E}_{j'}^j = \bigcup_{v \in Z_j} E'(v)$. Recall that Claim 8.18 guarantees that every vertex of Z_j is incident to $\left\lceil \hat{\Delta} \right\rceil = \left\lceil \frac{\Delta_{j'}}{2^{i+1}} \right\rceil$ edges of $\tilde{E}_{j'}^j$, and every vertex of B_j is incident to at most $\hat{r} \cdot \hat{\Delta} = \frac{\Delta_{j'}}{2^{i+1}} \cdot 8\gamma' \log n \leq \Delta_{j'} \cdot \tilde{r} \cdot k^{4(i-1)} \cdot (16 \log n)^{\beta - j'}$ such edges, as required. The running time of the algorithm from Claim 8.18 is bounded by $O(|E(H_j)| \cdot \log(n)) \leq O(|Z_j| \cdot \Delta_{j'} \cdot \log n)$.

The total running time of our algorithm to compute all sets $\tilde{E}_{j'}^j$ of edges, for all $0 \leq j' \leq j$ is bounded by:

$$O\left(|Z_j| \cdot \sum_{j'=0}^{j} \Delta_{j'} \log n\right) \leq O(|Z_j| \Delta_j \log n).$$

□

We are now ready to complete our algorithm for updating the graphs $R_0, \ldots, R_\beta$. Consider an integer $0 \leq \ell \leq \beta$, and the corresponding graph R_ℓ. For all $\ell \leq j \leq \beta$, we delete from R_ℓ all edges that are incident to the vertices of Z_j, and insert the edges of $\tilde{E}_\ell^j$ instead (if $\ell = 0$, then we only do so for indices $1 \leq j \leq \beta$). Recall that, before the current update, from Property P3, every vertex of B was incident to at most $\Delta_\ell \cdot \tilde{r} \cdot k^{4(i-1)} \cdot (16 \log n)^{\beta - \ell}$ edges in graph R_ℓ. For all $\ell \leq j \leq \beta$, we are guaranteed that every vertex of B is incident to at most $\Delta_\ell \cdot \tilde{r} \cdot k^{4(i-1)} \cdot (16 \log n)^{\beta - \ell}$ edges of $\tilde{E}_\ell^j$. Since $\beta \leq k^2$, we get that, in the updated graph R_ℓ, the degree of every vertex $v \in B$ is bounded by $(\beta+1) \cdot \Delta_\ell \cdot \tilde{r} \cdot k^{4(i-1)} \cdot (16 \log n)^{\beta - \ell} \leq \Delta_\ell \cdot \tilde{r} \cdot k^{4i} \cdot (16 \log n)^{\beta - \ell}$. From our construction and the discussion above, every vertex of A has degree at least $\frac{\Delta_\ell}{2^{i+1}}$ and at most Δ_ℓ in R_ℓ. Therefore Properties P1 – P3 continue to hold after the update. From our discussion, the time required for processing the set Z_j of vertices, for $0 \leq j \leq \beta$, is bounded by:

$$O(|Z_j| \Delta_j \log n) \leq O\left(\frac{|\pi_i^C| \cdot n^{10/k^2} \cdot 2^{i+1} \cdot \Delta_j \cdot \log n}{\Delta_{j-1}}\right) \leq O\left(|\pi^c| \cdot n^{O(1/k^2)}\right)$$

(we have used the fact that $i \leq k$, inequalities 12 and 18, together with the facts that $k \leq (\log n)^{1/49}$ and n is large enough). Since $\beta \leq k^2 \leq n^{O(1/k^2)}$, the total time required to process batch π_i is bounded by $O\left(|\pi^c| \cdot n^{O(1/k^2)}\right)$.

We are now ready to complete the proof of Claim 8.17. For every vertex $v \in V(C)$, we simply let $U'_C(v)$ be the set of all vertices that serve as endpoints to the edges of R_0 that are incident to v. From Property P2, for every vertex $v \in V(C)$, $\frac{\Delta_0}{2^k} \leq |U'_C(v)| \leq \Delta_0$ holds at all times. Since, from Inequality 17, $2^k \cdot \Delta' \leq \Delta_0 \leq 2^k \cdot \Delta' \cdot n^{10/k^2}$, we get that, for every vertex $v \in V(C)$, $\Delta' \leq |U'_C(v)| \leq 2^k \cdot \Delta' \cdot n^{10/k^2} \leq \Delta' \cdot n^{11/k^2}$ always holds.

From Property P3, for every vertex $x \in B$, the degree of x in R_0 is bounded by:

$$\Delta_0 \cdot \tilde{r} \cdot k^{4k} \cdot (16 \log n)^{\beta} \leq \Delta' \cdot n^{20/k^2} \cdot 2^{8k^2} \cdot (\log n)^{k^2} \leq \Delta' \cdot n^{16/k}.$$

since $\tilde{r} = n^{10/k^2}$, $\beta \leq k^2$, and $\Delta_0 \leq 2^k \cdot \Delta' \cdot n^{10/k^2}$ as observed above. We have also used the fact that, from the definition of valid input parameters, $k \leq (\log n)^{1/49}$, and so $n^{1/k^4} \geq 2^{(\log n)^{4/5}} \geq \log n$, since n is large enough. Lastly, we have used Inequality 12. As we have shown, the initialization time of our algorithm is $O(|V(C)| \cdot \Delta \cdot n^{O(1/k)})$, and, for all $1 \leq i \leq k$, the time required for updates following the processing of batch π_i^C of edge deletions to graph C is bounded by $O\left(|\pi_i^C| \cdot n^{O(1/k^2)}\right)$. This completes the proof of Claim 8.17. □

9 Applications to (Fault-Tolerant) Graph Sparsification

Consider a simple graph G, and a router decomosition $\left(\mathcal{C}, \{C'\}_{C \in \mathcal{C}}\right)$ of G with parameters $\Delta^*, \tilde{d}, \tilde{\eta}$ and ρ (see Definition 3.1). Recall that the decomposition ensures that, for every cluster $C \in \mathcal{C}$, the corresponding subgraph $C' \subseteq C$ is a $(\Delta^*, \tilde{d}, \tilde{\eta})$-router, for the set $V(C)$ of supported vertices. Also recall that we have denoted by $E^{\text{del}} = E(G) \setminus \left(\bigcup_{C \in \mathcal{C}} E(C)\right)$ – the set of edges that do not lie in any cluster of the decomposition. Consider the graph H', whose vertex set is $V(G)$, and edge set is $E^{\text{del}} \cup \left(\bigcup_{C \in \mathcal{C}} E(C')\right)$, and recall that the algorithm from Theorem 1.3 maintains a graph H with $H' \subseteq H$. In this subsection, we explore various properties of the graph H', that imply that the graph H itself has many useful properties as well. This, in turn, implies our results for constructing and maintaining spanners, fault-tolerant spanners with respect to bounded-degree faults, low-congestion spanners, and connectivity certificates. We start with discussing general dynamic spanners and low-congestion spanners.

9.1 Dynamic Spanners and Low-Congestion Spanners

Dynamic Spanners. Let G be a simple n-vertex graph, and let $\mathcal{R} = (\mathcal{C}, \{C'\}_{C \in \mathcal{C}})$ be a router decomposition of G with parameters $\Delta^*, \tilde{d}, \tilde{\eta}$ and ρ. Consider the corresponding graph H' whose vertex set is $V(G)$, and edge set is $E^{\text{del}} \cup \left(\bigcup_{C \in \mathcal{C}} E(C')\right)$, where $E^{\text{del}} = E(G) \setminus \left(\bigcup_{C \in \mathcal{C}} E(C)\right)$. In the following observation, we show that H' is a $\tilde{d}$-spanner.

Observation 9.1 *Graph H' is $\tilde{d}$-spanner.*

Proof: Let $e = (u, v)$ be any edge of G. If $e \in E(H')$, then $\mathsf{dist}_{H'}(u, v) = 1$. Otherwise, there must be some cluster $C \in \mathcal{C}$, with $e \in E(C)$. Consider the subgraph $C' \subseteq C$ that the decomposition provides. Since $V(C) = V(C')$, and C' is a $(\Delta^*, \tilde{d}, \tilde{\eta})$ router for the set $V(C)$, it must be the case that $\mathsf{dist}_{C'}(u, v) \leq \tilde{d}$. The claim follows as $C' \subseteq H'$. □

By combining Theorem 1.3 with Observation 9.1, we obtain the following immediate corollary.

Corollary 9.2 *There is a deterministic algorithm, whose input is a graph G with n vertices and no edges, that undergoes an online sequence of at most n^2 edge deletions and insertions, such that G remains a simple graph throughout the update sequence. Each edge inserted into G has an integral length $\ell(e) \in \{1, \ldots, L\}$. Additionally, the algorithm is given parameters $512 \leq k \leq (\log n)^{1/49}$ and $\frac{1}{k} \leq \delta \leq \frac{1}{400}$, such that k and $\frac{1}{\delta}$ are integers. The algorithm maintains a subgraph $H \subseteq G$ with $V(H) = V(G)$ and $|E(H)| \leq O\left(n^{1+O(1/k)} \log L\right)$, such that H is a k'-spanner for G, for $k' = k^{19} \cdot 2^{O(1/\delta^6)}$. The worst-case update time of the algorithm is $n^{O(\delta)}$ per operation, and the number of edge insertions and deletions that graph H undergoes after each update to G is bounded by $n^{O(1/k)}$.*

Proof: We partition all edges that ever belong to G over the course of the algorithm into sets $E_0, \ldots, E_{\lceil \log L \rceil}$, where, for $0 \leq i \leq \lceil \log L \rceil$, set E_i contains all edges e with $2^i \leq \ell(e) < 2^{i+1}$. For all $0 \leq i \leq \lceil \log L \rceil$, we let $G_i = G[E_i]$. For all $0 \leq i \leq \lceil \log L \rceil$, we apply the algorithm from Theorem 1.3 to graph G_i, with parameters k and δ that remain unchanged, and parameter $\Delta = \left\lceil n^{20/k} \right\rceil$. We denote the graph H that the algorithm maintains by H_i. Recall that $|E(H_i)| \leq n^{1+O(1/k)} \cdot \Delta \leq n^{1+O(1/k)}$. From Observation 9.1, it is immediate to verify that H_i is a k'-spanner for G_i, for $k' = k^{19} \cdot 2^{O(1/\delta^6)}$. Finally, we let $H = \bigcup_i H_i$. Notice that $|E(H)| \leq O\left(n^{1+O(1/k)} \log L\right)$ must hold, and it is easy to verify that the algorithm from Theorem 1.3 can be extended to maintain the graph H with worst-case update time $n^{O(\delta)}$, such that the number of edge updates that H undegroes following each update to H is at most $n^{O(1/k)}$. □

Low-Congestion Spanners. We formalize the notion of *low-congestion spanners*, that was implicitly introduced by Chen et al. [CKL+22]. The new notion views a spanner H of G as a sparse subgraph of G, with the additional property that G can be embedded into H with low congestion and path length bounds. Notice that, if we use the standard notion of a t-spanner H of G, then G can be embedded into H via paths of length at most t, but the congestion of the embedding may be as large as $|E(G)|$. Low-congestion spanners aim at optimizing the length and the congestion parameters of this embedding simultaneously. We now provide a formal definition of low-congestion spanners.

Definition 9.3 (Low-Congestion Spanners) *Let G be a graph, let $H \subseteq G$ be a subgraph of G with $V(H) = V(G)$, and let $d, \eta > 0$ be parameters. We say that H is a (d, η)-low congestion spanner for G, if there is an embedding $\mathcal{Q} = \{Q(e) \mid e \in E(G)\}$ of G into H, such that the length of every path in $\mathcal{Q}$ is a most d, and the paths in $\mathcal{Q}$ cause congestion at most η.*

We note that low-congestion spanners can also be defined with respect to vertex congestion, and it is the spanners of the latter kind that were considered by [vdBCK+23]. Their algorithm maintains such a spanner H for a dynamic n-vertex graph G, with $|E(H)| \leq O(n^{1+o(1)})$, together with the embedding $\mathcal{Q}$ of G into H, such that the lengths of the embedding paths are bounded by $d = n^{o(1)}$, and vertex-congestion of the embedding $\mathcal{Q}$ is bounded by $\eta = n^{o(1)} \cdot \max_{v \in V(G)} \{\deg_G(v)\}$. The amortized update time and recourse bounds are $n^{o(1)}$.

We use the following simple lemma.

Lemma 9.4 *Let G be an n-vertex graph, and let $\mathcal{R} = (\mathcal{C}, \{C'\}_{C \in \mathcal{C}})$ be a router decomposition of G with parameters $\Delta^*, \tilde{d}, \tilde{\eta}$ and ρ. Then for every cluster $C \in \mathcal{C}$, there is an embedding $\mathcal{Q}_C$ of C into C' via paths of length at most $\tilde{d}$, that cause congestion at most $\eta = \max\left\{\frac{16\tilde{\eta} \cdot \Delta_{\max}(C)}{\Delta^*}, 16 \log n\right\}$, where $\Delta_{\max}(C)$ is the maximum vertex degree in C.*

Proof: Consider a cluster $C \in \mathcal{C}$, and recall that $C' \subseteq C$ is a subgraph of C with $V(C') = V(C)$, that is a $(\overline{\Delta}^*, \tilde{d}, \tilde{\eta})$-router for the set $V(C)$ of supported vertices.

We define a demand $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ as follows: set Π contains all pairs (u,v) of vertices, for which edge (u,v) lies in C, and the corresponding demand value $D(u,v)$ is 1. Clearly, demand $\mathcal{D}$ is $\Delta_{\max}(C)$-restricted in C'. It is now enough to show that demand $\mathcal{D}$ can be routed integrally in C' via paths of length at most $\tilde{d}$, that cause congestion at most η.

Consider the demand $\mathcal{D}'$ that is obtained from $\mathcal{D}$ by scaling every demand $D(u,v)$ down by factor $\frac{\Delta_{\max}(C)}{\Delta^*}$. Then it is immediate to see that $\mathcal{D}'$ is a Δ^*-restricted demand, and so there is a routing f of this demand in C' via paths of length at most $\tilde{d}$, that cause congestion at most $\tilde{\eta}$. By scaling the flow on each flow-path up by factor $\frac{\Delta_{\max}(C)}{\Delta^*}$, we obtain a routing f of the demand $\mathcal{D}$ in C via paths of length at most $\tilde{d}$, that cause congestion at most $\frac{\tilde{\eta}\cdot\Delta_{\max}(C)}{\Delta^*}$. Our last step is to turn the resulting routing into an integral one, using the standard Randomized Rounding procedure of Raghavan and Thompson [RT87].

Consider any edge $e = (u,v) \in E(C)$, and let $\mathcal{P}_{u,v}$ be the set of all u-v paths in C' of length at most $\tilde{d}$. Since flow f sends 1 flow unit from u to v over paths of length at most $\tilde{d}$, it defines a probability distribution $D_{u,v}$ over the paths of $\mathcal{P}_{u,v}$, where every path $P \in \mathcal{P}_{u,v}$ is assigned probability $f(P)$. For each such edge $e = (u,v) \in E(C)$, we choose a single path $P(e) \in \mathcal{P}_{u,v}$ from this distribution, where the choices between the different edges are independent. Let $\mathcal{Q}_C = \{P(e) \mid e \in E(C)\}$ be the resulting collection of paths, that defines an embedding of C into C'. Clearly, the paths in $\mathcal{Q}$ have length at most $\tilde{d}$ each. Next, we argue that with high probability, the paths in $\mathcal{Q}$ cause congestion at most η in C'. Indeed, the expected congestion for an edge $e \in E(C')$ is at most $\frac{\tilde{\eta}\cdot\Delta_{\max}(C)}{\Delta^*}$. Using the Chernoff bound from Lemma 2.16 with $t = \eta = \max\left\{\frac{16\tilde{\eta}\cdot\Delta_{\max}(C)}{\Delta^*}, 16\log n\right\}$, we get that the probability that the congestion on e is greater than η is bounded by $2^{-\eta} < 1/n^3$, since $\eta \geq 16\log n$. Using the union bound, with high probability, the congestion on every edge of C' is at most η. □

We obtain the following immediate corollary of Lemma 9.4.

Corollary 9.5 *Let G be an n-vertex graph, and let $\mathcal{R} = (\mathcal{C}, \{C'\}_{C\in\mathcal{C}})$ be a router decomposition of G with parameters $\Delta^*, \tilde{d}, \tilde{\eta}$ and ρ. Consider the corresponding graph H' whose vertex set is $V(G)$, and edge set is $E^{\mathrm{del}} \cup \left(\bigcup_{C\in\mathcal{C}} E(C')\right)$, where $E^{\mathrm{del}} = E(G) \setminus \left(\bigcup_{C\in\mathcal{C}} E(C)\right)$. Then graph H' is an $(\tilde{d}, \eta)$-low congestion spanner for G, where $\eta = \max\left\{\frac{16\tilde{\eta}\cdot\Delta_{\max}(G)}{\Delta^*}, 16\log n\right\}$, and $\Delta_{\max}(G)$ is the maximum vertex degree in G.*

Proof: We obtain an embedding $\mathcal{Q}$ of $E(G)$ into H, as follows. First, since $E^{\mathrm{del}} \subseteq E(H')$, we embed every edge $e \in E^{\mathrm{del}}$ into itself. Each remaining edge $e \in E(G)$ must lie in some cluster $C \in \mathcal{C}$. For each such cluster C, we use the embedding $\mathcal{Q}_C$ of C into C' that is given by Lemma 9.4. Since, for every cluster $C \in \mathcal{C}$, the corresponding graph C' is contained in H', and since the clusters in $\mathcal{C}$ are edge-disjoint, we obtain an embedding of G into H' via paths of length at most $\tilde{d}$, that cause congestion at most η. □

Finally, by combining Theorem 1.3 with Corollary 9.5, we obtain the following immediate corollary; the proof is essentially identical to the proof of Corollary 9.2 and is omitted here.

Corollary 9.6 *There is an algorithm, whose input is a graph G with n vertices and no edges, that undergoes an online sequence of at most n^2 edge deletions and insertions, such that G remains a simple graph throughout the update sequence. Each edge inserted into G has an integral length $\ell(e) \in \{1,\ldots,L\}$. Additionally, the algorithm is given parameters $512 \leq k \leq (\log n)^{1/49}$, $\frac{1}{k} \leq \delta \leq \frac{1}{400}$ and $\Delta \geq n^{20/k}$, such that k, Δ and $\frac{1}{\delta}$ are integers. The algorithm maintains a subgraph $H \subseteq G$ with*

$V(H) = V(G)$ and $|E(H)| \leq O\left(n^{1+O(1/k)} \cdot \Delta \cdot \log L\right)$, such that H is a (k', η)-low-congestion spanner for G, for $k' = k^{19} \cdot 2^{O(1/\delta^6)}$ and $\eta \leq n^{O(1/k)} \cdot \max\left\{\frac{\Delta_{\max}(G)}{\Delta}, 1\right\}$, where $\Delta_{\max}(G)$ is the maximum degree in G. The worst-case update time of the algorithm is $n^{O(\delta)}$ per operation, and the number of edge insertions and deletions that graph H undergoes after each update to G is bounded by $n^{O(1/k)}$.

9.2 Dynamic Fault-Tolerant Spanners and Connectivity Certificates

In this subsection we provide a key observation that routers with a sufficiently large minimum degree are inherently resilient to edge failures. We then show that, by setting the parameter Δ in Theorem 1.3 appropriately, we can ensure that the resulting graph H is resilient to bounded-degree faults. Lastly, we prove that H can be used as a connectivity certificate.

9.2.1 Resilience of Routers to Bounded-Degree Faults

Let G be a graph, and let F be a subset of edges of G (edge faults). We denote by $\deg(F)$ the maximum, over all vertices of v, of the number of edges in F that are incident to v. We say that a set F of edges is an f-BD faulty set, if $\deg(F) \leq f$. Recall that a graph G is a $(\overline{\Delta}, d, \eta)$-router for the set $V(G)$ of supported vertices, if every Δ-restricted demand $\mathcal{D}$ over $V(G)$ can be routed with congestion at most η in G, via paths of length at most d. In this subsection we show that routers are resilient to bounded-degree faults, and the sense that $G \setminus F$ remains a router, albeit with slightly weaker parameters. The main result of this subsection is summarized in the following theorem, whose proof is provided below.

Theorem 9.7 *Let G be an n-vertex graph, and let $f, k, d > 0$, $1 \leq \eta < n^2$ and $\Delta \geq 32f \cdot n^{1/k} \cdot \eta$ be parameters. Assume further that G is a $(\overline{\Delta}, d, \eta)$-router with respect to the set $V(G)$ of supported vertices. Then for every f-BD faulty set $F \subseteq E(G)$ of edges, graph $G \setminus F$ is a $(\overline{\Delta}, O(k) \cdot d, O(k) \cdot \eta)$-router for the set $V(G)$ of supported vertices.*

The above theorem should be compared to the resilience of length-constrained expanders against f-BD faults, that was established in [BDR22]. Specifically, they show in Theorem 1.2 of [BDR22] that, if G is an n-vertex (h, s)-length φ-expander with minimum degree $\Omega(fn^{\epsilon}/\varphi)$, then, for any f-BD faulty set $F \subseteq E(G)$ of edges, the graph $G \setminus F$ is an $(hs)^{1/\epsilon}$-spanner of G; in other words, the distances in $G \setminus F$ may grow by at most factor $(hs)^{1/\epsilon}$. It then follows that the f-FD spanners obtained by [BDR22] via LC-expanders have size $\widetilde{O}(f \cdot n^{1+1/k})$ and stretch $t = k^{O(k)}$. The question of improving the exponential dependence of the stretch on k, while preserving the size of the spanner was left open in [BDR22] (see Theorem 1.14 therein), and this improvement would immediately imply stronger bounds for f-FD spanners. We resolve this open problem in Theorem 9.7, for the case where the LC-expander has a bounded diameter (that is, it is a router).

Proof of Theorem 9.7. Consider any demand $\hat{\mathcal{D}} = \left(\hat{\Pi}, \left\{\hat{D}(a,b)\right\}_{(a,b)\in\hat{\Pi}}\right)$. We say that demand $\hat{\mathcal{D}}$ is *integral*, if, for every pair $(a,b) \in \hat{\Pi}$, the demand value $\hat{D}(a,b)$ is an integer. It will be convenient for us to represent such an integral demand $\hat{\mathcal{D}}$ by a multiset $I = \{(a_1, b_1), \ldots, (a_\chi, b_\chi)\}$ of pairs of vertices, with $\chi = \sum_{(a,b)\in\hat{\Pi}} \hat{D}(a,b)$, where each pair $(a,b) \in \hat{\Pi}$ appears in I with multiplicity $\hat{D}(a,b)$. Throughout, we identify each pair (and its two vertices) by its index in the multiset, (a_ℓ, b_ℓ). Conversely, any multiset I of pairs of vertices in G naturally defines the corresponding demand $\hat{\mathcal{D}}(I)$, where the demand $\hat{D}(a,b)$ for every pair (a,b) is the number of times it appears in I. The resulting demand is $\hat{\Delta}$-restricted, for a parameter $\hat{\Delta}$, if every vertex of G appears in at most $\hat{\Delta}$ pairs in I. For

convenience, we will not distinguish between the integral demand $\hat{\mathcal{D}}$, and its corresponding multiset I of vertex pairs. Consider now an integral demand $I = \{(a_1, b_1), \ldots, (a_\chi, b_\chi)\}$. A routing f of I is *integral* if, for every pair (a_i, b_i), there is exactly one flow-path P connecting a_i to b_i with $f(P) > 0$, and moreover, $f(P) = 1$. An integral routing of an integral demand $I = \{(a_1, b_1), \ldots, (a_\chi, b_\chi)\}$ can equivalently be defined as a collection $\mathcal{P} = \{P_1, \ldots, P_\chi\}$ of paths, where, for all $1 \leq i \leq \chi$, path P_i connects a_i to b_i. We will use the following simple observation.

Observation 9.8 *Let $\hat{\mathcal{D}} = \left(\hat{\Pi}, \left\{\hat{D}(a,b)\right\}_{(a,b)\in\hat{\Pi}}\right)$ be an integral demand in G, and assume that $\hat{\mathcal{D}}$ is an $(\alpha \cdot \Delta)$-restricted demand, for some parameter $\alpha \geq 8 \log n$. Then there is an integral routing $\mathcal{P}$ of $\hat{\mathcal{D}}$, where each path in $\mathcal{P}$ has length at most d, and the paths cause congestion at most $8\alpha\eta$.*

Proof: Let $\hat{\mathcal{D}}'$ be a demand that is obtained from $\hat{\mathcal{D}}$, by scaling all demands in $\hat{\mathcal{D}}$ down by factor α. It is easy to verify that demand $\hat{\mathcal{D}}'$ is Δ-restricted. Since G is a $(\overline{\Delta}, d, \eta)$-router with respect to the set $V(G)$ of supported vertices, there is a routing f' of $\hat{\mathcal{D}}'$ in G via paths of length at most d, that causes congestion at most η. By scaling the flow value $f'(P)$ on every flow-path up by factor α, we obtain a routing f of $\hat{\mathcal{D}}$ via paths of length at most d, that cause congestion at most $\alpha\eta$. Lastly, in order to convert this routing into an integral one, we use the standard Randomized Rounding procedure of Raghavan and Thompson [RT87].

Consider the multiset $I = \{(a_1, b_1), \ldots, (a_\chi, b_\chi)\}$ of vertex pairs that corresponds to the demand $\hat{\mathcal{D}}$. We can naturally view the flow f as routing the pairs in I via paths of length at most d, with congestion at most $\alpha\eta$. Consider now any demand pair $(a_i, b_i) \in I$, and let $\mathcal{P}_{(a_i,b_i)}$ be the collection of all a_i-b_i paths of length at most d in G. Since flow f sends 1 flow unit from a_i to b_i, it defines a probability distribution D_i over the paths of $\mathcal{P}_{(a_i,b_i)}$, where every path $P \in \mathcal{P}_{(a_i,b_i)}$ is assigned probability $f(P)$. For all $1 \leq i \leq \chi$, we choose a single path $P_i \in \mathcal{P}_{(a_i,b_i)}$ from this distribution, where the choices between the different pairs are independent. Let $\mathcal{P} = \{P_1, \ldots, P_\chi\}$ be the resulting routing. Clearly, the paths in $\mathcal{P}$ have length at most d each. Next, we argue that with high probability, the congestion of the routing is at most $8\alpha\eta$. Indeed, the expected congestion for an edge $e \in E(G)$ is at most $\alpha \cdot \eta$. Using the Chernoff bound from Lemma 2.16 with $t = 8\alpha\eta$, we get that the probability that the congestion on e is greater than $8\alpha\eta$ is bounded by $2^{-8\alpha\eta} < 1/n^3$, since $\alpha \geq 8 \log n$. Using the union bound, with high probability, the congestion on every edge is at most $8\alpha\eta$. □

Let $\mathcal{D} = \left(\Pi, \{D(a,b)\}_{(a,b)\in\Pi}\right)$ be a Δ-restricted demand for G, and let $F \subseteq E(G)$ be an f-BD faulty edge set. We will denote by $V(F)$ the set of all vertices that serve as endpoints of the edges in F. Our goal is to show that $\mathcal{D}$ can be routed in $G \setminus F$ via paths of length at most $O(k) \cdot d$, with congestion at most $O(k) \cdot \eta$. Our first step is to convert $\mathcal{D}$ into an integral demand.

Let $\mathcal{D}' = \left(\Pi, \{D'(a,b)\}_{(a,b)\in\Pi}\right)$ be the demand obtained from $\mathcal{D}$, by setting $D'(a,b) = \lceil D(a,b) \cdot n \rceil$ for every pair $(a,b) \in \Pi$ of vertices. It is easy to verify, that, since every vertex of G may participate in at most n pairs in Π, the demand $\mathcal{D}'$ is Δ'-restricted, for $\Delta' = 2n\Delta$. Let $\eta' = 16\eta \cdot n$. Assume that there is a routing f' of $\mathcal{D}'$ in $G \setminus F$ via paths of length $O(k) \cdot d$, with congestion at most $O(k) \cdot \eta'$. Then we can obtain a routing f of $\mathcal{D}$ in G via paths of length $O(k) \cdot d$ and congestion at most $O(k) \cdot \eta$, by scaling the flow f' down by factor n, and then reducing it as needed. Therefore, from now on, it is enough to show that there is a routing f' of $\mathcal{D}'$ in $G \setminus F$ via paths of length $O(k) \cdot d$, with congestion at most $O(k) \cdot \eta'$. Notice that the resulting demand $\mathcal{D}' = \left(\Pi, \{D'(a,b)\}_{(a,b)\in\Pi}\right)$ is integral. Throughout, we denote by $I = \{(a_1, b_1), \ldots, (a_\chi, b_\chi)\}$ the corresponding multiset of vertex pairs. Our goal is to send 1 flow unit between every pair (a_ℓ, b_ℓ), such that the resulting flow only uses flow-paths of length at most $O(k) \cdot d$, and causes congestion at most $O(k \cdot \eta')$. We will repeatedly use the following immediate corollary of Observation 9.8.

Corollary 9.9 *If I' is any integral Δ'-restricted demand in G, then there is an integral routing $\mathcal{P}$ of I', where each path in $\mathcal{P}$ has length at most d, and the paths cause congestion at most $16\eta n = \eta'$.*

Our algorithm consists of $z \leq O(k)$ iterations. In each iteration i, we define a Δ'-restricted demand $\mathcal{D}_i$ for $V(G)$, and then consider its routing f_i in G. For each flow-path P that carries non-zero flow, we discard all edges of $F \cap E(P)$, partitioning P into segments, some of which continue to carry $f_i(P)$ flow units; we denote the resulting flow, that does not use the edges of F, by f'_i. The final routing of $\mathcal{D}$ is obtained by combining the resulting flows $f'_1, \ldots, f'_z$.

Consider now the multiset I of vertex pairs, corresponding to the demand $\mathcal{D}'$. From Corollary 9.9, there is an integral routing $\mathcal{P} = \{P_1, \ldots, P_\chi\}$ of I via paths of length at most d, with congestion at most η', where, for all $1 \leq \ell \leq \chi$, path P_ℓ connects a_ℓ to b_ℓ. We say that a path $P \in \mathcal{P}$ is *safe*, if $E(P) \cap F = \emptyset$, and we say that it is *unsafe* otherwise. We say that a pair $(a_\ell, b_\ell) \in I$ is *safe* if P_ℓ is a safe path, and we say that it is *unsafe* otherwise. From now on, it is sufficient to compute a routing for unsafe pairs in I.

Consider an unsafe pair $(a_\ell, b_\ell) \in I$. Let x_ℓ be the vertex of $V(F) \cap V(P_\ell)$ lying closest to a_ℓ on the path, and let P^1_ℓ be the subpath of P_ℓ from a_ℓ to x_ℓ. Similarly, let y_ℓ be the vertex of $V(F) \cap V(P_\ell)$ lying closest to b_ℓ on the path, and let P^2_ℓ be the subpath of P_ℓ from y_ℓ to b_ℓ. Notice that both P^1_ℓ and P^2_ℓ are safe paths. Consider the following multiset of vertex pairs:

$$I' = \{(x_\ell, y_\ell) \mid P_\ell \in \mathcal{P}, P_\ell \cap F \neq \emptyset\}$$

In the remainder of the proof, we will compute an integral routing of the pairs in I' in $G \setminus F$ via paths of length at most $O(k) \cdot d$, that cause congestion at most $O(k) \cdot \eta'$. From the following observation, such a routing will complete the proof of Theorem 9.7.

Observation 9.10 *Suppose there is an integral routing $\mathcal{Q}$ of the demand I' in $G \setminus F$ via paths of length at most $O(k) \cdot d$ that cause congestion at most $O(k) \cdot \eta'$. Then there is a routing $\mathcal{P}'$ of the demand I in $G \setminus F$ via paths of length at most $O(k) \cdot d$, that cause congestion at most $O(k) \cdot \eta'$.*

Proof: For every unsafe pair $(a_\ell, b_\ell) \in I$, let $Q_\ell \in \mathcal{Q}$ be the path connecting x_ℓ to y_ℓ. Let P'_ℓ be the path obtained by concatenating the paths P^1_ℓ, Q_ℓ, and P^2_ℓ. Then path P'_ℓ is contained in $G \setminus F$, it connects a_ℓ to b_ℓ, and its length is at most $O(k) \cdot d$. For every safe pair $(a_\ell, b_\ell) \in I$, we simply set $P'_\ell = P_\ell$. It is also immediate to verify that the resulting set $\mathcal{P}' = \left\{P'_1, \ldots, P'_\chi\right\}$ of paths causes congestion at most $O(k) \cdot \eta'$, and it routes the pairs in I in graph $G \setminus F$. □

In the remainder of the proof, we show that there exists an integral routing of the demands in I' via paths of length at most $O(k) \cdot d$, that cause congestion at most $O(k) \cdot \eta'$. We construct such a routing over the course of $z = 10k$ iterations. For all $1 \leq i \leq z$, at the beginning of iteration i, we are given a partition of the multiset I' into two subsets: set $\mathcal{S}_i$ of pairs that we refer to as *i-safe*, and set $\mathcal{U}_i$ of pairs that we refer to as *i-unsafe*. The set $\mathcal{S}_i$ of i-safe pairs is guaranteed to satisfy the following property:

Q1. there is an integral routing $\mathcal{P}_i$ of the pairs in $\mathcal{S}_i$ in graph $G \setminus F$ via paths of length at most $3i \cdot d$, that cause congestion at most $2i \cdot \eta'$.

For all $1 \leq i \leq z$, we denote by $V'_i = \{x_\ell, y_\ell\}_{(x_\ell, y_\ell) \in \mathcal{U}_i}$ the multiset of all vertices that appear in the i-unsafe pairs. For every vertex $v \in V'_i$, we will construct a tree-like structure (that, for brevity, we refer to as a tree), denoted by $T_i(v)$, whose root vertex is v, and all other vertices lie in $V(F)$. Graph

$T_i(v)$ is not a tree strictly speaking, in the sense that a vertex of $V(F)$ may appear multiple times in the tree. But if each such appearence is replaced with a new copy of the vertex, then we naturally obtain a tree graph, that is denoted by $T_i(v)$. We note that, if a vertex v appears in several pairs in $\mathcal{U}_i$, then we construct a separate tree for each such appearance of v, so v will serve as a root for several such trees. Throughout, we use a parameter $\lambda = \left\lfloor \frac{\Delta'}{f \cdot \eta'} \right\rfloor$. We ensure that, for all $1 \leq i \leq z$, at the beginning of iteration i, the following properties hold for every vertex $v \in V_i'$:

Q2. $T_i(v)$ is a λ-regular tree of depth $i-1$; and

Q3. the multiset $L_i(v)$ of the leaf vertices of $T_i(v)$ has cardinality λ^{i-1}.

Additionally, the collection $\mathcal{T}_i = \{T_i(w)\}_{w \in V_i'}$ of trees satisfies the following properties:

Q4. every vertex of $V(F)$ appears with multiplicity at most $f\eta'$ in the union of multisets $\{L_i(v)\}_{v \in V_i'}$; and

Q5. For every tree $T \in \mathcal{T}_i$, for every edge $e = (u, w) \in E(T)$, there is a path $R(e)$ connecting u to w in $G \setminus F$ of length at most d, such that the paths in set $\mathcal{R}_i = \{R(e) \mid e \in E(T), T \in \mathcal{T}_i\}$ cause congestion at most $i\eta'$.

We now show that, if the above invariants are always obeyed, then, for some $j \leq 10k$, $\mathcal{U}_j = \emptyset$ and $\mathcal{S}_j = I'$ holds.

Claim 9.11 *Assume that Invariants Q1–Q5 are always obeyed. Then, for some $1 \leq j \leq 10k$, $\mathcal{U}_j = \emptyset$ and $\mathcal{S}_j = I'$ must hold.*

Proof: Assume for contradiction that the claim is false, and so $\mathcal{U}_{10k} \neq \emptyset$ must hold. Consider any pair $(x_\ell, y_\ell) \in \mathcal{U}_{10k}$. Recall that $\lambda = \left\lfloor \frac{\Delta'}{f \cdot \eta'} \right\rfloor$, while $\Delta' = 2n\Delta \geq 64 f n^{1+1/k} \cdot \eta \geq 2 f n^{1/k} \eta'$, since $\eta' = 16\eta \cdot n$. Therefore, $\lambda = \left\lfloor \frac{\Delta'}{f \cdot \eta'} \right\rfloor \geq 2n^{1/k} - 1$. From Invariant Q3, the multiset $L_{10k}(x_\ell)$ of the leaf vertices of $T_i(x_\ell)$ has cardinality at least $\lambda^{10k-1} > 2n^5$. However, from Invariant Q4, every vertex of $V(F)$ appears with multiplicity at most $f\eta' \leq 2n^4$ in $L_{10k}(x_\ell)$. Therefore, the number of distinct vertices in $L_{10k}(x_\ell)$ is strictly greater than n, leading to a contradiction. Since multisets $\mathcal{U}_j$ and $\mathcal{S}_j$ partition the multiset I', the claim follows. □

At the beginning of the algorithm, we set $\mathcal{S}_1 = \emptyset$, $\mathcal{P}_1 = \emptyset$, and $\mathcal{U}_1 = I'$. As before, we let V_1' be the multiset of all vertices that participate in the pairs in $\mathcal{U}_1$. For every vertex $v \in V_1'$, we let $T_1(v)$ be a tree of depth 0, that only consists of the root vertex v. It is immediate to verify that Invariants Q1–Q3 and Q5 hold. In order to see that Invariant Q4 holds as well, recall that the collection $\mathcal{P}$ of paths that defined the original routing of the demands in I had congestion at most η'. Consider any vertex $u \in V(F)$. The number of times that u appears in the trees of $\mathcal{T}_1 = \{T_1(v) \mid v \in V_1'\}$ is bounded by the number of paths $P \in \mathcal{P}$, such that P contains an edge of the faulty set F that is incident to u. Since at most f edges incident to u may lie in F, and since each such edge participates in at most η' paths, vertex u may serve as the root of at most $f \cdot \eta'$ trees in $\mathcal{T}_1$.

The algorithm is executed as long as $\mathcal{U}_i \neq \emptyset$ holds. We now describe the ith iteration. We assume that we are given a partition $(\mathcal{S}_i, \mathcal{U}_i)$ of the set I' of pairs, the corresponding multiset V_i' of vertices, and the collection $\mathcal{T}_i = \{T_i(v) \mid v \in V_i'\}$ of trees, for which invariants Q1–Q5 hold. We now provide the description of the ith iteration.

Desciption of the ith Iteration. For every pair $(x_\ell, y_\ell) \in \mathcal{U}_i$, we construct a new demand $\mathcal{D}^i_\ell$, as follows. Consider the multisets $L_i(x_\ell)$ and $L_i(y_\ell)$ of vertices (the leaf vertices of the trees $T_i(x_\ell)$ and $T_i(y_\ell)$, respectively). From Invariant Q3, $|L_i(x_\ell)| = |L_i(y_\ell)| = \lambda^{i-1}$. Let Π^i_ℓ be an arbitrary complete matching between the vertices of $L_i(x_\ell)$ and the vertices of $L_i(y_\ell)$. For every pair $(u,v) \in \Pi^i_\ell$, we set the demand $D^i_\ell(u,v) = \lambda$. Since λ is an integer, the resulting demand $\mathcal{D}^i_\ell = \left(\Pi^i_\ell, \left\{D^i_\ell(u,v)\right\}_{(u,v)\in\Pi^i_\ell}\right)$ is integral. We then let $\mathcal{D}^i = \left(\Pi^i, \left\{D^i(u,v)\right\}_{(u,v)\in\Pi^i}\right)$ be the demand obtained by taking the union of the demands $\mathcal{D}^i_\ell$ for all $(x_\ell, y_\ell) \in \mathcal{U}_i$. Next, we prove that the demand $\mathcal{D}^i$ is Δ'-restricted.

Observation 9.12 *Demand $\mathcal{D}^i$ is Δ'-restricted.*

Proof: Notice that demand $\mathcal{D}^i$ is only defined over the vertices of $V(F)$. Consider any such vertex v. From Invariant Q4, the number of copies of v that appear in the sets $L_i(w)$ for $w \in V'_i$ is at most $f\eta'$. Each such copy appears in a single demand pair with demand λ. Therefore, the total demand on vertex v in $\mathcal{D}^i$ is bounded by $\lambda \cdot f \cdot \eta' \leq \Delta'$, since $\lambda = \left\lfloor \frac{\Delta'}{f\cdot\eta'} \right\rfloor$. We conclude that demand $\mathcal{D}^i$ is Δ'-restricted. □

Since demand $\mathcal{D}^i$ is Δ'-restricted, from Corollary 9.9, there is an integral routing $\mathcal{Q}^i$ of $\mathcal{D}^i$ in G via paths of length at most d, that cause congestion at most η'.

Consider now a pair $(x_\ell, y_\ell) \in \mathcal{U}_i$, and a corresponding demand $\mathcal{D}^i_\ell$. Let $\mathcal{Q}^i_\ell \subseteq \mathcal{Q}_i$ be the collection of λ^i paths routing the demand $\mathcal{D}^i_\ell$. We say that the pair (x_ℓ, y_ℓ) is *good*, if at least one path $P \in \mathcal{Q}^i_\ell$ is disjoint from $E(F)$, and we say that it is bad otherwise.

Consider a single good pair $(x_\ell, y_\ell) \in \mathcal{U}_i$, and let $P'_\ell \in \mathcal{Q}^i_\ell$ be any path that is disjoint from $E(F)$. Denote the endpoints of P'_ℓ by w and w', where $w \in L_i(x_\ell)$, and $w' \in L_i(y_\ell)$, respectively. Let $\hat{P}_1$ be the path in the tree $T_i(x_\ell)$, connecting x_ℓ to w, so the number of edges on $\hat{P}_1$ is at most i. We obtain a path P''_ℓ in graph $G \setminus F$ connecting x_ℓ to w, by replacing every edge $e \in E(\hat{P}_1)$ with its embedding path $R(e) \in \mathcal{R}_i$ given in Invariant Q5. Since the length of every path $R(e)$ is at most d, we get that the length of P''_ℓ is a tmost id. We use a similar procedure to compute a path P'''_ℓ in $G \setminus F$ connecting w' to y_ℓ. By concatenating the paths P''_ℓ, P'_ℓ and P'''_ℓ, we obtain a path P_ℓ in $G \setminus F$ connecting x_ℓ to y_ℓ, whose length is at most $3id$. Consider now the collection $\mathcal{P}'_i$ of paths, that contains a path P_ℓ for every good pair $(x_\ell, y_\ell) \in \mathcal{U}_i$. From our discussion, all such paths are contained in $G \setminus F$, and each such path has length at most $3id$. Since, from Invariant Q5, the paths in $\mathcal{R}_i$ cause congestion at most $\eta' \cdot i$, and since the routing $\mathcal{Q}_i$ has congestion at most η', it is immediate to verify that the paths in $\mathcal{P}'_i$ cause congestion at most $(i+1)\eta'$. We let set $\mathcal{S}_{i+1}$ contain all pairs of $\mathcal{S}_i$, and all good pairs that lie in $\mathcal{U}_i$. We also let $\mathcal{P}_{i+1} = \mathcal{P}_i \cup \mathcal{P}'_i$. From Invariant Q1 and our discussion above, the set $\mathcal{P}_{i+1}$ of paths causes congstion at most $2(i+1) \cdot \eta'$, and so Invariant Q1 continues to hold.

We let $\mathcal{U}_{i+1} \subseteq \mathcal{U}_i$ be the set of all bad pairs (x_ℓ, y_ℓ), and we let V'_{i+1} be the multisetset of all vertices that participate in the pairs in U_{i+1}. For every vertex $v \in V'_{i+1}$, we extend the tree $T_i(v)$, in order to obtain the tree $T_{i+1}(v)$ with the required properties.

Consider a vertex $v \in V'_{i+1}$, and assume that it belongs to a pair $(x_\ell, y_\ell) \in U_{i+1}$. Let $L = L_i(v)$ be the set of all leaf vertices of the tree $T_i(v)$, and let $u \in L$ be any such vertex. Recall that there is some pair $(u,w) \in \Pi^i_\ell$ in which u participates, and moreover, this pair is assigned demand λ in $\mathcal{D}^i_\ell$. Therefore, there is a collection $\mathcal{Q}(u) \subseteq \mathcal{Q}^i_\ell$ of λ paths that originate at u. Consider any such path $Q \in \mathcal{Q}(u)$, and recall that it must contain an edge of F. Let $u' \in V(F) \cap V(Q)$ be the vertex lying closes to u on path Q, and let $R(u,u')$ be the subpath of Q from u to u'. Then $R(u,u')$ is disjoint from F, and $u' \in V(F)$. We add vertex u' as a child vertex of u in the tree $T_i(v)$, and we let the embedding path of the new edge (u,u') be the path $R(u,u')$ defined above; note that the length of the path is at most d. Once we process all leaf vertices of $T_i(v)$, we obtain a new tree $T_{i+1}(v)$, that remains a λ-regular tree, whose

depth is $i+1$. From our construction, its set of leaf vertices $L_{i+1}(v)$ has cardinality $\lambda \cdot |L_i(v)| = \lambda^i$. For every edge e of the tree, if it lied in $T_i(v)$, then its embedding path $R(e) \in \mathcal{R}_i$ is as defined in Invariant Q5, and otherwise, its embedding path is a subpath of one of the routing paths in $\mathcal{Q}^i$ as defined above. Since the paths in $\mathcal{R}_i$ caused congestion at most $i\eta'$, while the paths in $\mathcal{Q}$ cause congestion at most η', it is easy to verify that the resulting set $\mathcal{R}_{i+1} = \left\{R(e) \mid e \in E(T_{i+1}(v)), v \in V'_{i+1}\right\}$ cause congestion at most $(i+1) \cdot \eta'$.

So far we have shown that Invariants Q1–Q3 and Q5 hold. It now only remains to establish Invariant Q4. Indeed, recall that the collection $\mathcal{Q}^i$ of paths causes congestion at most η'. Consider any vertex $u \in V(F)$. The number of times that u appears in the sets $L_{i+1}(v)$ of vertices for $v \in V'_{i+1}$ is bounded by the number of paths $Q \in \mathcal{Q}$, such that Q contains an edge of the faulty set F that is incident to u. Since at most f edges incident to u may lie in F, and since each such edge participates in at most η' paths, vertex u may appear at $f \cdot \eta'$ times in the sets $\{L_{i+1}(v)\}_{v \in V'_{i+1}}$.

Note that, when the algorithm terminates, which must happen after $i \leq 10k$ iterations, all pairs of I' lie in the set $\mathcal{S}_i$ of i-safe pairs. From Invariant Q1, we conclude that there is an integral routing of the pairs in I' in graph $G \setminus F$ via paths of length at most $O(k) \cdot d$, that cause congestion at most $O(k) \cdot \eta'$, as required. □

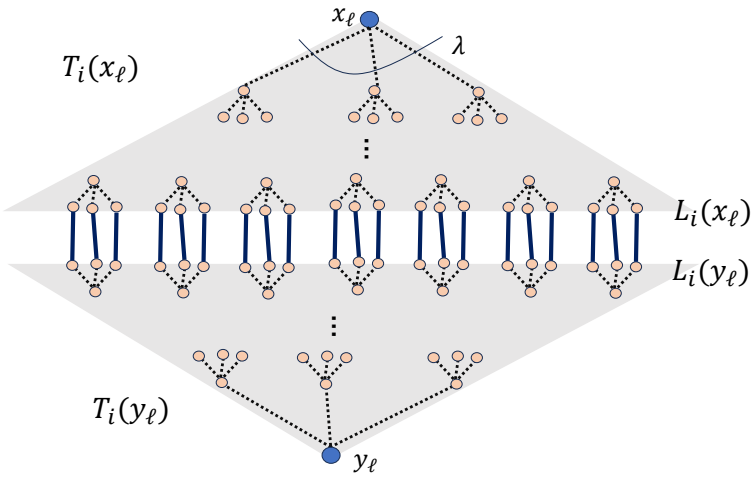

Figure 1: An illustration for the proof of Lemma 9.7. Shown is an i-unsafe pair (x_ℓ, y_ℓ) with their corresponding trees $T_i(x_\ell)$ and $T_i(y_\ell)$. The routing instance for iteration i is defined by sending λ flow units between matched leaf vertices in $L_i(x_\ell)$ and $L_i(y_\ell)$. The solid edges correspond to λ units of demand between the matched pair.

9.2.2 From Router Decomposition to Fault Tolerant Spanners

Fault-Tolerant Spanners. Recall that for a graph G, a stretch parameter $k \geq 1$, and an integer fault parameter $f \geq 1$, an f-FT k-spanner of G is a subgraph $H \subseteq G$ satisfying that, for every subset $F \subseteq E(G)$ of at most f edges, $H \setminus F$ is a k-spanner for $G \setminus F$.

Very recently, Bodwin, Haeupler and Parter [BHP24] introduced a considerably stronger notion of fault-tolerance, where only the degree of the faulty edge set, rather than its cardinality, is restricted. For a subset $F \subseteq E(G)$ of faulty edges, its *faulty-degree* $\deg(F)$ is the maximum, over all vertices v, of the number of edges of F incident to v. For example, if F is a matching, then $\deg(F) = 1$, while

$|F|$ may be as large as $n/2$. This naturally leads to the definition of faulty-degree spanners.

Definition 9.13 (Faulty-Degree Spanners) *[BHP24] Let G be a graph and let $H \subseteq G$ be a subgraph of G with $V(H) = V(G)$. For parameters f and $k > 0$, we say that H is an f-faulty-degree (FD) k-spanner for G if, for every subset $F \subseteq E(G)$ of edges with $\deg(F) \leq f$, for every pair u, v of vertices of G, $\mathsf{dist}_{H \setminus F}(u, v) \leq k \cdot \mathsf{dist}_{G \setminus F}(u, v)$ holds.*

In the following lemma, we show that, if H' is a graph obtained from a router decomposition of the graph G as before, then it is an f-FD k'-spanner, for appropriately chosen parameters f and k'.

Observation 9.14 (From Router Decomposition to f-FD Spanners) *Let G be an n-vertex graph, and let $k' > 0$ and $f \in \{1, \ldots, n\}$ be parameters. Let $\mathcal{R} = (\mathcal{C}, \{C'\}_{C \in \mathcal{C}})$ be a router decomposition of G with parameters $\tilde{d}$, $1 \leq \tilde{\eta} \leq n^2$, $\Delta^* \geq 32 \cdot f \cdot \tilde{\eta} \cdot n^{1/k'}$, and ρ. Consider the corresponding graph H' whose vertex set is $V(G)$, and edge set is $E^{\mathrm{del}} \cup \left(\bigcup_{C \in \mathcal{C}} E(C')\right)$, where $E^{\mathrm{del}} = E(G) \setminus \left(\bigcup_{C \in \mathcal{C}} E(C)\right)$. Then H' is an f-FD $O(\tilde{d} \cdot k')$-spanner.*

Proof: Consider an f-BD faulty set $F \subseteq E(G)$ of edges, and an edge $e = (u, v) \in E(G) \setminus F$. If $e \in E^{\mathrm{del}}$, then e is included in H', and so $\mathsf{dist}_{H' \setminus F}(u, v) = 1$. Otherwise, there must be some cluster $C \in \mathcal{C}$, with $u, v \in C$. Consider the corresponding subgraph $C' \subseteq C$, and recall that $V(C) = V(C')$, and C' is a $(\overline{\Delta}^*, \tilde{d}, \tilde{\eta})$-router for the set $V(C)$ of supported vertices. Then from Theorem 9.7, graph $C' \setminus F$ remains a $(\overline{\Delta}^*, O(k \cdot \tilde{d}), O(k \cdot \tilde{\eta}))$-router. In particular, the diameter of $C' \setminus F$ is bounded by $O(k'\tilde{d})$ which implies that $\mathsf{dist}_{C' \setminus F}(u, v) = O(k'\tilde{d})$. Since $E(C') \subseteq H'$, the observation follows. □

Fault-Tolerant Low-Congestion Spanners. Finally, we show the strongest property of subgraph H maintained by the algorithm from Theorem 1.3. We will use the fact that routers retain their routing properties (with a small loss), to show that H is a fault-tolerant low-congestion spanner even against bounded degree faults. We start by formally defining the latter notion.

Definition 9.15 (Fault-Tolerant Low-Congestion Spanners) *Let G be a graph, let $H \subseteq G$ be a subgraph of G, and let f, d and η be parameters. We say that H is an f-FT (d, η)-low congestion spanner if, for every subset $F \subseteq E(G)$ of edges with $|F| \leq f$, graph $G \setminus F$ embeds into $H \setminus F$ via paths of length at most d, with congestion at most η. Similarly, H is an f-FD (d, η)-low congestion spanner if the above holds for every subset $F \subseteq E(G)$ of edges with $\deg(F) \leq f$.*

In the following observation we show that the graph H' obtained from a router decomposition of G with an appropriately chosen parameter f must be an f-FD low-congestion spanner.

Observation 9.16 (From Router Decomposition to f-FD Low-Congestion Spanners) *Let G be an n-vertex graph, and let $k' > 0$ and $f \in \{1, \ldots, n\}$ be parameters. Let $\mathcal{R} = (\mathcal{C}, \{C'\}_{C \in \mathcal{C}})$ be a router decomposition of G with parameters $\tilde{d}$, $1 \leq \tilde{\eta} \leq n^2$, $\Delta^* \geq 32 \cdot f \cdot \tilde{\eta} \cdot n^{1/k'}$, and ρ. Consider the corresponding graph H' whose vertex set is $V(G)$, and edge set is $E^{\mathrm{del}} \cup \left(\bigcup_{C \in \mathcal{C}} E(C')\right)$, where $E^{\mathrm{del}} = E(G) \setminus \left(\bigcup_{C \in \mathcal{C}} E(C)\right)$. Then H' is an f-FD (d, η)-low congestion spanner for $d = O(k' \cdot \tilde{d})$ and $\eta = \max\left\{O\left(\frac{k' \cdot \tilde{\eta} \cdot \Delta_{\max}(G)}{\Delta^*}\right), 16 \log n\right\}$, where $\Delta_{\max}(G)$ is maximum vertex degree in G.*

Proof: Let $F \subseteq E(G)$ be a set of edges with $\deg(F) \leq f$. Our goal is to show that $H' \setminus F$ is a (d, η)-low congestion spanner for $G \setminus F$. In other words, we need to show that $G \setminus F$ can be embedded into $H' \setminus F$ via paths of length at most d and congestion at most η. Since $E^{\mathrm{del}} \subseteq E(H')$, and the clusters of $\mathcal{C}$ are edge-disjoint, it is sufficient to show that, for every cluster $C \in \mathcal{C}$, there is an embedding $\mathcal{Q}_C$ of $C \setminus F$ into $C' \setminus F$ via paths of length at most d, that cause congestion at most η.

Since, for every cluster $C \in \mathcal{C}$, the corresponding graph $C' \subseteq C$ is a $(\overline{\Delta}^*, \tilde{\eta}, \tilde{d})$ router for the set $V(C) = V(C')$ of supported vertices, and $\Delta^* \geq 32fn^{1/k'} \cdot \tilde{\eta}$ holds, from Theorem 9.7 we get that that $C' \setminus F$ is a $(\Delta^*, O(k' \cdot \tilde{d}), O(k' \cdot \tilde{\eta}))$-router. In particular, $\left(\mathcal{C}, \{C' \setminus F\}_{C \in \mathcal{C}}\right)$ remains a router decomposition of G with parameters $\Delta^*, O(k') \cdot \tilde{d}$, $O(k') \cdot \tilde{\eta}$, and ρ. From Lemma 9.4, there is an embedding $\mathcal{Q}_C$ of C into C' via paths of length at most $O(k' \cdot \tilde{d})$, that cause congetion at most $\max\left\{O\left(\frac{k' \cdot \tilde{\eta} \cdot \Delta_{\max}(G)}{\Delta^*}\right), 16 \log n\right\}$. □

Finally, we obtain the following analogue of Corollaries 9.2 and 9.6 for f-FD low-congestion spanners.

Corollary 9.17 *There is a deterministic algorithm, whose input is a graph G with n vertices and no edges, that undergoes an online sequence of at most n^2 edge deletions and insertions, such that G remains a simple graph throughout the update sequence. Each edge inserted into G has an integral length $\ell(e) \in \{1, \ldots, L\}$. Additionally, the algorithm is given parameters $512 \leq k \leq (\log n)^{1/49}$, $\frac{1}{k} \leq \delta \leq \frac{1}{400}$, $f \in \{1, \ldots, n\}$, and $\Delta \geq f \cdot n^{c/k}$, for a large enough constant c, such that k, Δ and $\frac{1}{\delta}$ are integers. The algorithm maintains a subgraph $H \subseteq G$ with $V(H) = V(G)$ and $|E(H)| \leq O\left(n^{1+O(1/k)} \cdot \Delta \cdot \log L\right)$, such that H is an a f-FD (d, η)-low congestion spanner for G, for $d \leq \mathrm{poly}(k) \cdot 2^{O(1/\delta^6)}$ and $\eta \leq n^{O(1/k)} \cdot \max\left\{\frac{\Delta_{\max}(G)}{\Delta}, 1\right\}$, where $\Delta_{\max}(G)$ is the maximum degree in G. The worst-case update time of the algorithm is $n^{O(\delta)}$ per operation, and the number of edge insertions and deletions that graph H undergoes after each update to G is bounded by $n^{O(1/k)}$.*

In order to prove the corollary, we use the algorithm from Corollary 9.6. By using Observation 9.16 with parameter k' replaced with k, it is easy to verify that the graph H that the algorithm maintains has all required properties.

Dynamic Connectivity Certificates. The notion of connectivity certificates was introduced by [NI92]. For a graph $G = (V, E)$ and integer $f \in \{1, \ldots, n\}$, an f-connectivity certificate is a subgraph $H \subseteq G$ with the following property: for every pair $u, v \in V$ of vertices, for every subset $F \subseteq E$ of at most f edges, u and v are connected in $H \setminus F$ if and only if they are connected in $G \setminus F$. The recent work of [BHP24] introduced a stronger notion of f-FD connectivity certificate: a subgraph $H \subseteq G$ that satisfies the above requirement for any subset $F \subseteq E$ of edges with $\deg(F) \leq f$. It is well known that any n-vertex graph admits an f-connectivity certificate containing $O(fn)$ edges, and such certificates can be computed in linear time in the static setting. Using expander decompositions and expander routing, [BHP24] presented an algorithm that constructs an f-FD connectivity certificates with $\widetilde{O}(fn)$ edges, which is nearly tight, in polynomial time.

Clearly, if a graph H is an f-FD (d, η)-low congestion spanner for G, for any parameters d and η, then H is an f-FD connectivity certificate and an f-connectivity certificate for G as well. Therefore, by using the algorithm from Corollary 9.17 with a parameter $\Delta = f \cdot n^{O(1/k)}$ and $\delta = \frac{1}{k}$, we get that the graph H that the algorithm maintains has $|E(H)| \leq O\left(f \cdot n^{1+O(1/k)}\right)$, and it is an f-FD connectivity certificate for G. The algorithm has $n^{O(1/k)}$ worst-case update time and $n^{O(1/k)}$ recourse. Note that both of the latter quantities are independent of f. To the best of our knowledge, all previous dynamic algorithms for (the classical notion of) f-connectivity certificates had a worst-case update time with a polynomial dependency on f. Therefore, for high values of the parameter f, we obtain a faster algorithm.

A Proof of Claim 2.6

Let G be an n-vertex graph, and assume that it is (d,η)-well-connected with respect to a set $S\subseteq V(G)$ of supported vertices. Let $1\leq k\leq \log n$ be a parameter. Our goal is to prove that G is a $(1,d',\eta')$-router with respect to S.

Consider any demand $\mathcal{D}=\left(\Pi,\{D(a,b)\}_{(a,b)\in\Pi}\right)$. We say that $\mathcal{D}$ is a *matching* demand, if Π is a matching, and, for all $(a,b)\in\Pi$, $D(a,b)=1$. Notice that any matching demand can be specified by just the set Π of pairs of its vertices, so for a matching demand $\mathcal{D}$, abusing the notation, we may denote $\mathcal{D}=\Pi$. We show that it is enough to prove that any matching demand can be routed via paths of length at most d' with congestion at most $\frac{\eta'}{16\log n}$, in the following simple claim.

Claim A.1 *Let G be an n-vertex graph with a set $S\subseteq V(G)$ of supported vertices, and assume that, for every matching demand $\mathcal{D}$ defined over S, there is a routing of $\mathcal{D}$ in G via paths of length at most $\hat{d}$ and congestion at most $\hat{\eta}$. Then G is a $(\overline{1},\hat{d},16\hat{\eta}\cdot\log n)$-router with respect to S.*

Proof: Consider any 1-restricted demand $\mathcal{D}=\left(\Pi,\{D(a,b)\}_{(a,b)\in\Pi}\right)$ over the vertices of S. Let $q=\lceil\log n\rceil+1$. For all $1\leq i<q$, we define a demand $\mathcal{D}_i=\left(\Pi_i,\{D'(a,b)\}_{(a,b)\in\Pi_i}\right)$ as follows: set Π_i contains all pairs (a,b) with $\frac{1}{2^i}\leq D(a,b)<\frac{1}{2^{i-1}}$, and $D'(a,b)=\frac{1}{2^i}$. Additionally, we define a demand $\mathcal{D}_q=\left(\Pi_q,\{D'(a,b)\}_{(a,b)\in\Pi_q}\right)$, where Π_q contains all pairs (a,b) with $D(a,b)<\frac{1}{n}$, and we set $D'(a,b)=\frac{1}{n}$. It is easy to verify that, for all $1\leq i\leq q$, demand $\mathcal{D}_i$ is 1-restricted. Moreover, it is now enough to prove that each such demand $\mathcal{D}_i$ can be routed in G via paths of length at most $\hat{d}$ and congestion at most $4\hat{\eta}$.

From now on we focus on any such demand $\mathcal{D}_i$, that we denote by $\mathcal{D}'=\left(\Pi',\{D'(a,b)\}_{(a,b)\in\Pi'}\right)$ for convenience. Recall that there is an integer z, so that $D'(a,b)=\frac{1}{z}$ for all $(a,b)\in\Pi'$, and that demand $\mathcal{D}'$ is 1-restricted. Therefore, every vertex of S participates in at most z pairs in Π'. We can then compute a collection $\Pi'_1,\ldots,\Pi'_{2z+1}$ of pairs of vertices that partition Π', such that, for all $1\leq j\leq 2z+1$, set Π'_j is a matching, using a standard greedy algorithm.

From our assumption, for all $1\leq j\leq 2z+1$, there is a flow f_j that routes the matching demand between the pairs of vertices in Π'_j, where, for each pair $(a,b)\in\Pi'_j$ of vertices, one flow unit is routed from a to b in f_j. Moreover, all flow-paths in f_j have length at most $\hat{d}$, and the flow causes congestion at most $\hat{\eta}$. By scaling each such flow f_j down by factor z (so it causes congestion at most $\frac{\hat{\eta}}{z}$), and taking the union of all such flows, for all $1\leq j\leq 2z+1$, we obtain a flow f that routes the demand $\mathcal{D}'$ via paths of length at most $\hat{d}$, with congestion at most $4\hat{\eta}$. □

From now on, it is enough to prove that, for every matching demand $\mathcal{D}$ defined over S, there is a routing of $\mathcal{D}$ in G via paths of length at most d', with congestion at most $\frac{\eta'}{16\log n}$.

Consider any matching demand $\mathcal{D}=\Pi$. We say that $\mathcal{D}$ is *half-routable*, if there is a subset $\Pi'\subseteq\Pi$ with $|\Pi'|\geq\frac{|\Pi|}{2}$, such that there is a routing of Π' via paths of length at most d', and congestion at most $\frac{\eta'}{32\log^2 n}$. It is easy to see that, if every matching demand $\mathcal{D}$ defined over the set S of vertices is half-routable in G, then G is a $(\overline{1},d',\eta')$-router with respect to S, as we show in the next observation.

Observation A.2 *Suppose that every matching demand $\mathcal{D}$ defined over the vertices of S is half-routable in G. Then G is a $(\overline{1},d',\eta')$-router with respect to S.*

Proof: We assume that every matching demand defined over the vertices of S is half-routable in G.

From Claim A.1, it is enough to prove that, for every matching demand $\mathcal{D}$ defined over S, there is a routing of $\mathcal{D}$ in G via paths of length at most d', that cause congestion at most $\frac{\eta'}{16\log n}$.

Let $\mathcal{D}=\Pi$ be any matching demand. We perform at most $\lceil \log n\rceil$ iterations, where in the ith iteration we route a subset $\Pi_i\subseteq\Pi$ of the demand pairs, which are then deleted from Π.

Specifically, the input to the ith iteration is the current set Π of pairs. Since any matching demand is half-routable, there is a subset $\Pi_i\subseteq\Pi$, containing at least $\frac{|\Pi|}{2}$ pairs, and a routing f_i of Π_i in G via paths of length at most d', that causes congestion at most $\frac{\eta'}{32\log^2 n}$. We then delete the pairs of Π_i from Π, and continue to the next iteration. It is immediate to verify that, after at most $\lceil \log n\rceil$ such iterations, set Π becomes empty, and the union of the resulting flows f_i provides a routing of Π, via paths of length at most d', and congestion at most $\lceil \log n\rceil\cdot\frac{\eta'}{32\log^2 n}\leq\frac{\eta'}{16\log n}$. □

In order to complete the proof of Claim 2.6, it is now enough to prove that every matching demand $\mathcal{D}$ defined over the set S of supported vertices is half-routable. Assume otherwise, and let $\mathcal{D}=\Pi$ be a matching demand that is not half-routable. Consider the following iterative process for constructing a collection $\mathcal{P}$ of paths routing some pairs in Π. We start from $\mathcal{P}=\emptyset$, and then iterate, as long as there is a pair $(a,b)\in\Pi$, for which there is a path of length at most d' connecting a to b in G. In each iteration, we select any such pair $(a,b)\in\Pi$, and any path P of length at most d' connecting a to b in G, add P to $\mathcal{P}$, delete the pair (a,b) from Π, and delete from G every edge that has been used in at least $\frac{\eta'}{32\log^2 n}$ of the paths in $\mathcal{P}$.

Let Π' be the collection of paths that remain in Π at the end of this process. Let E' be the set of edges deleted from G, and let $G'=G\setminus E'$. Recall that an edge e lies in E' if and only if it has been used by at least $\frac{\eta'}{32\log^2 n}$ of the paths in $\mathcal{P}$. Therefore, $|E'|\leq\frac{|\Pi|\cdot d'\cdot 32\log^2 n}{\eta'}$. Note also that $\mathcal{P}$ is a routing of the pairs in $\Pi\setminus\Pi'$ via paths of length at most d', that cause congestion at most $\frac{\eta'}{32\log^2 n}$. Since demand $\mathcal{D}$ is not half-routable, we get that $|\Pi'|\geq\frac{|\Pi|}{2}$, and, in particular, $|E'|\leq\frac{64|\Pi'|\cdot d'\cdot\log^2 n}{\eta'}$. Lastly, for every pair $(a,b)\in\Pi'$, $\mathsf{dist}_{G'}(a,b)\geq d'$.

We use the following claim that easily follows from Lemma 3.10 in [Chu23], by setting the parameter α to $1/2$ and parameter Δ to $64k$:

Lemma A.3 (Lemma 3.10 from [Chu23]) *There is an efficient algorithm, whose input is a graph G, a set $T\subseteq V(G)$ of z vertices called terminals, a distance parameter $\tilde{d}$, and another parameter $k>0$. The algorithm either computes a terminal $t\in T$ with $|B_G(t,64k\tilde{d})\cap T|>\frac{z}{2}$, or it computes two subsets T_1,T_2 of terminals, with $|T_1|=|T_2|$, such that $|T_1|\geq\frac{z^{1-1/k}}{3}$, and for every pair $t\in T_1$, $t'\in T_2$ of terminals, $\mathsf{dist}_G(t,t')\geq\tilde{d}$.*

We apply the lemma to graph G', with the set T of terminals containing all vertices that participate in the pairs of Π', so that $z=|T|=2|\Pi'|$, and parameter $\tilde{d}$ replaced by $2d$, and parameter k that remains unchanged. Recall that $d'=512kd$. Note that it is impossible that the algorithm from Lemma A.3 returns a terminal $t\in T$ with $|B_{G'}(t,64k\tilde{d})|>\frac{z}{2}$, since in this case it would mean that there is a pair $(a,b)\in\Pi'$ of vertices, with both $a,b\in B_{G'}(t,64k\tilde{d})$; in other words, $\mathsf{dist}_{G'}(a,b)\leq 256kd<d'$, which is impossible. Therefore, the algorithm from Lemma A.3 must return two equal-cardinality sets A,B of vertices, with $|A|,|B|\geq\frac{z^{1-1/k}}{3}\geq\frac{|\Pi'|}{3n^{1/k}}$, such that $\mathsf{dist}_{G'}(A,B)\geq 2d$.

Since graph G is (d,η)-well-connected with respect to S, there is a collection $\mathcal{Q}$ of $|A|$ paths in G, connecting vertices of A to vertices of B, so that the length of every path is at most d, and the paths cause congestion at most η. Since $\mathsf{dist}_{G'}(A,B)\geq 2d$, every path in $\mathcal{Q}$ must use an edge of E'. Therefore, $|E'|\geq\frac{|\mathcal{Q}|}{\eta}\geq\frac{|A|}{\eta}\geq\frac{|\Pi'|}{3\eta n^{1/k}}$ must hold. But, as we have established above, $|E'|\leq\frac{64|\Pi'|\cdot d'\cdot\log^2 n}{\eta'}$. Since $\eta'=\left(2^{17}\cdot kdn^{1/k}\log^2 n\right)\cdot\eta=\left(2^8\cdot d'n^{1/k}\log^2 n\right)\cdot\eta$, we get that $|E'|\leq\frac{|\Pi'|}{4\eta n^{1/k}}$, a contradiction.

Therefore, every matching demand defined over the set S of supported vertices must be routable in G, and so G is a $(\overline{1}, d', \eta')$-router with respect to S.

B Proof of Corollary 3.3

Consider the input graph G. Since $|E(G)| \leq \Delta \cdot n^{1+15/k}$, the average vertex degree in G is at most $2\Delta \cdot n^{15/k}$. Note that, since $k \leq (\log n)^{1/49}$ holds, we get that $k^{45} \leq \frac{\log n}{k^4}$, and so:

$$2^{k^{45}} \leq n^{1/k^4}. \tag{19}$$

We use the algorithm from Claim 2.13 to compute, in time $O(|E(G)|) \leq O\left(n^{1+O(1/k)} \cdot \Delta\right) \leq O(n^{1+O(\delta)} \cdot \Delta)$, a vertex-split graph G' for G, such that the degrees of all vertices in G' are at most $4n^{15/k} \cdot \Delta$, and $|V(G')| \leq 2|V(G)|$. For convenience, we denote $\hat{n} = |V(G')|$, so $n \leq \hat{n} \leq 2n$. Recall that every edge $e = (u, v) \in E(G)$ corresponds to a unique edge $e' = (u', v')$ of G', where u' is a copy of u, and v' is a copy of v; we do not distinguish between these edges. We apply the algorithm from Theorem 3.2 to the resulting graph G', with the same parameters k, δ, Δ^*, and parameter $\lceil \Delta \rceil$ replacing Δ. Notice that $512 \leq k \leq (\log n)^{1/49} \leq (\log \hat{n})^{1/49}$, $\frac{1}{k} \leq \delta < \frac{1}{400}$, and $k, \Delta^*, \lceil \Delta \rceil$ and $\frac{1}{\delta}$ are integers, as required. Moreover, $\hat{n}^{18/k} \leq (2n)^{18/k} \leq n^{19/k} \leq \Delta^*$ must hold, and, since $\hat{n}^{8/k^2} \leq (2n)^{8/k^2} \leq n^{9/k^2}$, we get that $\Delta^* \leq \frac{\Delta}{n^{9/k^2}} \leq \frac{\lceil \Delta \rceil}{\hat{n}^{8/k^2}}$ holds. Therefore, altogether, $\hat{n}^{18/k} \leq \Delta^* \leq \frac{\lceil \Delta \rceil}{\hat{n}^{8/k^2}} \leq \hat{n}$ must hold. The maximum vertex degree in G' is bounded by $4n^{15/k} \cdot \Delta \leq 4\hat{n}^{15/k} \cdot \Delta \leq \hat{n}^{16/k} \cdot \lceil \Delta \rceil$. Therefore, graph G', with parameters $\hat{n}$, k, δ, Δ^* and $\lceil \Delta \rceil$ is a valid input to the algorithm from Theorem 3.2. Consider the router decomposition $\left(\mathcal{C}, \{C'\}_{C \in \mathcal{C}}\right)$ of graph G' with parameters $\Delta^*, \tilde{d}, \tilde{\eta}$ and ρ, that the algorithm maintains, where $\tilde{d} = k^{19} \cdot 2^{O(1/\delta^6)}$, $\tilde{\eta} = \hat{n}^{17/k} \leq (2n)^{17/k} \leq n^{19/k}$, and $\rho = \hat{n}^{20/k} \leq (2n)^{20/k} \leq \frac{n^{21/k}}{2}$.

Consider a cluster $C \in \mathcal{C}$, and recall that it must be $(\overline{\deg}_C, \tilde{d}, \tilde{\eta})$-router with respect to the set $V(C)$ of supported vertices. Let $\hat{C} \subseteq G$ be the graph obtained by including all the edges of C, and all vertices whose copies lie in C. It is then easy to verify that C is a vertex-split graph of $\hat{C}$. From Observation 2.14, graph $\hat{C}$ is a $(\overline{\deg}_{\hat{C}}, \tilde{d}, \tilde{\eta})$-router with respect to the set $V(\hat{C})$ of supported vertices. Let $\hat{\mathcal{C}} = \left\{\hat{C} \mid C \in \mathcal{C}\right\}$. It is immediate to verify that all clusters in $\hat{\mathcal{C}}$ are edge disjoint, and that every edge $e \in E(G) \setminus E^{\mathrm{del}}$ lies in some cluster of $\hat{\mathcal{C}}$. It is also immediate to verify that $\sum_{\hat{C} \in \hat{\mathcal{C}}} |V(\hat{C})| \leq \sum_{C \in \mathcal{C}} |V(C)| \leq \rho \cdot \hat{n} \leq 2\rho n$.

Consider again a cluster $C \in \mathcal{C}$, and the corresponding subgraph $C' \subseteq C$. We can again define a subgraph $\hat{C}' \subseteq \hat{C}$, by including in it all edges of C', and all vertices of G whose copies lie in C'. Since graph C' is a $(\overline{\Delta}^*, \tilde{d}, \tilde{\eta})$-router with respect to the set $V(C')$ of supported vertices, from Observation 2.15, $\hat{C}'$ is a $(\overline{\Delta}^*, \tilde{d}, \tilde{\eta})$-router with respect to the set $V(\hat{C}')$ of supported vertices. Therefore, throughout the algorithm $\left(\hat{\mathcal{C}}, \left\{\hat{C}'\right\}_{\hat{C} \in \hat{\mathcal{C}}}\right)$ is a router decomposition of G, for parameters $\Delta^*, \tilde{d}, \tilde{\eta}$ and $2\rho \leq n^{21/k}$.

Once the algorithm from Theorem 3.2 initializes the router decomposition $(\mathcal{C}, \{C'\}_{C \in \mathcal{C}})$ of G', we can initialize the router decomposition $\left(\hat{\mathcal{C}}, \left\{\hat{C}'\right\}_{\hat{C} \in \hat{\mathcal{C}}}\right)$ of G within the same asymptotic running time. Recall that the algorithm guaranees that initially, $|E^{\mathrm{del}}| \leq \hat{n} \cdot \lceil \Delta \rceil \cdot \hat{n}^{2/k^2} \leq n^{1+4/k^2} \cdot \Delta$.

For all $1 \leq i \leq k$, once the algorithm from Theorem 3.2 processes the batch π_i of edge deletions from G' and updates the clustering $\mathcal{C}$ and the subgraphs $C' \subseteq C$ for $C \in \mathcal{C}$ accordingly, we update the clustering $\hat{\mathcal{C}}$ and the graphs $\hat{C}' \subseteq \hat{C}$ for $\hat{C} \in \hat{\mathcal{C}}$ similarly: whenever an edge is deleted from a cluster $C \in \mathcal{C}$, we delete it from $\hat{C}$; whenever an edge is deleted from or inserted into a graph C' for some

cluster $C \in \mathcal{C}$, we insert or delete the same edge from the corresponding graph $\hat{C}'$; and whenever a vertex v is deleted from some cluster C, if v is a copy of some vertex $v' \in V(G)$, and no other copies of v' remain in C, then we delete v' from $\hat{C}$ and $\hat{C}'$ as well. It is easy to verify that the time required to update the router decomposition $\left(\hat{\mathcal{C}}, \left\{\hat{C}'\right\}_{\hat{C}\in\hat{\mathcal{C}}}\right)$ of G is asymptotically bounded by the time that the algorithm from Theorem 3.2 spends on processing the batch π_i of updates. Lastly, recall that, following the processing of batch π_i, the number of edges that may be added to E^{del} is bounded by $|\pi_i| \cdot \hat{n}^{13/k^2} \leq |\pi_i| \cdot (2n)^{13/k^2} \leq |\pi_i| \cdot n^{15/k^2}$, as required, and the number of edge insertions and deletions from the graphs in $\left\{\hat{C}' \mid \hat{C} \in \hat{\mathcal{C}}\right\}$ while batch π_i is processed is bounded by the number of edge insertions and deletions from the graphs in $\{C' \mid C \in \mathcal{C}\}$, which, in turn, is bounded by $|\pi_i| \cdot n^{O(1/k^2)}$.

C Proof of Claim 5.2

We say that a vertex $v \in V(G)$ is scattered if $|B_G(v,d)| < n^{1-\epsilon}$. We start with $G' = G$, and then delete all isolated vertices from G'. Over the course of the algorithm, we may delete additional vertices from G'.

The algorithm consists of a number of iterations. In each iteration, we may delete some vertices from G' (which must be scattered vertices), and we may mark some additional vertices of G' as scattered. Throughout, we denote by $U \subseteq V(G')$ the set of vertices that were not marked as scattered yet. We maintain the following invariants:

I1. If a vertex $v \in V(G)$ is not scattered, then $v \in U$ holds at all times; and

I2. For every vertex $u \in U$, $B_G(u,d) \subseteq G'$.

The algorithm continues as long as $U \neq \emptyset$ and $|V(G')| \geq n^{1-\epsilon}$ hold. If either of these conditions does not hold, then we are guaranteed that G is (d,ϵ)-scattered. Indeed, if G contains a vertex v with $|B_G(v,d)| \geq n^{1-\epsilon}$, then, from Invariant I1, $v \in U$, and from Invariant I2 $B_G(u,d) \subseteq G'$ must hold, so $U \neq \emptyset$ and $|V(G')| \geq n^{1-\epsilon}$ must hold. Therefore, if either $U \neq \emptyset$ or $|V(G')| \geq n^{1-\epsilon}$ hold, we terminate the algorithm and report that G is (d,ϵ)-scattered.

At the beginning of the algorithm, $U = V(G')$, and it is easy to see that both invariants hold. We now describe a single iteration. We assume that both invariants hold at the beginning of the iteration, and we will ensure that they hold at the end of the iteration.

At the beginning of the iteration, we delete from G' all isolated vertices. It is easy to verify that both invariants continue to hold. Next, we select an arbitrary vertex $u \in U$. We denote $L_0 = B_{G'}(u, 2d)$, and, for all $j > 0$, we denote by $L_j = B_{G'}(u, 2(j+1)d) \setminus B_{G'}(u, 2jd)$. We also denote by E_j the set of all edges with both endpoints in $L_0 \cup \cdots \cup L_j$.

We perform a BFS in G', starting from vertex u, until we encounter the first index j, for which $|E_{j+1}| \leq m^{\epsilon} \cdot |E_j|$. It is easy to verify that $1 \leq j \leq 2\lfloor 1/\epsilon \rfloor$ must hold, since otherwise, for all $1 \leq j \leq 2\lfloor 1/\epsilon \rfloor$, $|E_j| > m^{\epsilon} \cdot |E_{j-1}| \geq m^{(j-1)\cdot\epsilon}$, and so $|E_{2\lfloor 1/\epsilon \rfloor}| > m$, which is impossible.

Let $S = L_0 \cup \cdots \cup L_j$, let $S'' = L_0 \cup \cdots \cup L_{j+1}$, and let $S' = B_{G'}(u, 2(j+1)d + d)$. In other words, S' contains all vertices of S, and the subsequent d layers of the BFS. We now consider two cases. The first case happens if $|S''| \geq n^{1-\epsilon}$. In this case, we say the current iteration is good. Notice that $S'' \subseteq B_G(u, 4d/\epsilon)$. We then return vertex u as the outcome of the algorithm. The running time of the current iteration is bounded by $O(m)$ in this case.

Consider now the second case, where $|S''| < n^{1-\epsilon}$. In this case, we say that the iteration is bad. Notice that, from our invariants, we are guaranteed that, for every vertex $v \in S' \cap U$, $B_G(u,d) \subseteq B_{G'}(u,d)$, and moreover, since $B_{G'}(u,d) \subseteq S''$, we get that $|B_G(u,d)| \leq n^{1-\epsilon}$. Therefore, every vertex in S' is scattered. We delete the vertices of S from G, and we mark the vertices of $S' \setminus S$ as scattered, so they are deleted from U. Notice that, for every vertex x that remains in U, if $B_G(x,d) \subseteq B_{G'}(x,d)$ held at the beginning of the current iteration, then $B_G(x,d) \subseteq B_{G'}(x,d)$ continues to hold at the end of the current iteration, since $\mathsf{dist}_{G'}(x,S) > d$. The running time of the current iteration in the second case is bounded by $O(|E_{j+1}|) \leq O(\mathrm{Vol}_G(S) \cdot m^{\epsilon})$.

It now remains to analyze the running time of the algorithm. There is at most one good iteration in the algorithm, and its running time is $O(m)$. The running time of a bad iteration is $O(\mathrm{Vol}_G(S) \cdot m^{\epsilon})$, where S is the set of vertices deleted from G' in that iteration. Therefore, the total running time of the algorithm is $O(m^{1+\epsilon})$.

D Proofs Omitted from Section 8

D.1 Proof of Claim 8.5

Observe first that our algorithm ensures that the graph C has no isolated vertices. It also ensures that C is a large cluster, so $n' = |V(C)| \geq \Delta$ holds. From the definition of valid input parameters (see Definition 8.1), n is greater than a large enough constant, and $k \leq (\log n)^{1/49}$, so $n' > \Delta \geq n^{18/k} \geq 2^{18(\log n)^{48/49}}$ can also be assumed to be sufficiently large.

Since $n' \geq \Delta$, and since, from the definition of valid input graph and parameters, $\Delta \geq 16$, we get that $16 \leq \Delta \leq n'$ holds as required. From the definition of valid input parameters, we are also guaranteed that $512 \leq k \leq (\log n)^{1/49}$ holds. Since $n' > \Delta \geq n^{18/k} \geq n^{1/(4k^2)}$ (from the definition of valid input parameters), and $k \leq (\log n)^{1/49}$, we get that:

$$(\log n') \geq \frac{1}{4k^2} \cdot \log n \geq \frac{\log n}{4(\log n)^{1/20}} \geq (\log n)^{3/5}. \tag{20}$$

Since $\hat{k} = k^2$, we then get that:

$$\hat{k} \leq (\log n)^{1/20} \leq (\log n')^{1/3}.$$

Lastly, from the definition of valid input parameters, $1/\delta$ must be an integer, with $\frac{1}{k} \leq \delta \leq \frac{1}{400}$. Since $k \leq (\log n)^{1/49}$, we get that $\frac{1}{(\log n)^{1/49}} \leq \frac{1}{k} \leq \delta$, and so $\frac{2}{(\log n')^{1/24}} \leq \delta \leq \frac{1}{400}$, as required.

D.2 Proof of Claim 8.6

Recall that the algorithm for the NiceRouterPruning problem ensures that the total number of vertices that initially lied in $W^{N,\Delta}_{\hat{k}}$, but no longer lie in $U_{\hat{k}}$ at the end of Phase Φ_0 is bounded by:

$$\frac{|E'_0|}{\Delta} \cdot \hat{k}^{4\hat{k}} \leq \frac{N^{\hat{k}}}{2}.$$

Therefore, at the end of Phase Φ_0, graph W_C and set $U_{\hat{k}}$ contain at least $\frac{N^{\hat{k}}}{2}$ vertices. From the definition of a properly pruned subgraph (see Definition 4.1), graph W_C must contain at least $\frac{|U_k| \cdot \Delta}{4} \geq$

$\frac{N^{\hat{k}} \cdot \Delta}{8}$ edges.

Since $\mathcal{P}$ is an embedding of W_C into C', we get that:

$$|V(C')| \geq |V(W_C)| \geq \frac{N^{\hat{k}}}{2}$$

holds. Therefore, $N \leq (2|V(C')|)^{1/\hat{k}}$ holds. Additionally, since $N = \left\lfloor |V(C)|^{1/\hat{k}-1/\hat{k}^2} \right\rfloor$, we get that $N \geq \frac{1}{2} \cdot |V(C)|^{1/\hat{k}-1/\hat{k}^2} \geq \frac{1}{2} \cdot |V(C')|^{1/\hat{k}-1/\hat{k}^2}$. Since every edge of C' lies on some path of $\mathcal{P}_C$, and every vertex of C' participates in at least $\frac{\Delta}{n^{4/k^2}}$ such paths, we conclude that $\left(W_{\hat{k}}^{N,\Delta}, W_C, \mathcal{P}\right)$ is a valid router certificate for C'.

Since the edges of W_C are embedded into graph C' with congestion at most η^*, we get that:

$$|E(C')| \geq \frac{|E(W_C)|}{\eta^*} \geq \frac{N^{\hat{k}} \cdot \Delta}{8\eta^*}.$$

Recall that all vertex degrees in graph G are bounded by $\Delta \cdot n^{16/k}$, and that $N^{\hat{k}} \geq \frac{|V(C)|}{2^{\hat{k}} \cdot n^{1/\hat{k}}}$. Therefore:

$$|E(C)| \leq |V(C)| \cdot \Delta \cdot n^{16/k} \leq N^{\hat{k}} \cdot \Delta \cdot n^{17/k} \cdot 2^{k^2}.$$

Altogether, we get that:

$$|E(C')| \geq \frac{N^{\hat{k}} \cdot \Delta}{8\eta^*} \geq \frac{|E(C)|}{8\eta^* \cdot n^{17/k} \cdot 2^{k^2}} \geq \frac{|E(C)|}{n^{18/k}},$$

since, from Inequalty 15, $\eta^* \leq n^{5/k^2}$; we also used Inequality 13.

D.3 Proof of Claim 8.18

The claim easily follows from the following observation.

Observation D.1 *There is an algorithm, whose input is a connected bipartite graph $H = (X, Y, E)$, and parameters $z \geq 2$, $\hat{\Delta}, \gamma' \geq 1$ and $r > 4\gamma'$, such that, for every vertex $x \in X$, $\deg_H(x) \geq z \cdot \hat{\Delta}$, and, for every vertex $y \in Y$, $\deg_H(y) \leq \gamma' \cdot z \cdot \hat{\Delta}$. The algorithm computes a collection $S \subseteq X$ of at least $\frac{|X|}{2}$ vertices, and, for every vertex $x \in S$, a collection $E'(x) \subseteq \delta_H(x)$ of its incident edges with $|E'(x)| = \left\lceil \hat{\Delta} \right\rceil$, such that every vertex $y \in Y$ serves as an endpoint in at most $r \cdot \left\lceil \hat{\Delta} \right\rceil$ edges in $\bigcup_{x \in S} E'(x)$. The running time of the algorithm is $O(|E|)$.*

We provide the proof of the observation below, after we complete the proof of Claim 8.18 using it. We let $r = \frac{\hat{r}}{\lceil \log |X| \rceil}$; notice that $r > 4\gamma'$ must hold. We start with a graph $H' = H$, and perform at most $\lceil \log |X| \rceil$ iterations. In the ith iteration, we apply the algorithm from Observation D.1 to the current graph H', with the parameters $\hat{\Delta}, z, \gamma'$ and r that remain unchanged. Let $S_i \subseteq X$ be the collection of vertices computed by the algorithm, and, for every vertex $x \in S_i$, let $E'(x)$ be the correponding subset of its incident edges. We then delete from H' all vertices of S_i, and any additional vertices of Y that become isolated, and continue to the next iteration. It is immediate to verify that, after at most $\lceil \log(|X|) \rceil$ iteration, graph H' becomes empty, and, for every vertex $x \in X$, we obtain a collection

$E'(x)$ of its incident edges with $|E'(x)| = \left\lceil \hat{\Delta} \right\rceil$. Moreover, it is immediate to verify that every vertex $y \in Y$ serves as an endpoint in at most $r \cdot \left\lceil \hat{\Delta} \right\rceil \cdot \lceil \log(|X|) \rceil \leq \hat{r} \cdot \hat{\Delta}$ edges in $\bigcup_{x \in S} E'(x)$. Since the number of iteration is bounded by $O(\log(|X|))$, we get that the running time of the algorithm is $O(|E| \cdot \log(|X|))$.

It now remains to prove Observation D.1.

Proof of Observation D.1. Throughout the algorithm, we maintain a subset $S \subseteq X$ of vertices, starting with $S = \emptyset$. Whenever a vertex x is added to set S, we also define a collection $E'(x) \subseteq \delta_H(x)$ of its incident edges with $|E'(x)| = \left\lceil \hat{\Delta} \right\rceil$. Throughout the algorithm, we denote by $E' = \bigcup_{x \in S} E'(x)$. For every vertex $y \in Y$, we maintain a counter n_y, that counts the number of edges in E' incident to n_y. We say that vertex y is *overloaded*, if $n_y \geq (r-1) \cdot \left\lceil \hat{\Delta} \right\rceil$ holds. At the beginning of the algorithm, we initalize the counter n_y of every vertex $y \in Y$ to 0. We then process the vertices of X one by one. We now describe the processing of a vertex $x \in X$.

We consider the set $\delta_H(x)$ of edges incident to x in H, and we denote by $E''(x) \subseteq \delta_H(x)$ the subset of all such edges, whose endpont $y \in Y$ is not overloaded. If $|E''(x)| \geq \hat{\Delta}$, then we let $E'(x) \subseteq E''(x)$ be any subset containing $\left\lceil \hat{\Delta} \right\rceil$ edges. We add x to S, and, for every vertex $y \in Y$, that serves as an enpoint of at least one edge in $E'(x)$, we increase the counter n_y by the number of edges in $E'(x)$ for which y is an endpoint; notice that, since $|E'(x)| = \left\lceil \hat{\Delta} \right\rceil$, $n_y \leq r \cdot \left\lceil \hat{\Delta} \right\rceil$ continues to hold. Otherwise, if $|E''(x)| < \hat{\Delta}$, then we say that x is a *discarded vertex*. Once all vertices of X are processed, we consider the final set $S \subseteq X$, and, for every vertex $x \in S$, the subset $E'(x) \subseteq \delta_H(x)$ of its incident edges. Our algorithm ensures that $|E'(x)| = \left\lceil \hat{\Delta} \right\rceil$, and that every vertex $y \in Y$ serves as an endpoint of at most $r \cdot \left\lceil \hat{\Delta} \right\rceil$ edges in $E' = \bigcup_{x \in S} E'(x)$. It now only remains to prove that $|S'| \geq \frac{|X|}{2}$.

Assume for contradiction that this is not the case, and let $S'' = X \setminus S'$ be the set of all discarded vertices, so $|S''| \geq \frac{|X|}{2}$. Let Π be the collection of pairs (x, y), where $x \in S''$, and $y \in Y$ is an overloaded vertex, that is an endpoint of at least one edge in $\delta_H(x)$. From the definition of the discarded vertices, and since the degrees of all vertices in X are at least $z \cdot \hat{\Delta}$, we get that $|\Pi| \geq (z-1) \cdot \hat{\Delta} \cdot |S''| \geq \frac{z \cdot \lceil \hat{\Delta} \rceil \cdot |X|}{4}$. On the other hand, since every vertex $y \in Y$ has degree at most $\hat{\Delta} \cdot z \cdot \gamma'$ in H, we get that every overloaded vertex may participate in at most $\hat{\Delta} \cdot z \cdot \gamma'$ pairs in Π. Therefore, at least $\frac{|X|}{4\gamma'}$ vertices in Y are overloaded.

Recall that we have assumed that $|S| \leq \frac{|X|}{2}$, and so $|E'| = |S| \cdot \left\lceil \hat{\Delta} \right\rceil \leq \frac{|X| \cdot \lceil \hat{\Delta} \rceil}{2}$. If a vertex $y \in Y$ is overloaded, then the number of edges in E' that are incident to y is at least $(r-1) \cdot \left\lceil \hat{\Delta} \right\rceil$. Therefore, the number of overloaded vertices in Y is bounded by:

$$\frac{|E'|}{(r-1) \cdot \left\lceil \hat{\Delta} \right\rceil} \leq \frac{2|E'|}{r \cdot \left\lceil \hat{\Delta} \right\rceil} \leq \frac{|X|}{r} < \frac{|X|}{4\gamma'},$$

since we have assumed that $r > 4\gamma'$, a contradiction. □

References

[ACK+16] Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P Woodruff, and Qin Zhang. On sketching quadratic forms. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 311–319, 2016.

[ADF+07] Giorgio Ausiello, Camil Demetrescu, Paolo Giulio Franciosa, Giuseppe F. Italiano, and Andrea Ribichini. Small stretch spanners in the streaming model: New algorithms and experiments. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, volume 4698 of *Lecture Notes in Computer Science*, pages 605–617. Springer, 2007.

[AOSS19] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1919–1936. SIAM, 2019.

[AS23] Sepehr Assadi and Vihan Shah. Tight bounds for vertex connectivity in dynamic streams. In Telikepalli Kavitha and Kurt Mehlhorn, editors, *2023 Symposium on Simplicity in Algorithms, SOSA 2023, Florence, Italy, January 23-25, 2023*, pages 213–227. SIAM, 2023.

[Bas06] Surender Baswana. Dynamic algorithms for graph spanners. In Yossi Azar and Thomas Erlebach, editors, *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, volume 4168 of *Lecture Notes in Computer Science*, pages 76–87. Springer, 2006.

[Bas08] Surender Baswana. Streaming algorithm for graph spanners - single pass and constant processing time per edge. *Inf. Process. Lett.*, 106(3):110–114, 2008.

[BC18] Aaron Bernstein and Shiri Chechik. Incremental topological sort and cycle detection in expected total time. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 21–34. SIAM, 2018.

[BDPW18] Greg Bodwin, Michael Dinitz, Merav Parter, and Virginia Vassilevska Williams. Optimal vertex fault tolerant spanners (for fixed stretch). In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1884–1900. SIAM, 2018.

[BDR22] Greg Bodwin, Michael Dinitz, and Caleb Robelle. Partially optimal edge fault-tolerant spanners. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3272–3286. SIAM, 2022.

[BFH19] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1899–1918. SIAM, 2019.

[BHP24] Greg Bodwin, Bernhard Haeupler, and Merav Parter. Fault-tolerant spanners against bounded-degree edge failures: Linearly more faults, almost for free. In David P.

Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 2609–2642. SIAM, 2024.

[BK06] Surender Baswana and Telikepalli Kavitha. Faster algorithms for approximate distance oracles and all-pairs small stretch paths. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 591–602. IEEE Computer Society, 2006.

[BK16] Greg Bodwin and Sebastian Krinninger. Fully dynamic spanners with worst-case update time. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 17:1–17:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[BKS12] Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Trans. Algorithms*, 8(4):35:1–35:51, 2012.

[BS07] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Struct. Algorithms*, 30(4):532–563, 2007.

[BSS22] Sayan Bhattacharya, Thatchaphol Saranurak, and Pattara Sukprasert. Simple dynamic spanners with near-optimal recourse against an adaptive adversary. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 17:1–17:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[BvdBG+22] Aaron Bernstein, Jan van den Brand, Maximilian Probst Gutenberg, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, and He Sun. Fully-dynamic graph sparsifiers against an adaptive adversary. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 20:1–20:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[CGL+20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167. IEEE, 2020.

[CGP+20] Timothy Chu, Yu Gao, Richard Peng, Sushant Sachdeva, Saurabh Sawlani, and Junxing Wang. Graph sparsification, spectral sketches, and faster resistance computation via short cycle decompositions. *SIAM Journal on Computing*, 52(6):FOCS18–85, 2020.

[CHK16] Keren Censor-Hillel, Elad Haramaty, and Zohar S. Karnin. Optimal dynamic distributed MIS. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 217–226. ACM, 2016.

[Chu23] Julia Chuzhoy. A distanced matching game, decremental apsp in expanders, and faster deterministic algorithms for graph cut problems. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2122–2213. SIAM, 2023.

[CKL+22] Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 612–623. IEEE, 2022.

[CKP+17] Michael B. Cohen, Jonathan A. Kelner, John Peebles, Richard Peng, Anup B. Rao, Aaron Sidford, and Adrian Vladu. Almost-linear-time algorithms for markov chains and new spectral primitives for directed graphs. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 410–419. ACM, 2017.

[CKT93] Joseph Cheriyan, Ming-Yang Kao, and Ramakrishna Thurimella. Scan-first search and sparse certificates: An improved parallel algorithms for k-vertex connectivity. *SIAM J. Comput.*, 22(1):157–174, 1993.

[CLPR10] Shiri Chechik, Michael Langberg, David Peleg, and Liam Roditty. Fault tolerant spanners for general graphs. *SIAM J. Comput.*, 39(7):3403–3423, 2010.

[CZ19] Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 370–381. IEEE Computer Society, 2019.

[CZ23] Julia Chuzhoy and Ruimin Zhang. A new deterministic algorithm for fully dynamic all-pairs shortest paths. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 1159–1172. ACM, 2023.

[DFKL21] Michal Dory, Orr Fischer, Seri Khoury, and Dean Leitersdorf. Constant-round spanners and shortest paths in congested clique and MPC. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 223–233. ACM, 2021.

[DGPV08] Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. On the locality of distributed sparse spanner construction. In Rida A. Bazzi and Boaz Patt-Shamir, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*, pages 273–282. ACM, 2008.

[DHNS19] Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed edge connectivity in sublinear time. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 343–354. ACM, 2019.

[Din06] Yefim Dinitz. Dinitz' algorithm: The original version and Even's version. In *Theoretical computer science*, pages 218–240. Springer, 2006.

[DP09] Devdatt P Dubhashi and Alessandro Panconesi. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press, 2009.

[Elk07] Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and

Andrzej Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 716–727. Springer, 2007.

[EP01] Michael Elkin and David Peleg. Approximating k-spanner problems for k>2. In Karen I. Aardal and Bert Gerards, editors, *Integer Programming and Combinatorial Optimization, 8th International IPCO Conference, Utrecht, The Netherlands, June 13-15, 2001, Proceedings*, volume 2081 of *Lecture Notes in Computer Science*, pages 90–104. Springer, 2001.

[Erd64] P. Erdös. Extremal problems in graph theory. *Theory of Graphs and its Applications (Proc. Symp. Smolenice, 1963)*, pages 29–36, 1964.

[ES81] Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *Journal of the ACM (JACM)*, 28(1):1–4, 1981.

[EZ04] Michael Elkin and Jian Zhang. Efficient algorithms for constructing (1+, varepsilon;, beta)-spanners in the distributed and streaming models. In Soma Chaudhuri and Shay Kutten, editors, *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*, pages 160–168. ACM, 2004.

[FG19] Sebastian Forster and Gramoz Goranci. Dynamic low-stretch trees via dynamic low-diameter decompositions. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 377–388. ACM, 2019.

[FNY+20] Sebastian Forster, Danupon Nanongkai, Liu Yang, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Computing and testing small connectivity in near-linear time and queries via fast local cut algorithms. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2046–2065. SIAM, 2020.

[GHN+23] Gramoz Goranci, Monika Henzinger, Danupon Nanongkai, Thatchaphol Saranurak, Mikkel Thorup, and Christian Wulff-Nilsen. Fully dynamic exact edge connectivity in sublinear time. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 70–86. SIAM, 2023.

[GK13] Mohsen Ghaffari and Fabian Kuhn. Distributed minimum cut approximation. In Yehuda Afek, editor, *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2013.

[GK18] Mohsen Ghaffari and Fabian Kuhn. Derandomizing distributed algorithms with small messages: Spanners and dominating set. In Ulrich Schmid and Josef Widder, editors, *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*, volume 121 of *LIPIcs*, pages 29:1–29:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[GKS17] Mohsen Ghaffari, Fabian Kuhn, and Hsin-Hao Su. Distributed MST and routing in almost mixing time. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 131–140. ACM, 2017.

[GL18] Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. *arXiv preprint arXiv:1805.04764*, 2018.

[GLN+19] Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Deterministic graph cuts in subquadratic time: Sparse, balanced, and k-vertex. *arXiv preprint arXiv:1910.07950*, 2019.

[HHG22] Bernhard Haeupler, Jonas Huebotter, and Mohsen Ghaffari. A cut-matching game for constant-hop expanders. *arXiv preprint arXiv:2211.11726*, 2022.

[HHL+24] Bernhard Haeupler, D. Ellis Hershkowitz, Jason Li, Antti Roeyskoe, and Thatchaphol Saranurak. Low-step multi-commodity flow emulators. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 71–82. ACM, 2024.

[HHT23] Bernhard Haeupler, D. Ellis Hershkowitz, and Zihan Tan. Parallel greedy spanners. *CoRR*, abs/2304.08892, 2023.

[HHT24] Bernhard Haeupler, D Ellis Hershkowitz, and Zihan Tan. New structures and algorithms for length-constrained expander decompositions. *CoRR*, abs/2404.13446, 2024. FOCS 2024, to appear.

[HK95] Monika Rauch Henzinger and Valerie King. Fully dynamic biconnectivity and transitive closure. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 664–672. IEEE, 1995.

[HLS24] Bernhard Haeupler, Yaowei Long, and Thatchaphol Saranurak. Dynamic deterministic constant-approximate distance oracles with n^{ϵ} worst-case update time. *CoRR*, abs/2402.18541, 2024. FOCS 2024, to appear.

[HRG22] Bernhard Haeupler, Harald Räcke, and Mohsen Ghaffari. Hop-constrained expander decompositions, oblivious routing, and distributed universal optimality. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1325–1338, 2022.

[JS18] Arun Jambulapati and Aaron Sidford. Efficient õ (n/epsilon) spectral sketches for the laplacian and its pseudoinverse. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2487–2503. SIAM, 2018.

[JS22] Wenyu Jin and Xiaorui Sun. Fully dynamic st edge connectivity in subpolynomial time. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 861–872. IEEE, 2022.

[KLOS14] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 217–226, 2014.

[KM97] David R. Karger and Rajeev Motwani. An NC algorithm for minimum cuts. *SIAM J. Comput.*, 26(1):255–272, 1997.

[KMPG24] Rasmus Kyng, Simon Meierhans, and Maximilian Probst Gutenberg. A dynamic shortest paths toolbox: Low-congestion vertex sparsifiers and their applications. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1174–1183, 2024.

[KP12] Michael Kapralov and Rina Panigrahy. Spectral sparsification via random spanners. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 393–398. ACM, 2012.

[LNP+21] Jason Li, Danupon Nanongkai, Debmalya Panigrahi, Thatchaphol Saranurak, and Sorrachai Yingchareonthawornchai. Vertex connectivity in poly-logarithmic max-flows. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 317–329. ACM, 2021.

[LNS98] Christos Levcopoulos, Giri Narasimhan, and Michiel Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 186–195. ACM, 1998.

[LR99] F. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46:787–832, 1999.

[Mat93] David W. Matula. A linear time 2+epsilon approximation algorithm for edge connectivity. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 500–504. ACM/SIAM, 1993.

[NI92] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(5&6):583–596, 1992.

[NS17] Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, las vegas, and o (n1/2-ε)-time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1122–1129, 2017.

[NSW17] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 950–961, 2017.

[NSWN17] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 950–961. IEEE, 2017.

[Par22] Merav Parter. Nearly optimal vertex fault-tolerant spanners in optimal time: sequential, distributed, and parallel. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1080–1092. ACM, 2022.

[PS89] David Peleg and Alejandro A. Schäffer. Graph spanners. *J. Graph Theory*, 13(1):99–116, 1989.

[PU87] David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. In Fred B. Schneider, editor, *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10-12, 1987*, pages 77–85. ACM, 1987.

[PU89] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

[RT87] Prabhakar Raghavan and Clark D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[RTZ05] Liam Roditty, Mikkel Thorup, and Uri Zwick. Deterministic constructions of approximate distance oracles and spanners. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 261–272. Springer, 2005.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90, 2004.

[SW19] Thatchaphol Saranurak and Di Wang. Expander decomposition and pruning: Faster, stronger, and simpler. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2616–2635. SIAM, 2019.

[TK00] Mikkel Thorup and David R. Karger. Dynamic graph algorithms with applications. In Magnús M. Halldórsson, editor, *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, volume 1851 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2000.

[TZ01] M. Thorup and U. Zwick. Approximate distance oracles. *Annual ACM Symposium on Theory of Computing*, 2001.

[vdBCK+23] Jan van den Brand, Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *64th IEEE Annual Symposium on Foundations of Computer Science (FOCS), (to appear)*, 2023.

[WN17] Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1130–1143. ACM, 2017. Full version at arXiv:1611.02864.

[Wul17] Christian Wulff-Nilsen. Fully-dynamic minimum spanning forest with improved worst-case update time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1130–1143. ACM, 2017.