Routing in Undirected Graphs with Constant Congestion

Julia Chuzhoy†

November 20, 2015

Abstract

Given an undirected graph $G = (V,E)$, a collection $\{(s_1,t_1), \ldots, (s_k,t_k)\}$ of $k$ pairs of its vertices, called demand pairs, and an integer $c$, the goal in the Edge Disjoint Paths with Congestion problem is to connect maximum possible number of the demand pairs by paths, so that the maximum load on any edge (called edge congestion) does not exceed $c$.

We show an efficient randomized algorithm to route $\Omega(\frac{OPT}{\text{poly log } k})$ demand pairs with congestion at most 14, where OPT is the maximum number of pairs that can be simultaneously routed on edge-disjoint paths. The algorithm in fact routes $\Omega(\frac{OPT_{\text{LP}}}{\text{poly log } k})$ pairs, where OPT_{LP} is the optimal value of the standard multicommodity flow relaxation for the problem. The best previous efficient algorithm that routed $\Omega(\frac{OPT}{\text{poly log } n})$ pairs required congestion $\text{poly}(\log \log n)$, and for the setting where the maximum allowed congestion is bounded by a constant $c$, the best previous efficient algorithms could only guarantee the routing of $\Omega(\frac{OPT}{n^{1/c}})$ pairs.

We also introduce a new type of vertex sparsifiers that we call integral flow sparsifiers, that approximately preserve both fractional and integral routings, and show an algorithm to construct such sparsifiers.

---

* A preliminary version of this paper appeared in STOC 2012.
† Toyota Technological Institute, Chicago, IL 60637. Email: c@ttic.edu. Supported in part by NSF CAREER grant CCF-0844872, Sloan Research Fellowship and NSF grant CCF-1318242.
1 Introduction

We study network routing problems in undirected graphs. In such problems, we are given an undirected $n$-vertex graph $G = (V, E)$, and a collection $M = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$ of $k$ pairs of vertices of $G$, called source-sink pairs, that we also refer to as demand pairs. In order to route a pair $(s_i, t_i)$, we need to select a path connecting $s_i$ to $t_i$ in graph $G$. Given a routing of any subset of the demand pairs, its congestion is the maximum load on any edge, that is, the maximum number of paths containing the same edge. In general, we would like to route as many demand pairs as possible, while minimizing the edge congestion. These two conflicting objectives naturally give rise to several basic optimization problems.

One of the central routing problems is Edge Disjoint Paths (EDP), where the goal is to route maximum number of the demand pairs on edge-disjoint paths (that is, with congestion 1). Robertson and Seymour [RS90, RS95] have shown an efficient algorithm to solve this problem, when the number $k$ of the demand pairs is bounded by a constant. However, when $k$ is part of the input, it is NP-hard to even decide whether all pairs can be simultaneously routed on edge-disjoint paths [Kar72, EIS76]. The best currently known approximation algorithm for the problem, due to Chekuri, Khanna and Shepherd [CKS06], achieves an $O(\sqrt{n})$-approximation factor, while the best current hardness of approximation is $\Omega(\log^{1/2-\epsilon} n)$ for any constant $\epsilon$, unless $\text{NP} \subseteq \text{ZPTIME}(n^{\text{poly log} n})$ [AZ05, ACG+10]. We note that the standard multicommodity flow LP relaxation for EDP, that is commonly used in approximation algorithms for network routing problems, has an integrality gap of $\Omega(\sqrt{n})$ [GVY97]. Interestingly, Rao and Zhou [RZ10] have shown an $O(\text{poly log } n)$-approximation algorithm for EDP in graphs where the value of the global minimum cut is $\Omega(\log^8 n)$, by rounding the same LP relaxation.

In the EDP problem, we require that the congestion of the routing is the best possible (congestion 1), and the goal is to maximize the number of pairs routed subject to this constraint. On the other extreme is the Congestion Minimization problem, where we need to route all source-sink pairs, while minimizing the edge congestion. The classical randomized rounding technique of Raghavan and Thompson [RT87] gives the best currently known approximation algorithm for this problem, whose approximation factor is $O(\log n / \log \log n)$. On the negative side, Andrews and Zhang [AZ07] show that the problem is hard to approximate to within a factor of $\Omega\left(\frac{\log \log n}{\log \log \log n}\right)$ unless $\text{NP} \subseteq \text{ZPTIME}(n^{\text{poly log } n})$.

A problem that lies between these two extremes, and is a natural framework for studying the tradeoff between the number of pairs routed and the edge congestion is the Edge Disjoint Paths with Congestion problem (EDPwC). We say that an algorithm $A$ achieves a factor-$\alpha$ approximation with congestion $c$ for EDPwC, if it routes at least $\text{OPT}/\alpha$ source-sink pairs, and the congestion of this routing is bounded by $c$, where $\text{OPT}$ is the maximum number of the demand pairs that can be simultaneously routed on edge-disjoint paths. In particular, a very interesting question is whether, by slightly relaxing the capacity constraints, and allowing small edge congestion, we can significantly increase the number of pairs routed.

When the congestion $c$ is allowed to be as high as $\Omega(\log n / \log \log n)$, the randomized rounding algorithm of Raghavan and Thompson [RT87] gives a constant-factor approximation for EDPwC. For smaller values of $c$, until recently, only $O(n^{1/c})$-approximation algorithms have been known [AR06, BS00, KS04]. In a recent breakthrough, Andrews [And10] has shown an efficient randomized algorithm that routes $\Omega\left(\frac{\text{OPT}}{\log^6 n}\right)$ pairs with congestion $O((\log \log n)^6)$. In another recent result, Kawarabayashi and Kobayashi [KK11] have shown an efficient algorithm that routes $\Omega\left(\frac{\text{OPT}}{n^{1/7}}\right)$ pairs with congestion 2, thus improving the best previously known $O(\sqrt{n})$-approximation for $c = 2$.

In this paper we show an efficient randomized algorithm, that routes $\Omega\left(\frac{\text{OPT}}{\log^{12\cdot c} k \log \log k}\right)$ demand pairs.
with congestion at most 14. We note that on the negative side, Andrews et al. [ACG+10] have shown that for any fixed constant $\epsilon > 0$, for any $1 \leq c \leq O\left(\frac{\log \log n}{\log \log \log n}\right)$, there is no $O\left((\log n)^{\frac{1}{1+c}}\right)$-approximation algorithm for EDPwC with congestion $c$, unless $\text{NP} \subseteq \text{ZTIME}(n^{\text{poly log} n})$. Therefore, the best approximation factor one may hope to achieve for EDPwC in the setting where the allowed congestion is bounded by a constant is polylogarithmic.

In a subsequent work, our main result has been extended in [CL12] to an algorithm that achieves an $O(\text{poly log}(k))$-approximation for EDPwC with congestion 2. Chekuri and Ene [CE13] considered a closely related, and more general Node-Disjoint Paths problem, where the congestion is measured on vertices and not edges. They show an algorithm that achieves an $O(\text{poly log}(k))$-approximation with constant congestion for this problem, building on some of the ideas from this paper. Some of the new technical ideas introduced in this paper have also lead to new results in graph theory, including a new algorithm to decompose large-treewidth graphs into a number of disjoint large-treewidth subgraphs [CC13], a new proof of the Excluded Grid Theorem that achieves the first polynomial bounds on the size of the grid-minor [CC14], and a construction of degree-3 treewidth sparsifiers [CC15].

While our algorithm is guaranteed, with high probability,$^1$ to route at least $\Omega(\text{OPT/ poly log } k)$ of the demand pairs with constant congestion, we have no control over the choice of the pairs that are routed. In some applications it may be useful to be able to choose specific pairs for the algorithm to route beforehand. While we do not expect to be able to pre-select an arbitrary collection of the demand pairs to be routed (from the NP-hardness of the decision version of EDP), under some conditions we can still have some control over their selection. We show that if a given set $T$ of vertices is well-linked in the graph $G$ (see a formal definition below), then we can efficiently find a partition $G$ of $T$ into groups of size $O(\text{poly log } k)$ each, with the following property: if we are given any collection $M = \{(s_1, t_1), \ldots, (s_t, t_t)\}$ of demand pairs, with $M \subseteq T \times T$, where for each group $U \in G$, the vertices of $U$ appear in at most one pair in $M$, then there is an efficient randomized algorithm that with high probability routes all demand pairs in $M$ with constant congestion.

We then turn to vertex flow sparsifiers. Given a graph $G$ with a subset $T$ of $k$ vertices called terminals, and a set $D : T \times T \rightarrow \mathbb{R}^+$ of demands over pairs of vertices of $T$, let $\eta(G, D)$ be the minimum congestion required to fractionally route the demands in $D$ in graph $G$. We say that a graph $H$ is a quality-$q$ vertex flow sparsifier for $(G, T)$ iff $T \subseteq V(H)$, and for any set $D$ of demands over $T$, $\eta(G, D) \leq \eta(H, D) \leq q\eta(G, D)$. Flow sparsifiers were first introduced by Moitra [Moi09] and Leighton and Moitra [LM10]. One of their motivations was to obtain better and faster approximation algorithms for combinatorial optimization problems, whose solution value only depends on the congestion values $\eta(G, D)$ for various sets $D$ of demands. The improvement is obtained by running the approximation algorithm on the sparsifier $H$ instead of $G$, assuming that $|V(H)| \ll |V(G)|$. This results in a better running time, and in better approximation factors for algorithms whose approximation factor depends on the input graph size. Several efficient algorithms are now known for constructing quality-$O(\text{log } k / \text{log log } k)$ sparsifiers $H$ with $V(H) = T$ [CLLM10, EGK+14, MM10], and constant-quality sparsifiers when $V(H)$ may contain additional, non-terminal vertices [Chu12]. However, such sparsifiers do not preserve integral routings. For example, if we were to solve the EDP problem, or some other routing problem in graph $H$, we are not guaranteed that we can transform this solution into an integral solution in graph $G$. This motivates our definition of integral sparsifiers, that approximately preserve both fractional and integral routings. Suppose we are given any $n$-vertex graph $G = (V, E)$ with a subset $T$ of $k$ vertices called terminals. We say that a graph $H$ is a quality-$(q_1, q_2)$ integral flow sparsifier for $G$ and $T$, iff (1) $T \subseteq V(H)$; (2) for any set $D$ of demands over $T$, $\eta(H, D) \leq q_1 \eta(G, D)$ (so in particular if we scale the demands in $D$ down by factor $q_1$, we can route them fractionally in

$^1$Throughout the paper, we use “with high probability” to mean probability $1 - 1/\text{poly}(n)$, where $n$ is the number of the graph vertices.
an integral routing $\mathcal{P}$ of any set $\mathcal{M}$ of pairs of terminals in graph $H$ with congestion $c$, there is an efficient randomized algorithm that with high probability finds an integral routing $\mathcal{P}'$ of $\mathcal{M}$ in $G$ with congestion at most $q_2 \cdot c$. We show an efficient algorithm to construct integral sparsifiers $H$ of quality $(q_1, q_2)$ with $q_1 = O(\text{poly} \log k)$, $q_2 = 31$, and $|V(H)| = O(d)$, where $d$ is the sum of the degrees of all terminals.

Other related work. EDP and its variants have been studied extensively, and better approximation algorithms are known for several special cases. Some examples include planar graphs [Fra85, KT98, Kle05, CKS05, CKS09, KK13, SCS11], trees [GVY97, CMS07], and expander graphs [LR99, BFU94, BFSU94, KR96, Fri00].

We note that the approximability of routing problems is somewhat better understood in directed graphs. The EDP problem has $\tilde{O}$-approximability in directed graphs, where $m$ is the number of the graph edges [CK07, VV04, Kle96], and it is hard to approximate to within a factor of $\Omega(m^{1/2-\epsilon})$ for any constant $\epsilon$ [GKR+03]. The randomized rounding technique of Raghavan and Thompson [RT87] gives an $O(\log n / \log \log n)$-approximation for directed Congestion Minimization, and the problem is hard to approximate to within a factor of $\Omega(\log n / \log \log n)$, unless $\text{NP} \subseteq \text{ZPTIME}(n^{\text{poly} \log n})$ [AZ08, CGKT07]. As for EDPwC, the randomized rounding technique gives an $O(cn^{1/c})$-approximation [KS04, Sri97] for any congestion bound $c$. On the other hand, for any $1 \leq c \leq O\left(\frac{\log n}{\log \log n}\right)$, there is no $n^{O(1/c)}$-approximation algorithm for the problem in directed graphs, unless $\text{NP} \subseteq \text{ZPTIME}(n^{\text{poly} \log n})$ [CGKT07].

Our results and techniques. Our main result is summarized in the following theorem.

**Theorem 1.1** There is a randomized polynomial-time algorithm, that, given an undirected graph $G$ and a set $\mathcal{M} = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ of $k$ pairs of vertices of $G$ called demand pairs, with high probability finds a collection $\mathcal{P}$ of paths, connecting $\Omega\left(\frac{\text{OPT}}{\log^{2.5} k \log \log k}\right)$ demand pairs with congestion at most 14, where $\text{OPT}$ is the maximum number of the demand pairs that can be simultaneously routed on edge-disjoint paths in $G$.

Our algorithm in fact routes $\Omega\left(\frac{\text{OPT}_{LP}}{\log^{2.5} k \log \log k}\right)$ demand pairs, where $\text{OPT}_{LP}$ is the value of the optimal solution to the standard multicommodity flow linear programming relaxation for the problem, where each edge carries at most one flow unit. Since the integrality gap of this LP relaxation is $\Omega(\sqrt{n})$ for EDP when no congestion is allowed, our result shows that the integrality gap improves from polynomial to polylogarithmic if we allow a congestion of 14.

Remark We compare the number of the demand pairs routed by our solution with congestion 14 to the number $\text{OPT}$ of the demand pairs routed by the optimal solution with congestion 1.
factor 14, we obtain a fractional routing with congestion 1. Therefore, the number of the demand pairs routed by our algorithm is at least $\Omega \left( \frac{\text{OPT}_{IP}}{\log^{32} k \log \log k} \right) = \Omega \left( \frac{\text{OPT}}{\log^{32} k \log \log k} \right)$, and we obtain a (real) $O(\text{poly log } k)$ approximation for EDPwC with congestion 14.

A basic notion used throughout the paper is that of well-linkedness. Well-linkedness and its variations have been used extensively in graph minor theory and in previous work on routing problems [R"ac02, CKS05, RZ10, And10]. We say that a set $T$ of vertices is $\alpha$-well-linked in a graph $G$, iff for any partition $(A, B)$ of $V(G)$, $|E(A, B)| \geq \alpha \cdot \min\{|T \cap A|, |T \cap B|\}$.

Suppose we are given a graph $G = (V, E)$, a set $T \subseteq V$ of vertices called terminals, a partition $G$ of $T$, and a collection $M = \{(s_1, t_1), \ldots, (s_r, t_r)\}$ of demand pairs, where for each $1 \leq i \leq r$, $s_i, t_i \in T$. We say that the demand set $M$ is $(1, G)$-restricted, iff for every group $U \in G$, at most one demand pair $(s_i, t_i)$ contains a terminal of $U$ (and only one terminal of $U$ may participate in this pair). Our next theorem allows us to pre-select, to some extent, the demand pairs to be routed, if the set $T$ of terminals is well-linked in $G$.

**Theorem 1.2** Suppose we are given an $n$-vertex graph $G = (V, E)$, a subset $T \subseteq V$ of $k_0$ vertices called terminals, such that $T$ is $\alpha_0$-well-linked in $G$, for some $0 < \alpha_0 < 1$, and an integer $c \geq 1$. Then we can efficiently find a partition $G$ of the terminals of $T$ into groups of size $O \left( \frac{(\log k_0)^{21+11/c}}{\alpha_0} \right)$, such that, given any set $M$ of demand pairs over $T$, where $M$ is $(1, G)$-restricted, there is an efficient randomized algorithm that with high probability finds a routing of all pairs in $M$ with congestion at most $14c + 1$.

In particular, we can achieve congestion 15 with group size $O(\log^{32} k_0/\alpha_0)$, and if the group size is $O(\log^{22} k_0/\alpha_0)$, then the congestion is 155. Finally, the next theorem provides an algorithm for constructing integral sparsifiers.

**Theorem 1.3** There is an efficient algorithm that constructs, for any graph $G$ and set $T \subseteq V(G)$ of $k$ terminals, an integral sparsifier $H$ of quality $(q_1, q_2)$, with $q_1 = O(\text{poly log } k)$, $q_2 = 31$, and $|V(H)| = O(d)$, where $d$ is the sum of the degrees of all terminals.

We now give an overview of our techniques and compare them to previous work. The starting point of the proof of Theorem 1.1 is the same as in the work of [CKS05, RZ10, And10]. We start with the standard LP-relaxation for the EDP problem on graph $G$, and we compute a partition of $G$ into disjoint induced sub-graphs $G_1, \ldots, G_r$. For each $1 \leq i \leq r$, we compute a subset $M_i \subseteq M$ of demand pairs that are contained in $G_i$, such that the corresponding set $T_i$ of terminals, containing all vertices that participate in the pairs in $M_i$, is $\frac{1}{c}$-well-linked in $G_i$, and moreover, $\sum_{i=1}^r |M_i| \geq \Omega \left( \frac{|M|}{\log^2 k} \right)$.

An algorithm for efficiently computing such a decomposition was shown by Chekuri, Khanna and Shepherd [CKS05]. From now on, it is enough to find a good routing in each resulting sub-instance $G_i$ separately. To simplify notation, let $G$ denote any such sub-instance $G_i$, let $M$ denote the set $M_i$ of demand pairs, and let $T$ denote the corresponding set $T_i$ of terminals. Since set $T$ is $\frac{1}{c}$-well-linked in $G$, graph $G$ has good expansion properties with respect to $T$. However, graph $G$ may be far from being an expander, since it may contain many vertices besides the terminals. Intuitively, a natural approach is to embed an expander $X$, whose vertex set is $T$, into the graph $G$. Each edge $e = (t_i, t_j)$ of the expander is mapped to a path $P_e$ connecting $t_i$ to $t_j$ in $G$, and the congestion of the embedding is the maximum, over all edges $e' \in E(G)$, of the number of paths in $\{P_e \mid e \in E(X)\}$, containing $e'$. If we could find a low-congestion embedding of an expander $X$ into $G$, then we could use existing algorithms for routing on expanders to find a low-congestion routing of a polylogarithmic fraction of the demand pairs in $X$, which in turn would give us a low-congestion routing of the same demand pairs in $G$. This general framework was first suggested by Chekuri, Khanna and Shepherd [CKS05], who proposed
to embed a crossbar into the input graph $G$. Intuitively, a crossbar is a graph for which efficient algorithms to approximately compute integral routings are known. In particular, expander graphs can be viewed as crossbars. This framework was used by Rao and Zhou [RZ10] and by Andrews [And10]. A very useful tool in embedding an expander into any graph with a well-linked set $T$ of terminals is the cut-matching game of Khandekar, Rao and Vazirani [KRV09]. In this game, we have two players: a cut player, who wants to construct an expander $X$, and a matching player, who tries to delay its construction. We start with $X$ containing only the set $V(X)$ of $2N$ vertices and no edges. In each iteration $i$, the cut player computes a partition $(A_i, B_i)$ of $V(X)$ with $|A_i| = |B_i| = N$, and the matching player computes a matching $M_i$ between $A_i$ and $B_i$. The edges of $M_i$ are then added to $X$. Khandekar, Rao and Vazirani [KRV09] have shown that no matter what the matching player does, there is a strategy for the cut player (that we denote by $A_{CMG}$), such that after $O(\log^2 N)$ iterations, $X$ becomes an expander. Moreover, there is an efficient algorithm to compute the partitions $(A_i, B_i)$, given the previous responses of the matching player. A natural approach to constructing an expander $X$ and embedding it into the graph $G$ using the cut-matching game, is the following. We use the algorithm $A_{CMG}$ for the cut player, while the matching player is simulated by finding appropriate flows in $G$. Specifically, we let $V(X) = T$. If $(A_i, B_i)$ is the bi-partition of $V(X)$ computed by the cut player, then we can send $|A_i| = |B_i|$ flow units from the terminals of $A_i$ to the terminals of $B_i$ with low congestion in graph $G$, using the fact that $T$ is well-linked in $G$. We then use the resulting flow to define the matching $M_i$. This procedure can be used to both construct the expander $X$, and embed it into $G$. In fact, Khandekar, Rao and Vazirani use precisely this procedure in their algorithm for the sparsest cut problem.

One problem with this approach is that we need to compute $\Theta(\log^2 k)$ different flows in graph $G$, and together they may cause a poly-logarithmic congestion. Moreover, the partitions that the cut player computes depend on the matchings computed in previous iterations, so we cannot attempt to route all these flows simultaneously in graph $G$ with low congestion. Rao and Zhou [RZ10] have proposed the following approach to overcome this difficulty. Let $\gamma = \Theta(\log^2 k)$ be the number iterations in the algorithm of [KRV09]. We can build $\gamma$ graphs $G_1, \ldots, G_{\gamma}$, where for each $1 \leq i \leq \gamma$, $V(G_i) = V(G)$, and the sets $E(G_1), \ldots, E(G_{\gamma})$ of edges form a partition of the edges in $E(G)$. If we can construct the family $G_1, \ldots, G_{\gamma}$ of graphs so that, in each graph $G_i$, the set of terminals is still well-linked, then we can now construct the expander $X$ and embed it into $G$ by using the cut-matching game of [KRV09], where in each iteration $i$, matching $M_i$ is computed by finding a flow from $A_i$ to $B_i$ in graph $G_i$. Since the edges of each set $M_i$ are embedded into distinct graphs $G_i$, the congestion does not accumulate, and we obtain a good embedding of $X$ into $G$. In order to construct the graphs $G_i$, Rao and Zhou use the random procedure of Karger [Kar99], where each edge $e \in E$ is added to one of the graphs $G_i$ uniformly at random. However, this procedure only works if the value of the global minimum cut in $G$ is at least polylogarithmic. In order to overcome this difficulty, Andrews [And10] used Raacek’s tree decomposition technique [Rac02]. Roughly speaking, he decomposes the graph $G$ into a collection $\mathcal{C}$ of disjoint clusters, where each cluster $C \in \mathcal{C}$ has some useful properties that allow us to find good routings across the cluster $C$ efficiently. Moreover, if $H$ is the graph obtained from $G$ by contracting each cluster $C \in \mathcal{C}$ into a single vertex, then the set of terminals is well-linked in $H$, and the global minimum cut in $H$ is large, so we can use the algorithm of Rao and Zhou to complete the routing.

Our algorithm uses a slightly different way to embed an expander into $G$, somewhat similar to the one in [RZ10]. Specifically, each vertex $t \in V(X)$ is represented by a connected component $C_t$ in graph $G$, that contains the terminal $t$. Each edge $e = (t, t') \in E(X)$ is represented by a path $P_e$ connecting some vertex $v \in C_t$ to some vertex $v' \in C_{t'}$ in $G$. We ensure that each edge $e' \in E(G)$ only participates in a constant number of the components $\{C_t\}_{t \in V(X)}$ and paths $\{P_e\}_{e \in E(X)}$. Once we find such an embedding, we use vertex-disjoint routing in the expander $X$, that gives a low edge-congestion routing in the original graph $G$. 

5
A major point of our departure from previous work is in how the expander $X$ is constructed and embedded into $G$. A central notion in our algorithm is that of a family of good routers. Let $k' = \Theta(k/\text{poly log } k)$ be a parameter, where $k = |\mathcal{M}|$ is the number of the demand pairs. For any subset $S \subseteq V$ of vertices, let $\text{out}(S)$ be the set of edges with exactly one endpoint in $S$. Given a subset $S \subseteq V$ of vertices of $G$, we say that $S$ has the $\alpha$-bandwidth property, if the edges of $\text{out}(S)$ are $\alpha$-well-linked in $G[S]$. (More formally, subdivide every edge $e \in \text{out}(S)$ with a new vertex $t_e$, and consider the sub-graph $H_S$ of the resulting graph induced by $S \cup \mathcal{T}'$, where $\mathcal{T}' = \{t_e \mid e \in \text{out}(S)\}$). We say that the set $S$ has the $\alpha$-bandwidth property iff $\mathcal{T}'$ is $\alpha$-well-linked in $H_S$). Similarly, if we are given a subset $\Gamma \subseteq \text{out}(S)$ of edges, we say that $S$ has the $\alpha$-bandwidth property with respect to $\Gamma$ iff the set $\{t_e \mid e \in \Gamma\}$ is $\alpha$-well-linked in $H_S$.

We say that a subset $S \subseteq G$ of vertices is a good router, iff there is a collection $\Gamma \subseteq \text{out}(S)$ of $k'$ edges, such that $S$ has the $\alpha$-bandwidth property with respect to $\Gamma$ (where $\alpha = \Omega(1/\text{poly log } k)$), and moreover the edges in $\Gamma$ can send $|\Gamma|$ flow units in $G$ to the terminals in $\mathcal{T}$ with constant edge-congestion. A family $\mathcal{F}$ of vertex subsets is a family of good routers iff it contains $\gamma$ mutually disjoint good routers $S_1, \ldots, S_\gamma$, where $\gamma = O(\log^2 k)$ is the parameter from the cut-matching game of [KRV09].

Suppose we find a family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of good routers. For each $1 \leq j \leq \gamma$, let $\Gamma_j \subseteq \text{out}(S_j)$ be the corresponding subset $\Gamma$ of edges. In order to construct the expander $X$, we select a subset $\mathcal{T}' = \{t_1, \ldots, t_{k'}\} \subseteq \mathcal{T}$ of $k'$ terminals, and we let $V(X) = \mathcal{T}'$. For each $1 \leq i \leq k'$, we then construct a connected sub-graph $C_i$ in graph $G$, that contains, for each $1 \leq j \leq \gamma$, a distinct edge $e_{i,j} \in \Gamma_j$, and also contains the terminal $t_i$ (see Figure 1). For each $1 \leq j \leq \gamma$, the edges $e_{1,j}, \ldots, e_{k',j}$ are all distinct, and we view the edge $e_{i,j}$ as the representative of terminal $t_i$ for the set $S_j$. We also ensure that each edge of graph $G$ only participates in a constant number of the sub-graphs $\{C_i\}_{i=1}^{k'}$. For each $i$, $C_i$ is viewed as representing the vertex $t_i$ of $X$ in graph $G$. In order to construct the expander $X$, we use the cut-matching game of [KRV09], where in each iteration $1 \leq j \leq \gamma$, we use the sub-graph $G[S_j]$ to route some matching $M_j$ between the representatives of the terminals in $A_j$ and $B_j$ for the set $S_j$. This ensures that the congestion does not accumulate across different iterations. Finally, we show an efficient algorithm for finding a family $\mathcal{F}$ of good routers.

**Organization.** Most of this paper is devoted to the proof of our main result, Theorem 1.1. We start with preliminaries in Section 2, and provide the proof in Section 3. For convenience, a list of parameters is provided in Section D of the Appendix. The proofs of Theorems 1.2 and 1.3 appear in Sections 4 and 5 respectively.
2 Preliminaries and Notation

In this section we provide notation and basic results that we use to prove Theorem 1.1. We assume that we are given an undirected $n$-vertex graph $G = (V,E)$, and a set $\mathcal{M} = \{(s_1,t_1), \ldots, (s_k,t_k)\}$ of $k$ source-sink pairs, that we also refer to as demand pairs. We denote by $\mathcal{T}$ the set of vertices that participate in pairs in $\mathcal{M}$, and we call them terminals. Let $\text{OPT}$ denote the maximum number of the demand pairs that can be simultaneously routed via edge-disjoint paths. Our goal is to connect $\Omega(\text{OPT}/\text{poly log } k)$ distinct pairs in $\mathcal{M}$ with paths that cause congestion at most 14. Throughout the paper, all logarithms are to the base of 2. We sometimes refer to vertex subsets as clusters. We say that a cluster $C \subseteq V(G)$ is connected iff $G[C]$ is connected.

We perform a number of transformations to our graph, to ensure that the maximum vertex degree in $G$ is 4, the degree of every terminal is 1, and every terminal participates in one source-sink pair. We note that these are standard transformations that were used in prior work. We assume without loss of generality that every terminal of $\mathcal{T}$ participates in exactly one source-sink pair. Otherwise, if a terminal $v \in \mathcal{T}$ participates in $r > 1$ source-sink pairs, we can add $r$ new terminals $t_1(v), \ldots, t_r(v)$, connect each of them to $v$ with an edge, and use a distinct terminal in $\{t_1(v), \ldots, t_r(v)\}$ for each source-sink pair in which $v$ participates. We also assume without loss of generality that the maximum vertex degree in $G$ is 4, and that the degree of every terminal is 1. In order to achieve this, we perform the following simple transformation of graph $G$. If $v$ is a terminal, whose degree is greater than 1, then we add a new vertex $u$ to graph $G$ that connects to $v$ with an edge, and becomes a terminal instead of $v$. Next, we process the non-terminal vertices one-by-one. Let $v$ be any such vertex, and assume that the degree of $v$ is $d > 4$. Let $u_1, \ldots, u_d \in V$ be the neighbors of $v$. We replace $v$ with a $d \times d$ grid $Z_v$, and denote by $u'_1, \ldots, u'_d$ the vertices in the first row of $Z_v$. For each $1 \leq i \leq d$, we add an edge $(u_i, u'_i)$. It is easy to verify that any solution to the EDP problem in the original graph can be transformed into a feasible routing of the same value and no congestion in the new graph, and any routing in the new graph with congestion $\eta$ can be transformed into a routing in the original graph with the same congestion. Therefore, we assume from now on that the maximum vertex degree in $G$ is 4, the degree of every terminal is 1, and every terminal participates in one source-sink pair.

For any subset $S \subseteq V$ of vertices, we denote by $\text{out}_G(S) = E_G(S, V \setminus S)$, and by $E_G(S)$ the subset of edges with both endpoints in $S$. When clear from context, we omit the subscript $G$. Throughout the paper, we say that a random event succeeds with high probability, if the probability of success is at least $(1 - 1/\text{poly}(n))$.

Let $\mathcal{P}$ be any collection of paths in graph $G$. We say that paths in $\mathcal{P}$ cause congestion $\eta$ in $G$, iff for every edge $e \in E$ at most $\eta$ paths in $\mathcal{P}$ contain $e$, and some edge belongs to $\eta$ paths of $\mathcal{P}$. Assume that we are given a subset $S \subseteq V$ of vertices and a subset $E' \subseteq E$ of edges of $G$. We say that a collection $\mathcal{P}$ of paths connects the vertices of $S$ to the edges of $E'$ with congestion at most $\eta$, and denote $\mathcal{P} : S \tilde{\rightarrow} \eta E'$, iff $\mathcal{P} = \{ P_v | v \in S \}$, where path $P_v$ has $v$ as its first vertex and some edge of $E'$ as its last edge, and $\mathcal{P}$ causes congestion at most $\eta$ in $G$. In particular, each edge in $E'$ serves as the last edge on at most $\eta$ paths in $\mathcal{P}$. Similarly, given two subsets $S, S'$ of vertices, if $\mathcal{P}$ is a collection of paths, connecting every vertex of $S$ to some vertex of $S'$ with overall edge-congestion at most $\eta$, then we denote this by $\mathcal{P} : S \rightarrow \eta S'$. Finally, if $|S| = |S'| = |\mathcal{P}|$, and each path in $\mathcal{P}$ connects a distinct vertex of $S$ to a distinct vertex of $S'$ with edge-congestion at most $\eta$, then we denote this by $\mathcal{P} : S \rightarrow \eta S'$. Similarly, we say that a flow $F$ connects the vertices of $S$ to the edges of $E'$ with congestion at most $\eta$, and denote $F : S \rightarrow \eta E'$, iff each vertex $v \in S$ sends one flow unit to the edges in $E'$, and the flow $F$ causes congestion at most $\eta$ in $G$. Notice that each flow-path in $F$ starts at a vertex of $S$ and terminates at some edge $e \in E'$. We view edge $e$ as part of the flow-path, so in particular each edge in $E'$ receives at most $\eta$ flow units. Notice that from the integrality of flow, for any integer $\eta \geq 1$, there is a flow $F : S \rightarrow \eta E'$ iff there is a collection $\mathcal{P} : S \rightarrow \eta E'$ of paths. We define
flows and paths between subsets of edges similarly. Given two subsets \(S, S'\) of vertices, if \(F : S \sim \eta, S'\) is a flow where every vertex of \(S\) sends one flow unit, and every vertex of \(S'\) receives one flow unit, then we denote this by \(F : S \xrightarrow{1/1} S'\).

We will often be interested in a scenario where we are given a subset \(S\) of vertices and two subsets \(E_1, E_2 \subseteq \text{out}(S)\) of edges. We say that the flow \(F : E_1 \sim \eta, E_2\) is contained in \(S\) if every flow-path is completely contained in \(G[S]\), except for its first and last edges, that belong to \(\text{out}(S)\). Similarly, we say that a set \(\mathcal{P} : E_1 \sim \eta, E_2\) of paths is contained in \(S\) if all inner edges on every path of \(\mathcal{P}\) belong to \(G[S]\).

Given a graph \(G = (V, E)\), and a set \(T \subseteq V\) of terminals, a set \(D\) of demands is a function \(D : T \times T \to \mathbb{R}^+\), that specifies, for every unordered pair \(t, t' \in T\), a demand \(D_{t, t'}\). We say that the set \(D\) of demands is \(\gamma\)-restricted, iff for each \(t \in T\), the total demand \(\sum_{t' \in T} D_{t, t'} \leq \gamma\). Given a partition \(\mathcal{G}\) of the set \(T\) of terminals, we say that the set \(D\) of demands is \((\gamma, \mathcal{G})\)-restricted, iff for each \(U \in \mathcal{G}\), the total demand \(\sum_{t \in U} \sum_{t' \in T} D_{t, t'} \leq \gamma\). We say that the set \(D\) of demands is integral iff \(D_{t, t'}\) is integral for each \(t, t' \in T\).

Given any set \(D\) of demands, a fractional routing of \(D\) is a flow \(F\), where every unordered pair \(t, t' \in T\) sends \(D_{t, t'}\) flow units to each other. Given an integral set \(D\) of demands, an integral routing of \(D\) is a collection \(\mathcal{P}\) of paths, where for each unordered pair \((t, t') \in T\), there are \(D_{t, t'}\) paths connecting \(t\) to \(t'\) in \(\mathcal{P}\). The congestion of this routing is the congestion caused by the set \(\mathcal{P}\) of paths in \(G\). Observe that any matching \(\mathcal{M}\) on the set \(T\) of terminals defines a set \(D\) of demands where \(D_{t, t'} = 1\) for \((t, t') \in \mathcal{M}\) and \(D_{t, t'} = 0\) otherwise. We do not distinguish between the matching \(\mathcal{M}\) and the set \(D\) of demands.

**Sparsest Cut and the Flow-Cut Gap.** Suppose we are given a graph \(G = (V, E)\), with non-negative weights \(w_v\) on vertices \(v \in V\), and a subset \(T \subseteq V\) of \(k\) terminals, such that for all \(v \notin T\), \(w_v = 0\). For any subset \(S \subseteq V\) of vertices, let \(w(S) = \sum_{v \in S} w(v)\). The sparsity of a cut \((S, \overline{S})\) in \(G\) is \(\Phi(S) = \frac{|E(S, \overline{S})|}{w(S) \cdot w(\overline{S})}\), and the value of the sparsest cut in \(G\) is defined to be: \(\Phi(G) = \min_{S \subseteq V} \{\Phi(S)\}\). In the sparsest cut problem, the input is a graph \(G\) with non-negative vertex weights, and the goal is to find a cut of minimum sparsity. Arora, Rao and Vazirani [ARV09] have shown an \(O(\sqrt{k \log k})\)-approximation algorithm for the sparsest cut problem. We will often work with a special case of the sparsest cut problem, where for each \(t \in T\), \(w_t = 1\).

A problem dual to sparsest cut is the maximum concurrent flow problem. For the case where the weights of all terminals are unit, the goal in the maximum concurrent flow problem is to find the maximum value \(\lambda\), such that every pair of terminals can send \(\lambda\) flow units to each other simultaneously with no congestion. The flow-cut gap is the maximum ratio, in any graph, between the value of the minimum sparsest cut and the maximum concurrent flow. The value of the flow-cut gap in undirected graphs with \(k\) terminals, that we denote by \(\beta(k)\) throughout the paper, is \(\Theta(\log k)\) [LR99]. Therefore, if \(\Phi(G) = \alpha\), then every pair of terminals can send \(\alpha / \beta(k)\) flow units to each other with no congestion.

We will use a slightly different, but also standard, and roughly equivalent, definition of sparsity. Given any partition \((S, \overline{S})\) of \(V\), the sparsity of the cut \((S, \overline{S})\) is \(\Psi(S, \overline{S}) = \frac{|E(S, \overline{S})|}{\min\{w(S), w(\overline{S})\}}\). We then denote: \(\Psi(G) = \min_{S \subseteq V} \{\Psi(S, \overline{S})\}\). It is easy to see that, if the weight of every terminal is 1, then \(2\Psi(G) / k \geq \Phi(G) \geq \Psi(G) / k\). Therefore, if \(\Psi(G) = \alpha\), then \(\Phi(G) \geq \alpha / k\), and every pair of terminals can send \(\frac{\alpha}{k \cdot \beta(k)}\) flow units to each other with no congestion. Equivalently, every pair of terminals can send \(1 / k\) flow units to each other with congestion at most \(\beta(k)/\alpha\). Moreover, any matching on the set \(T\) of terminals can be fractionally routed with congestion at most \(2 \beta(k)/\alpha\). In the rest of the paper, we will use the latter definition of sparsity, and we will use the term cut sparsity and the value of sparsest cut to denote \(\Psi(S, \overline{S})\) and \(\Phi(G)\) respectively. The algorithm of [ARV09] can still be used to
obtain a cut of sparsity at most $O(\sqrt{\log k}) \cdot \Psi(G)$ in $G$. We denote by $A_{\text{ARV}}$ this algorithm and by $\alpha_{\text{ARV}}(k) = O(\sqrt{\log k})$ its approximation factor.

**Routing on Expanders.** We say that a multi-graph $G = (V, E)$ is an $\alpha$-expander, iff

$$\min_{0 < |S| \leq |V|/2} \left\{ \frac{|E(S, S')|}{|S|} \right\} \geq \alpha.$$  

There are many algorithms for routing on expanders, e.g. [LR99, BFU94, BFSU94, KR96, Fri00], which give different types of guarantees. For example, Frieze [Fri00] has shown that if $G$ is an $r$-regular graph (where $r$ is a constant) with strong enough expansion properties, then there is an efficient randomized algorithm for routing any subset of $\Omega(n/\log n)$ of its vertices via edge-disjoint paths. We need a slightly different type of guarantee: the routing should be on **vertex-disjoint** paths, and the graph degree may be super-constant. Rao and Zhou [RZ10] give such an algorithm, which is summarized in the next theorem. For completeness, we provide a proof sketch in the Appendix.

**Theorem 2.1 (Theorem 7.1 in [RZ10])** Let $G = (V, E)$ be any $n$-vertex $d$-regular $\alpha$-expander. Assume further that $n$ is even, and that the vertices of $G$ are partitioned into $n/2$ disjoint demand pairs $M = \{(s_1, t_1), \ldots, (s_{n/2}, t_{n/2})\}$. Then there is an efficient algorithm that routes $\Omega\left(\frac{\alpha^2 n}{\log n d^2}\right)$ of the demand pairs on vertex-disjoint paths in $G$.

**The Cut-Matching Game.** We use the cut-matching game of Khandekar, Rao and Vazirani [KRV09]. In this game, we are given a set $V$ of $N$ vertices, where $N$ is even, and two players: a cut player, whose goal is to construct an expander $X$ on the set $V$ of vertices, and a matching player, whose goal is to delay its construction. The game is played in iterations. We start with the graph $X$ containing the set $V$ of vertices, and no edges. In each iteration $j$, the cut player computes a bi-partition $(A_j, B_j)$ of $V$ into two equal-sized sets, and the matching player returns some perfect matching $M_j$ between the two sets. The edges of $M_j$ are then added to $X$. Khandekar, Rao and Vazirani have shown that there is a strategy for the cut player, guaranteeing that after $O(\log^2 N)$ iterations we obtain a $\frac{1}{2}$-expander with high probability. Subsequently, Orecchia et al. [OSVV08] have shown the following improved bound:

**Theorem 2.2 ([OSVV08])** There is an efficient probabilistic algorithm for the cut player, such that, no matter how the matching player plays, after $\gamma_{\text{CMG}}(N) = O(\log^2 N)$ iterations, graph $X$ is an $\Omega(\log N)$-expander, with constant probability.

**Well-Linkedness, Bandwidth Property, and Bandwidth Decompositions.** We define the notion of well-linkedness, that has been used extensively in algorithms for network routing, e.g. in [Räc02, CKS13, CKS05, RZ10, And10].

**Definition 2.1** We say that a set $T$ of vertices is $\alpha$-well-linked in graph $G$, iff for any partition $(A, B)$ of $V(G)$, $|E_G(A, B)| \geq \alpha \cdot \min \{|A \cap T|, |B \cap T|\}$.

Following is an easy observation about well-linkedness, that follows from the duality between cuts and flows.

**Observation 2.3** Set $T$ of vertices is $\alpha$-well-linked in graph $G$, for any real number $0 < \alpha \leq 1$, iff for any pair $T_1, T_2$ of disjoint equal-sized subsets of vertices of $T$, there is a flow $F : T_1 \xrightarrow{1/\alpha} T_2$ in $G$.

Next, we define **bandwidth property** - a notion closely related to well-linkedness, and similar to the notion of bandwidth property defined in [Räc02]. Intuitively, we say that a cluster $S$ has the $\alpha$-
bandwidth property, iff the boundary \(\text{out}(S)\) of the cluster is \(\alpha\)-well-linked in \(G[S]\). The formal definition appears below.

**Definition 2.2** Given a graph \(G\), a subset \(S\) of its vertices, and a real number \(0 < \alpha \leq 1\), we say that \(S\) has the \(\alpha\)-bandwidth property, iff for any partition \((A,B)\) of \(S\), if we denote by \(T_A = \text{out}(A) \cap \text{out}(S)\), and by \(T_B = \text{out}(B) \cap \text{out}(S)\), then \(|E(A,B)| \geq \alpha \cdot \min\{|T_A|,|T_B|\}\).

Given any subset \(\Gamma \subseteq \text{out}(S)\) of edges, we say that \(S\) has the \(\alpha\)-bandwidth property with respect to \(\Gamma\) iff for any partition \((A,B)\) of \(S\), \(|E(A,B)| \geq \alpha \cdot \min\{|\text{out}(A) \cap \Gamma|,|\text{out}(B) \cap \Gamma|\}\).

We also need a more general notion of bandwidth property that we define below. Intuitively, it handles subsets \(S\) of vertices, where \(|\text{out}(S)|\) may be large, but we only need to route small amounts of flow across \(S\).

**Definition 2.3** Let \(S\) be any subset of vertices of a graph \(G\). For any integer \(k > 0\) and for any real number \(0 < \alpha < 1\), we say that set \(S\) has the \((k,\alpha)\)-bandwidth property iff for any pair \(T_1, T_2 \subseteq \text{out}(S)\) of disjoint subsets of edges, with \(|T_1| + |T_2| \leq k\), for any partition \((X,Y)\) of \(S\) with \(T_1 \subseteq \text{out}(X)\) and \(T_2 \subseteq \text{out}(Y)\), \(|E_G(X,Y)| \geq \alpha \cdot \min\{|T_1|,|T_2|\}\).

Note that if \(|\text{out}(S)| \leq k\), then set \(S\) has the \((k,\alpha)\)-bandwidth property iff it has the \(\alpha\)-bandwidth property, and the two definitions of bandwidth property become equivalent.

**Observation 2.4** If \(S \subseteq V(G)\) has the \((k,\alpha)\)-bandwidth property, then for any subset \(T \subseteq \text{out}(S)\) of at most \(k\) edges, any matching on \(T\) can be fractionally routed inside \(S\) with congestion at most \(2\beta(k)/\alpha\).

**Proof:** We set up an instance of the sparsest cut problem on a graph \(H\), obtained as follows: first, we subdivide every edge \(e \in T\) by a vertex \(v_e\), and we let \(\Gamma = \{v_e \mid e \in T\}\) be the resulting set of new vertices. Let \(H\) be the sub-graph of the resulting graph, induced by \(S \cup \Gamma\). Consider an instance of the sparsest cut problem on graph \(H\), with the set \(\Gamma\) of terminals. Since \(S\) has the \((k,\alpha)\)-bandwidth property, the value of the sparsest cut is at least \(\alpha\), and so any matching on \(T\) can be routed with congestion at most \(2\beta(k)/\alpha\).

Assume now that \(S\) does not have the \((k,\alpha)\)-bandwidth property. Then there must be a partition \((X,Y)\) of \(S\), such that \(|E(X,Y)| < \alpha \cdot \min\{|\text{out}(X) \cap \text{out}(S)|,|\text{out}(Y) \cap \text{out}(S)|,k/2\}\). We say that \((X,Y)\) is a \((k,\alpha)\)-violating cut for \(S\).

The reason for defining the more general \((k,\alpha)\)-bandwidth property is the following. Given a set \(S\) of vertices of \(G\), we will sometimes need to find a partition of \(S\) into a set \(W\) of clusters, such that, on the one hand, each cluster \(W \in W\) has the \(\alpha\)-bandwidth property, while on the other hand, \(\sum_{W \in W} |\text{out}(W)|\) is very close to \(|\text{out}(S)|\). Procedures for finding such decompositions are well-known (sometimes under the name of well-linked decomposition). The problem is that \(|\text{out}(S)|\) may be much larger than \(k\) (for example, it can be close to \(n\)), and in order to obtain such a partition, we need to set \(\alpha = 1/\text{poly log} \ n\). In our algorithm, since we are trying to obtain an \(O(\text{poly log} k)\)-approximation for \(\text{EDPwC}\), including in scenarios where \(k \ll n\), this dependency of \(\alpha\) on \(n\) is unacceptable, and we need \(\alpha = 1/\text{poly log} \ k\). If we use the new \((k,\alpha)\)-bandwidth property, instead of the standard \(\alpha\)-bandwidth property, such a decomposition \(W\) exists, as we show below, with \(\alpha = 1/\text{poly log} \ k\). The \((k,\alpha)\)-bandwidth property is sufficient for us, since we only use the bandwidth property of \(S\) in order to route small (up to \(k\)) amounts of flow through \(S\).

We now discuss the bandwidth-decomposition procedure in more detail. Given a subset \(S\) of vertices of \(G\), we would like to find a partition \(W\) of \(S\), such that every set in \(W \in W\) has the \((k,\alpha)\)-bandwidth
property. We could do so using the standard well-linked decomposition procedures, for example similar to those used in [Räc02, CKS05]. However, in order to do so, we need to be able to check whether a given subset \( W \) of vertices has the \((k, \alpha)\)-bandwidth property, and if not, find a \((k, \alpha)\)-violating cut efficiently. We do not know how to do this, even approximately. Therefore, we will assume for now that we are given an oracle that finds a \((k, \alpha)\)-violating cut in a given subset of vertices, if such a cut exists.

We describe the decomposition procedure and bound the number of edges \( \sum_{W \in \mathcal{W}} |\text{out}(W)| \) in the resulting decomposition. When we use this decomposition later in the algorithm, we will be interested in routing small amounts of flow (up to \( k \)) across the clusters of the decomposition. Whenever we will be unable to route this flow, we will naturally obtain a \((k, \alpha)\)-violating cut. Therefore, our algorithm itself will serve as an oracle to the decomposition procedure. We note that in the final decomposition \( \mathcal{W} \), not all sets \( W \in \mathcal{W} \) may have the \((k, \alpha)\)-bandwidth property, but we will be able to route the flow that we need to route across these clusters, and this is sufficient for us. We now describe the oracle-based decomposition procedure and analyze it.

We are given as input a subset \( S \) of vertices, an integer \( k > 8 \), and a real parameter \( 0 < \alpha < 1 \). Throughout the decomposition procedure, we maintain a partition \( \mathcal{W} \) of \( S \), and at the beginning, \( \mathcal{W} = \{ S \} \). The algorithm proceeds as follows. As long as not all sets in \( \mathcal{W} \) have the \((k, \alpha)\)-bandwidth property, our oracle computes a \((k, \alpha)\)-violating cut \((X, Y)\) of one of the sets \( W \in \mathcal{W} \). We then remove \( W \) from \( \mathcal{W} \) and add \( X \) and \( Y \) to \( \mathcal{W} \) instead. In the next theorem, we bound \( \sum_{W \in \mathcal{W}} |\text{out}(W)| \).

\textbf{Theorem 2.5} Assume that \( k > 8 \), \( \alpha < \frac{1}{48 \log k} \), and let \( \mathcal{W} \) be any partition of \( S \) produced over the course of the above algorithm. Then \( \sum_{W \in \mathcal{W}} |\text{out}(W)| \leq |\text{out}(S)|(1 + 32 \alpha \log k) \).

We emphasize that the bound on \( \sum_{W \in \mathcal{W}} |\text{out}(W)| \) holds for any partition produced over the course of the algorithm, and not just the final partition.

\textbf{Proof:} The proof uses standard arguments. Let \( E^* = E(S) \cup \text{out}(S) \). Throughout the algorithm, we assign non-negative budgets \( B(v, e) \) to all edges \( e \in E^* \), and vertices \( v \in S \), where \( v \) is an endpoint of \( e \). Given a current partition \( \mathcal{W} \) of \( S \), the budgets are assigned as follows. For each set \( W \in \mathcal{W} \), for each edge \( e \in \text{out}(W) \), and endpoint \( v \) of \( e \) lying in \( W \), we set: \( B(v, e) = 8 \alpha \log(|\text{out}(W)|) \) if \( |\text{out}(W)| < k \), and we set \( B(v, e) = 16 \alpha \log k + 8 \alpha \left( 1 - \frac{k}{|\text{out}(W)|} \right) \) otherwise. All other budgets \( B(v, e) \) are set to 0. It is easy to see that all budgets are non-negative. It is now enough to prove that throughout the algorithm, the following invariant holds:

\[
\sum_{e \in E^*} \sum_{v \in e \cap S} B(v, e) + \sum_{W \in \mathcal{W}} |\text{out}(W)| \leq |\text{out}(S)|(1 + 32 \alpha \log k). \tag{1}
\]

Clearly, the budget value \( B(v, e) \) for \( e \in E^* \) and \( v \in S \cap e \) never exceeds \( 24 \alpha \log k \). At the beginning of the algorithm, the sum of all budgets \( B(v, e) \) is therefore bounded by \( 24 \alpha \log k |\text{out}(S)| \), and so the invariant holds. We now consider some iteration, where cluster \( W \) was partitioned into clusters \( X \) and \( Y \), and we assume without loss of generality that \( |\text{out}(X)| \leq |\text{out}(Y)| \). We assume that (1) holds at the beginning of the current iteration, and we prove that it continues to hold at the end of the iteration. Recall that from the definition of the \((k, \alpha)\)-violating cut, \( |E(X, Y)| < \alpha \cdot \text{min} \{|\text{out}(W) \cap \text{out}(X)|, k/2\} \).

Notice also that \( |\text{out}(X)| \leq |\text{out}(X) \cap \text{out}(W)| + |E(X, Y)| < 1.1|\text{out}(X) \cap \text{out}(W)| \leq 0.55|\text{out}(W)| \).

We now consider three cases.

First, if \( |\text{out}(W)| < k \), then the total decrease in the budgets of pairs \((v, e)\) where \( e \in \text{out}(X) \cap \text{out}(W) \) and \( v \in X \cap e \) is at least:
\(|\text{out}(X) \cap \text{out}(W)| \cdot (8\alpha \log(|\text{out}(W)|) − 8\alpha \log(|\text{out}(X)|)) = |\text{out}(X) \cap \text{out}(W)| \cdot 8\alpha \log(|\text{out}(W)|/|\text{out}(X)|) ≥ 6\alpha|\text{out}(X) \cap \text{out}(W)|,\)

since \(|\text{out}(X)| ≤ 0.55|\text{out}(W)|\). The total number of edges in \(E(X, Y)\) is bounded by \(\alpha|\text{out}(X) \cap \text{out}(W)|\), and each such edge contributes 2 to the summation \(\sum_{W \in \mathcal{W}} |\text{out}(W)|\). Additionally, for each edge \(e \in E(X, Y)\), for each endpoint \(v\) of \(e\), the budget \(B(v, e)\) becomes at most \(8\alpha \log k\). Therefore, the total increase in \(\sum_{W \in \mathcal{W}} |\text{out}(W)|\) and \(\sum_{e \in E(X, Y)} \sum_{v \in e} B(v, e)\) is bounded by:

\(|E(X, Y)|(2 + 16\alpha \log k) \leq 2.5|E(X, Y)| \leq 2.5\alpha|\text{out}(X) \cap \text{out}(W)|.\)

The budgets \(B(v, e)\) where \(e \in \text{out}(W) \cap \text{out}(Y)\) and \(v \in e \cap Y\) can only decrease, and all other budgets remain unchanged. We conclude that the invariant (1) continues to hold at the end of the iteration.

The second case is when \(|\text{out}(W)| ≥ k\), but \(|\text{out}(X)| < k\). In this case, for each edge \(e \in \text{out}(X) \cap \text{out}(W)\), and endpoint \(v \in X\) of \(e\), budget \(B(v, e)\) decreases by at least \(8\alpha \log k\), and so the total decrease in budgets \(\sum_{e \in \text{out}(X) \cap \text{out}(W)} \sum_{v \in X \cap e} B(v, e)\) is at least \(8|\text{out}(X) \cap \text{out}(W)| \cdot \alpha \log k\). Similarly to the previous case, the total increase in \(\sum_{W \in \mathcal{W}} |\text{out}(W)|\) and \(\sum_{e \in E(X, Y)} \sum_{v \in e} B(v, e)\) is bounded by:

\(|E(X, Y)|(2 + 32\alpha \log k) \leq 3|E(X, Y)| \leq 3\alpha|\text{out}(X) \cap \text{out}(W)|.\)

Therefore, the invariant (1) continues to hold at the end of the iteration.

The third, and the final case, is when \(|\text{out}(W)| ≥ k\) and \(|\text{out}(X)| ≥ k\). In this case, the decrease in \(\sum_{e \in \text{out}(X) \cap \text{out}(W)} \sum_{v \in X \cap e} B(v, e)\) is at least:

\(|\text{out}(W) \cap \text{out}(X)| \cdot \left(8\alpha \left(1 - \frac{k}{|\text{out}(W)|}\right) − 8\alpha \left(1 - \frac{k}{|\text{out}(X)|}\right)\right) = |\text{out}(W) \cap \text{out}(X)| \cdot 8k\alpha \left(\frac{1}{|\text{out}(X)|} − \frac{1}{|\text{out}(W)|}\right) ≥ |\text{out}(W) \cap \text{out}(X)| \cdot \frac{3.6k\alpha}{|\text{out}(X)|} ≥ 3.24k\alpha,\)

since \(|\text{out}(X)| ≤ 0.55|\text{out}(W)|\), and \(|\text{out}(W) \cap \text{out}(X)| ≥ 0.9|\text{out}(X)|\) (as \(|E(X, Y)| ≤ 0.1|\text{out}(W) \cap \text{out}(X)|\)). On the other hand, the total increase in \(\sum_{W \in \mathcal{W}} |\text{out}(W)|\) and \(\sum_{e \in E(X, Y)} \sum_{v \in e} B(v, e)\) is bounded by:

\(|E(X, Y)|(2 + 48\alpha \log k) \leq 3|E(X, Y)| \leq 3\alpha k.\)

The budgets \(B(v, e)\) where \(e \in \text{out}(W) \cap \text{out}(Y)\) and \(v \in e \cap Y\) can only decrease, and all other budgets remain unchanged. Therefore, the invariant (1) continues to hold at the end of the iteration. □

Throughout the paper, we use \(α(k) = \frac{1}{2^{\Theta(\sqrt{\log k})}}\) and \(α_{BW}(k) = α(k)/α_{ARV}(k)\).

If \(|\text{out}(S)| ≤ k\), then we can obtain a decomposition \(\mathcal{W}\) of \(S\), where each set \(W \in \mathcal{W}\) has the \((k, α_{BW}(k))\)-bandwidth property efficiently, by using the algorithm \(A_{ARV}\) for Sparsest Cut as our oracle:
In each iteration, for each $W \in \mathcal{W}$, we apply the algorithm $\mathcal{A}_{ARV}$ to the corresponding instance of the sparsest cut problem (where the edges of $\text{out}(W)$ are viewed as terminals). If algorithm $\mathcal{A}_{ARV}$ returns a $(k, \alpha(k))$-violating cut $(X, Y)$ for any set $W \in \mathcal{W}$, then we can proceed with the decomposition procedure as before. Otherwise, we are guaranteed that each set $W \in \mathcal{W}$ has the $\alpha_{bw}(k)$-bandwidth property. We therefore have the following corollary.

**Corollary 2.6** Let $S$ be any subset of vertices of $G$, such that $|\text{out}(S)| \leq k$, where $k \geq 8$. Then we can efficiently find a partition $W$ of $S$, such that for each $W \in \mathcal{W}$, $|\text{out}(W)| \leq k$, and $W$ has the $\alpha_{bw}(k)$-bandwidth property. Moreover, $\sum_{W \in \mathcal{W}} |\text{out}(W)| \leq |\text{out}(S)| \left(1 + \frac{1}{44\gamma_{CMG}(k)}\right)$.

The Grouping Technique. The grouping technique was first introduced by Chekuri, Khanna and Shepherd [CKS13], and has since been widely used in algorithms for network routing [CKS05, RZ10, And10], to boost network connectivity and well-linkedness parameters. We summarize it in the following theorem, whose proof appears in the Appendix for completeness.

**Theorem 2.7** Suppose we are given a connected graph $G = (V, E)$, with weights $w(v)$ on vertices $v \in V$, and a parameter $p$. Assume further that for each $v \in V$, $0 \leq w(v) \leq p$, and $\sum_{v \in V} w(v) \geq p$. Then we can find a partition $\mathcal{G}$ of the vertices in $V$, and for each group $U \in \mathcal{G}$, find a tree $T_U \subseteq G$ containing all vertices of $U$, such that the trees $\{T_U\}_{U \in \mathcal{G}}$ are edge-disjoint, and for each $U \in \mathcal{G}$, $p \leq \sum_{v \in U} w(v) \leq 3p$.

We will sometimes use the grouping theorem in slightly different settings. The first such setting is when we are given a subset $T \subseteq V$ of vertices called terminals, and we would like to group them into groups of cardinality at least $p$ and at most $3p$. In this case we will think of all non-terminal vertices as having weight 0, and terminal vertices as having weight 1. Instead of finding a partition $\mathcal{G}$ of all vertices, we will be looking for a partition $\mathcal{G}'$ of the set $T$ of terminals. This partition is obtained from $\mathcal{G}$ by ignoring the non-terminal vertices. Another setting is when we are given a subset $E' \subseteq E$ of edges, and we would like to find a partition $\mathcal{G}$ of these edges into groups of at least $p$ and at most $3p$ edges. As before, we would also like to find, for each group $U \in \mathcal{G}$, a tree $T_U$ containing all edges of $U$, and we require that the trees $\{T_U\}_{U \in \mathcal{G}}$ are edge-disjoint. This setting can be reduced to the previous one, by sub-dividing each edge $e \in E'$ with a terminal vertex. It is easy to verify that Theorem 2.7 can be applied in this setting as well.

### 3 The Algorithm

This section is dedicated to proving Theorem 1.1. Our starting point is similar to that used in previous work on the problem [CKS13, CKS05, RZ10, And10]: namely, we use the standard multicommodity flow LP-relaxation for the EDP problem to partition our graph into several disjoint sub-graphs, and compute a subset of the demand pairs for each such sub-graph, such that the terminals participating in the demand pairs are well-linked in the sub-graph. We then solve the problem separately on each such sub-graph. We use the standard multicommodity-flow LP-relaxation for EDP, where for each $1 \leq i \leq k$, we have an indicator variable $x_i$ for whether or not we route the pair $(s_i, t_i)$. Let $P_i$ denote the set of all paths connecting $s_i$ to $t_i$ in $G$. The LP relaxation is defined as follows.
(LP) \( \max \sum_{i=1}^{k} x_i \)
\[
\text{s.t.} \\
\sum_{P \in \mathcal{P}} f(P) \geq x_i \quad \forall 1 \leq i \leq k \\
\sum_{P \in \mathcal{P}} f(P) \leq 1 \quad \forall e \in E \\
0 \leq x_i \leq 1 \quad \forall 1 \leq i \leq k \\
f(P) \geq 0 \quad \forall 1 \leq i \leq k, \forall P \in \mathcal{P}_i
\]

While this LP has exponentially many variables, it can be efficiently solved using standard techniques, e.g. by using an equivalent polynomial-size edge-based LP formulation. Throughout the rest of the paper, we denote by \( \text{OPT} \) the value of the optimal solution to the LP. Clearly, the value of the optimal solution to the EDP problem instance is at most \( \text{OPT} \). The next theorem follows from the work of Chekuri, Khanna and Shepherd [CKS13, CKS05], and we provide a short proof sketch in the Appendix for completeness.

**Theorem 3.1** Suppose we are given a graph \( G = (V, E) \) and a set \( \mathcal{M} \) of \( k \) source-sink pairs in \( G \). Then we can efficiently partition \( G \) into a collection \( G_1, \ldots, G_\ell \) of vertex-disjoint induced sub-graphs, and compute, for each \( 1 \leq i \leq \ell \), a collection \( \mathcal{M}_i \subseteq \mathcal{M} \) of source-sink pairs contained in \( G_i \), such that \( \sum_{i=1}^\ell |\mathcal{M}_i| = \Omega(\text{OPT} / \log^2 k) \), and moreover, if for each \( 1 \leq i \leq \ell \), \( \mathcal{T}_i \) denotes the set of terminals participating in pairs in \( \mathcal{M}_i \), then \( \mathcal{T}_i \subseteq V(G_i) \), and \( \mathcal{T}_i \) is \( \frac{1}{2} \)-well-linked in \( G_i \).

We now proceed to solve the problem on each one of the graphs \( G_i \) separately. We assume without loss of generality that \( G_i \) is connected. In order to simplify the notation, we denote the graph \( G_i \) by \( G \), the set \( \mathcal{M}_i \) of the source-sink pairs by \( \mathcal{M} \), and the set \( \mathcal{T}_i \) of terminals by \( \mathcal{T} \). For simplicity, we denote \( |\mathcal{M}| = k \). Recall that \( \mathcal{T} \) is \( \frac{1}{2} \)-well-linked in \( G \), the degree of every terminal in \( \mathcal{T} \) is 1, and the maximum vertex degree in \( G \) is at most 4. It is now enough to prove that we can route \( \Omega\left(\frac{k}{\log \alpha \cdot k \log \log k}\right) \) demand pairs in \( \mathcal{M} \) with congestion at most 14. We also assume that \( k > k_0 \), where \( k_0 \) is a large enough constant: otherwise, we can simply pick any source-sink pair \((s, t) \in \mathcal{M}\), connect it with any path \( P \) and output this as a solution. In particular, we will assume that \( k > \log^{24} k \), and \( \gamma_{\text{CMG}}(k) = \Theta(\log^2 k) > 20 \).

Throughout the algorithm, we let \( \gamma = \gamma_{\text{CMG}}(k) = \Theta(\log^2 k) \), and we use a parameter \( k_1 = \frac{k}{1927^2 \log \gamma} = \Omega\left(\frac{k}{\log^6 k \log \log k}\right) \). We say that a cluster \( C \subseteq V(G) \) is small if \( |\text{out}_G(C)| \leq k_1 \), and we say that it is large otherwise.

### Families of Good Routers

We define a family of good routers in graph \( G \). We then proceed in two steps. First, we show that we can efficiently find a family of good routers. Next, we show that given such a family, we can find the desired routing of a subset of the source-sink pairs in \( \mathcal{M} \).

**Definition 3.1** We say that a subset \( S \subseteq V(G) \setminus \mathcal{T} \) of vertices is a good router iff there is a subset \( \Gamma \subseteq \text{out}_G(S) \) of edges, with \( |\Gamma| = k_1 \), such that:

- \( S \) has the \( \alpha_{\text{bw}}(k) \)-bandwidth property for \( \Gamma \). That is, for any partition \((X, Y) \) of \( S \), if \( \Gamma_X = \Gamma \cap \text{out}(X) \) and \( \Gamma_Y = \Gamma \cap \text{out}(Y) \), then \( |E_G(X, Y)| \geq \alpha_{\text{bw}}(k) \cdot \min\{|\Gamma_X|, |\Gamma_Y|\} \).
There is a flow $F$ in graph $G$, where every edge $e \in \Gamma$ sends one flow unit to a distinct terminal $t_e \in T$ (so for $e \neq e', t_e \neq t_{e'}$), and the congestion caused by $F$ is at most $2\beta(k) / \alpha_{BW}(k) = O(\log^{4.5} k)$.

We say that a family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of $\gamma = \gamma_{CMG}(k) = \Theta(\log^2 k)$ subsets of vertices is a family of good routers, iff each subset $S_j$ is a good router, and $S_1, \ldots, S_\gamma$ are pairwise disjoint.

We view the subset $\Gamma \subseteq \text{out}_G(S)$ of edges as part of the definition of a family of good routers. In particular, when we say that we are given a family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of good routers, we assume that we are also given the corresponding subsets $\Gamma_j \subseteq \text{out}_G(S_j)$ of edges, for all $1 \leq j \leq \gamma$. The rest of the proof consists of two parts. First, we show how to find a family of good routers in $G$. Next, we exploit this family to embed an expander into $G$, and find the desired routing.

### 3.1 Finding a Family of Good Routers

The goal of this section is to prove the following theorem.

**Theorem 3.2** There is an efficient randomized algorithm that with high probability computes a family of good routers in graph $G$.

In order to prove Theorem 3.2, we use the notions of good clustering and legal contracted graphs. Intuitively, these notions allow us to “hide” some irrelevant parts of our graph by contracting some connected clusters into super-nodes. Our algorithm will perform a number of iterations. In every iteration, we start from some legal contracted graph, and either produce a family of good routers, or contract the graph even further. This guarantees that after polynomially many iterations, we will produce a family of good routers. We now turn to define the notions of good clustering and legal contracted graph.

We say that a partition $C$ of $V(G)$ is a good clustering, iff:

- Each terminal $t \in T$ belongs to a cluster that contains only $t$. That is, for each $t \in T$, $\{t\} \in C$.
- For each cluster $C \in C$, $|\text{out}_G(C)| \leq k_1$, and the set $C$ has the $\alpha_{BW}(k)$-bandwidth property. (In particular, since $G$ is connected, due to the bandwidth property, so is $G[C]$.)

Given a good clustering $C$, we let $G' = G_C$ be obtained from $G$, by contracting each cluster $C \in C$ into a super-node $v_C$. We delete self-loops, but keep parallel edges. A graph $G'$ obtained in this manner from a good clustering $C$ is called a legal contracted graph.

Notice that graph $G'$ may contain parallel edges, and $T$ remains $\frac{1}{2}$-well-linked in $G'$. Also, since the maximum vertex degree in $G$ is 4, the maximum vertex degree in $G'$ is at most $k_1$, and every terminal has degree 1. Every edge in graph $G'$ corresponds to some edge in the original graph $G$, and we will not distinguish between them. In particular, for every vertex subset $S' \subseteq V(G')$, if $S \subseteq V(G)$ is the corresponding subset of vertices in $G$, where every super-node $v_C \in S' \cap V_2$ is replaced by the vertices of $C$, then there is a one-to-one mapping between $\text{out}_{G'}(S')$ and $\text{out}_G(S)$, and we will identify the edges in these two sets, that is, $\text{out}_{G'}(S') = \text{out}_G(S)$. We need the following simple claim.

**Claim 3.3** If $G'$ is a legal contracted graph for $G$, then $G' \setminus T$ contains at least $k/6$ edges.
Proof: For each terminal \( t \in \mathcal{T} \), let \( e_t \) be the unique edge adjacent to \( t \) in \( G' \), and let \( u_t \) be the other endpoint of \( e_t \). We partition the terminals in \( \mathcal{T} \) into groups, where two terminals \( t, t' \) belong to the same group iff \( u_t = u_{t'} \). Let \( \mathcal{G} \) be the resulting partition of the terminals. Since the degree of every vertex in \( G' \) is at most \( k_1 \), each group \( U \in \mathcal{G} \) contains at most \( k_1 \) terminals. Next, we partition the terminals in \( \mathcal{T} \) into two subsets \( X, Y \), where \( |X|, |Y| \geq k/3 \), and for each group \( U \in \mathcal{G} \), either \( U \subseteq X \) or \( U \subseteq Y \) holds. It is possible to find such a partition by greedily processing each group \( U \in \mathcal{G} \), and adding all terminals of \( U \) to one of the subsets \( X \) or \( Y \), that currently contains fewer terminals. Finally, we remove terminals from set \( X \) until \( |X| = k/3 \), and we do the same for \( Y \). Since the terminals are \( \frac{1}{2} \)-well-linked in \( G' \), it is possible to route \( k/3 \) flow units from the terminals in \( X \) to the terminals in \( Y \), with edge-congestion at most 2. Since no group \( U \) is split between the two sets \( X \) and \( Y \), each flow-path must contain at least one edge of \( G' \setminus \mathcal{T} \). Therefore, the number of edges in \( G' \setminus \mathcal{T} \) is at least \( k/6 \).

In order to prove Theorem 3.2, we use the following theorem.

**Theorem 3.4** Let \( C \) be a good clustering of \( G \). Then there is an efficient randomized algorithm that with high probability either returns a family \( F = \{S_1, \ldots, S_{\gamma} \} \) of good routers in \( G \), together with the corresponding subsets \( \Gamma_j \subseteq \text{out}_G(S_j) \) of edges for all \( 1 \leq j \leq \gamma \), or finds a good clustering \( C' \) of \( G \), with \( \sum_{C \in C'} |\text{out}_G(C)| < \sum_{C \in C} |\text{out}_G(C)| \).

We prove Theorem 3.4 below, and complete the proof of Theorem 3.2 here first. We start with the good clustering \( C = \{\{v\} \mid v \in V(G)\} \), where every vertex belongs to a separate cluster, and repeatedly apply Theorem 3.4 to it. In every iteration, if Theorem 3.4 returns a family \( F \) of good routers, then we terminate the algorithm and return \( F \). Otherwise, we replace \( C \) with \( C' \), and continue to the next iteration. Clearly, after at most \( 2|E(G)| \) iterations, the algorithm will produce a family of good routers with high probability. From now on we focus on the proof of Theorem 3.4.

Let \( G' = G_C \) be the legal contracted graph corresponding to \( C \), and let \( m \) be the number of edges in \( G' \setminus \mathcal{T} \). From Claim 3.3, \( m \geq k/6 \). The proof consists of two steps. First, we randomly partition the vertices in \( G' \setminus \mathcal{T} \) into \( \gamma \) subsets \( X_1, \ldots, X_\gamma \). We show that with high probability, for each \( 1 \leq j \leq \gamma \), \( |\text{out}_{G'}(X_j)| < \frac{10m}{7} \), while the number of edges with both endpoints in \( X_j \), \( |E_{G'}(X_j)| \geq \frac{m}{2\gamma} \). Therefore, \( |E_{G'}(X_j)| > \frac{|\text{out}_{G'}(X_j)|}{20\gamma} \) with high probability. For each \( j : 1 \leq j \leq \gamma \), we then try to recover a good router \( S_j \) from the cluster \( X_j \). If we succeed, then we obtain a family \( F = \{S_1, \ldots, S_{\gamma} \} \) of good routers. If we fail to recover a good router for some \( 1 \leq j \leq \gamma \), then we will produce a new good clustering \( C' \), with \( \sum_{C \in C'} |\text{out}_G(C)| < \sum_{C \in C} |\text{out}_G(C)| \).

We start with the first step. We partition the vertices in \( V(G') \setminus \mathcal{T} \) into subsets \( X_1, \ldots, X_\gamma \), where each vertex \( v \in V(G') \setminus \mathcal{T} \) selects an index \( 1 \leq j \leq \gamma \) independently uniformly at random, and is then added to \( X_j \). We need the following claim.

**Claim 3.5** With probability at least \( \frac{1}{2} \), for each \( 1 \leq j \leq \gamma \), \( |\text{out}_{G'}(X_j)| < \frac{10m}{7} \), while \( |E_{G'}(X_j)| \geq \frac{m}{2\gamma} \).

**Proof:** Let \( H = G' \setminus \mathcal{T} \). Fix some \( 1 \leq j \leq \gamma \). Let \( \mathcal{E}_1(j) \) be the bad event that \( \sum_{v \in X_j} d_H(v) \geq \frac{2m}{\gamma} \cdot \left( 1 + \frac{1}{\gamma} \right) \). In order to bound the probability of \( \mathcal{E}_1(j) \), we define, for each vertex \( v \in V(H) \), a random variable \( x_v \), whose value is \( \frac{d_H(v)}{k_1} \) if \( v \in X_j \) and 0 otherwise. Notice that \( x_v \in [0, 1] \), and the random variables \( \{x_v\} \) are pairwise independent. Let \( B = \sum_{v \in V(H)} x_v \). Then the expectation of \( B \), \( \mu_1 = \sum_{v \in V(H)} \frac{d_H(v)}{\gamma k_1} = \frac{2m}{\gamma k_1} \). Using the standard Chernoff bound (see e.g. Theorem 1.1 in [DP09]),

\[
\Pr[\mathcal{E}_1(j)] = \Pr[B > (1 + 1/\gamma) \mu_1] \leq e^{-\mu_1/(3\gamma^2)} = e^{-\frac{2m}{3\gamma^2 k_1}} < \frac{1}{6\gamma}
\]
since $m \geq k/6$ and $k_1 = \frac{k}{192 \gamma^3 \log \gamma}$.

For each terminal $t \in \mathcal{T}$, let $e_t$ be the unique edge adjacent to $t$ in graph $G'$, and let $u_t$ be its other endpoint. Let $U = \{u_t \mid t \in \mathcal{T}\}$. For each vertex $u \in U$, let $w(u)$ be the number of terminals $t$, such that $u = u_t$. Notice that $w(u) \leq k_1$ must hold. We say that a bad event $E_2(j)$ happens iff

$$\sum_{u \in U \cap X_j} w(u) \geq \frac{k}{\gamma} \left(1 + \frac{1}{\gamma}\right).$$

In order to bound the probability of the event $E_2(j)$, we define, for each $u \in U$, a random variable $y_u$, whose value is $w(u)/k_1$ if $u \in X_j$, and it is 0 otherwise. Notice that $y_u \in [0, 1]$, and the variables $y_u$ are independent for all $u \in U$. Let $Y = \sum_{u \in U} y_u$. The expectation of $Y$ is $\mu_2 = \frac{k}{k_1 \gamma}$, and event $E_3(j)$ holds iff $Y \geq \frac{k}{k_1 \gamma} \left(1 + \frac{1}{\gamma}\right) \geq \mu_2 \cdot \left(1 + \frac{1}{\gamma}\right)$. Using the standard Chernoff bound again, we get that:

$$\Pr [E_2(j)] \leq e^{-\mu_2/(3\gamma^2)} \leq e^{-k/(3k_1 \gamma^3)} \leq \frac{1}{6\gamma}$$

since $k_1 = \frac{k}{192 \gamma^3 \log \gamma}$. Notice that if events $E_1(j), E_2(j)$ do not happen, then:

$$|\text{out}_{G^r}(X_j)| \leq \sum_{v \in X_j} d_{H}(v) + \sum_{u \in U \cap X_j} w(u) \leq \left(1 + \frac{1}{\gamma}\right) \left(\frac{2m}{\gamma} + \frac{k}{\gamma}\right) < \frac{10m}{\gamma}$$

since $m \geq k/6$.

Let $E_3(j)$ be the bad event that $|E_{G^r}(X_j)| < \frac{m}{2k_1}$. We next prove that $\Pr [E_3(j)] \leq \frac{1}{6\gamma}$. We say that two edges $e, e' \in E(G' \setminus \mathcal{T})$ are independent if they do not share any endpoints. Our first step is to compute a partition $U_1, \ldots, U_r$ of the set $E(G' \setminus \mathcal{T})$ of edges, where $r \leq 2k_1$, such that for each $1 \leq i \leq r$, $|U_i| \geq \frac{m}{4k_1}$, and all edges in set $U_i$ are mutually independent. In order to compute such a partition, we construct an auxiliary graph $Z$, whose vertex set is $\{v_e \mid e \in E(H)\}$, and there is an edge $(v_e, v_{e'})$ iff $e$ and $e'$ are not independent. Since the maximum vertex degree in $G'$ is at most $k_1$, the maximum vertex degree in $Z$ is bounded by $2k_1 - 2$. We use the following theorem due to Hajnal and Szemerédi [HS70].

**Theorem 3.6** [HS70] For any graph $H$ with maximum vertex degree $\Delta$, there is a partition $S_1, \ldots, S_r$ of $V(H)$ into $r \leq \Delta + 1$ disjoint subsets, such that for each $1 \leq i \leq r$, $S_i$ is an independent set, and for $1 \leq i \neq i' \leq r$, $|S_i| - |S_{i'}| \leq 1$.

From Theorem 3.6, we can find a partition $V_1, \ldots, V_r$ of the vertices of $Z$ into $r \leq 2k_1$ subsets, where each subset $V_i$ is an independent set, and $|V_i| \geq \frac{|V[Z]|}{r} - 1 \geq \frac{m}{4k_1}$. The partition $V_1, \ldots, V_r$ of the vertices of $Z$ gives the desired partition $U_1, \ldots, U_r$ of the edges of $G' \setminus \mathcal{T}$. For each $1 \leq i \leq r$, we say that the bad event $E_3^r(j)$ happens iff $|U_i \cap E(X_j)| < \frac{|U_i|}{2k_1}$. Notice that if $E_3(j)$ happens, then event $E_3^r(j)$ must happen for some $1 \leq i \leq r$. Fix some $1 \leq i \leq r$. The expectation of $|U_i \cap E(X_j)|$ is $\mu_3 = \frac{|U_i|}{2k_1}$. Since all edges in $U_i$ are independent, we can use the standard Chernoff bound to bound the probability of $E_3^r(j)$, as follows:

$$\Pr [E_3^r(j)] = \Pr [U_i \cap E(X_j) < \mu_3/2] \leq e^{-\mu_3/8} = e^{-\frac{|U_i|}{16k_1^2}}$$

Since $|U_i| \geq \frac{m}{4k_1}, m \geq k/6, k_1 = \frac{k}{192 \gamma^3 \log \gamma}$, and $\gamma = \Theta(\log^2 k)$, this is bounded by $\frac{1}{12k_1 \gamma}$. We conclude that $\Pr [E_3^r(j)] \leq \frac{1}{12k_1 \gamma}$, and by using the union bound over all $1 \leq i \leq r$, $\Pr [E_3(j)] \leq \frac{1}{6\gamma}$. 

17
Using the union bound over all $1 \leq j \leq \gamma$, with probability at least $\frac{1}{2}$, none of the events $E_1(j), E_2(j), E_3(j)$ for any $1 \leq j \leq \gamma$ happen, and so for each $1 \leq j \leq \gamma$, $|\text{out}_{G'}(X_j)| < \frac{10m}{\gamma}$, and $|E_{G'}(X_j)| \geq \frac{m}{2\gamma}$ must hold.

Given a partition $X_1, \ldots, X_\gamma$, we can efficiently check whether the conditions of Claim 3.5 hold. If they do not hold, we repeat the randomized partitioning procedure. From Claim 3.5, we are guaranteed that with high probability, after poly($n$) iterations, we will obtain a partition with the desired properties. Assume now that we are given the partition $X_1, \ldots, X_\gamma$ of $V(G') \setminus \mathcal{T}$, for which the conditions of Claim 3.5 hold. Then for each $1 \leq j \leq \gamma$, $|E_{G'}(X_j)| > \frac{|\text{out}_{G'}(X_j)|}{20\gamma}$. Let $X'_j \subseteq V(G) \setminus \mathcal{T}$ be the set obtained from $X_j$, after we un-contract each cluster, that is, $X'_j = \bigcup_{v_C \in X_j} C$. Notice that $\left\{ X'_j \right\}_{j=1}^{\gamma}$ is a partition of $V(G) \setminus \mathcal{T}$. We use the following theorem to finish the proof of Theorem 3.4.

Theorem 3.7 There is an efficient algorithm, that for each $1 \leq j \leq \gamma$, either finds a good router $S_j \subseteq X'_j$, or produces a good clustering $\mathcal{C}'$ of $G$, with $\sum_{C \in \mathcal{C}} |\text{out}_{G}(C)| < \sum_{C \in \mathcal{C}} |\text{out}_{G}(C)|$.

In order to complete the proof of Theorem 3.4, we apply Theorem 3.7 to each cluster $X_j$ in turn. If, for some $1 \leq j \leq \gamma$, Theorem 3.7 produces a good clustering $\mathcal{C}'$, with $\sum_{C \in \mathcal{C}'} |\text{out}_{G}(C)| < \sum_{C \in \mathcal{C}} |\text{out}_{G}(C)|$, then we are done. Otherwise, for each $1 \leq j \leq \gamma$, we obtain a good router $S_j \subseteq X'_j$, and, since the sets $X'_1, \ldots, X'_\gamma$ are mutually disjoint, we obtain a family $\mathcal{F} = \{S_1, \ldots, S_\gamma\}$ of good routers. In order to complete the proof of Theorem 3.4, it now remains to prove Theorem 3.7.

Proof of Theorem 3.7. Recall that we say that a cluster $S \subseteq V(G)$ is large if $|\text{out}_{G}(S)| > k_1$, and we say that it is small otherwise. Fix some $1 \leq j \leq \gamma$. We partition the set $\mathcal{C}$ of clusters into two subsets: $\mathcal{W}_1$ contains all clusters $C$ with $C \cap X'_j = \emptyset$, and $\mathcal{W}_2$ contains all clusters $C \subseteq X'_j$. Throughout the algorithm, we also maintain another partition $\mathcal{C}$ of $V(G)$ into clusters (which may not be a good clustering). We will also maintain a partition $(\mathcal{W}_1, \mathcal{W}_2)$ of $\mathcal{C}$, where $\mathcal{W}_1 = \mathcal{C} \cap \mathcal{W}_1$, and $\mathcal{W}_2 = \mathcal{C} \setminus \mathcal{W}_1$. Recall that from Claim 3.5, $\sum_{C \in \mathcal{W}_2} |\text{out}_{G}(C)| \geq |\text{out}_{G}(X'_j)| + 2|E_{G'}(X_j)| > |\text{out}_{G}(X'_j)| \left( 1 + \frac{1}{10\gamma} \right)$. Our goal is to either find a good router $S_j \subseteq X'_j$, or to produce a good clustering $\mathcal{C}'$, where $\sum_{C \in \mathcal{W}_2} |\text{out}_{G}(C)| < |\text{out}_{G}(X'_j)| \left( 1 + \frac{1}{10\gamma} \right)$. This will ensure that $\sum_{C \in \mathcal{C}} |\text{out}_{G}(C)| < \sum_{C \in \mathcal{C}} |\text{out}_{G}(C)|$.

Assume now that we are given some large cluster $C \subseteq V(G) \setminus \mathcal{T}$, and a current partition $\mathcal{C}$ of $V(G)$ into clusters. We say that a partition $(A, B)$ of $V(G)$ is a small canonical separator for $C$, iff $C \subseteq A$, $\mathcal{T} \subseteq B$, $|E_{G}(A, B)| \leq k_1$, and for each cluster $C' \in \mathcal{C}$, either $C' \subseteq A$, or $C' \subseteq B$ holds.

Our algorithm proceeds in two steps. In the first step, we either find a good router $S_j \subseteq X'_j$, or compute a partition $\mathcal{C}$ of $V(G)$ into clusters, such that each large cluster $C \in \mathcal{C}$ has a small canonical separator. If we find a good router $S_j \subseteq X'_j$, then we terminate the algorithm and return $S_j$. Otherwise, we continue to the second step, that transforms the partition $\mathcal{C}$ into a good clustering.

Step 1. Given a partition $\mathcal{C}$ of $V(G)$, a cluster $C \in \mathcal{C}$ is called a candidate cluster, if it is a large cluster, and there is no small canonical separator $(A, B)$ for $C$. Notice that in this case, there is a flow $F$ in the corresponding contracted graph $G'_{\mathcal{C}}$, of value at least $k_1$, from $v_C$ to $\mathcal{T}$. This step is summarized in the following claim.

Claim 3.8 There is an efficient algorithm, that either finds a good router $S_j \subseteq X'_j$, or computes a partition $\mathcal{R}$ of $X'_j$, such that $\sum_{C \in \mathcal{R}} |\text{out}_{G}(C)| \leq |\text{out}_{G}(X'_j)| \left( 1 + \frac{1}{10\gamma} \right)$, and, if $\mathcal{C} = \mathcal{R} \cup \mathcal{W}_1$, then no cluster of $\mathcal{C}$ is a candidate cluster.
Proof: We start with \( \mathcal{R} = \{ X'_j \} \). Throughout the algorithm, we maintain a partition \( \hat{\mathcal{C}} \) of \( V(G) \) into clusters, where \( \hat{\mathcal{C}} = \mathcal{R} \cup \mathcal{W}_1 \). While \( \hat{\mathcal{C}} \) contains at least one candidate cluster, let \( S \) be any such cluster. Notice that \( S \in \mathcal{R} \) must hold, as every cluster in \( \mathcal{W}_1 \) is small. For simplicity, we denote \( G_{\hat{\mathcal{C}}} \) by \( \hat{G} \). From the integrality of flow, there is a collection \( \mathcal{P} \) of \( k_1 \) edge-disjoint paths in \( \hat{G} \), connecting distinct edges in \( \text{out}_{\hat{G}}(S) \) to distinct terminals in \( \mathcal{T} \). Let \( \Gamma \subseteq \text{out}_{\hat{G}}(S) \) be the set of \( k_1 \) edges which serve as endpoints of the paths in \( \mathcal{P} \). We set up an instance of the sparsest cut problem in graph \( G[S] \cup \text{out}_{\hat{G}}(S) \), where the edges in set \( \Gamma \) serve as terminals. We then run the algorithm \( \mathcal{A}_{\text{ARV}} \) on the resulting instance. If the algorithm returns a cut \((Y,Z)\) of sparsity less than \( \alpha(k) \), then \((Y,Z)\) is a \((k,\alpha(k))\)-violating cut for \( S \). We then replace \( S \) with \( Y \) and \( Z \) in \( \mathcal{R} \), and update \( \hat{\mathcal{C}} \) and \( \hat{G} \) accordingly. This ends the current iteration, and we then proceed to the next iteration. Assume now that algorithm \( \mathcal{A}_{\text{ARV}} \) returns a cut whose sparsity is at least \( \alpha(k) \). Then we are guaranteed that \( S \) has the \( \alpha_{\text{bw}}(k) = \alpha(k)/\alpha_{\text{ARV}}(k) \)-bandwidth property in graph \( \Gamma \). Recall that we are given a set \( \mathcal{P} \) of \( k_1 \) edge-disjoint paths connecting the edges in \( \Gamma \) to the terminals \( \mathcal{T} \) in graph \( \hat{G} \), where each path connects a distinct edge \( e \in \Gamma \) to a distinct terminal \( t_e \in \mathcal{T} \). In order for \( S \) to be a good router, a low-congestion flow connecting the edges in \( \Gamma \) to the terminals must exist in the original graph \( G \). We will try to find this flow, as follows. The flow will follow the paths in \( \mathcal{P} \), except that we need to specify how the flow is routed inside each cluster \( C \in \hat{\mathcal{C}} \). Observe that for each such cluster \( C \), the paths in \( \mathcal{P} \) define a set \( D_C \) of 1-restricted demands on the edges of \( \text{out}_{\hat{G}}(C) \). Moreover, the total number of edges in \( \text{out}_{\hat{G}}(C) \) participating in the paths in \( \mathcal{P} \) is at most \( k_1 \), as there are only \( k_1 \) paths in \( \mathcal{P} \) and we can assume without loss of generality that they are simple. If \( C \in \mathcal{W}_1 \), then we are guaranteed that cluster \( C \) has the \( \alpha_{\text{bw}}(k) \)-bandwidth property in graph \( G \). From Observation 2.4, we can route the set \( D_C \) of demands inside \( G[C] \) with congestion at most \( 2\beta(k)/\alpha_{\text{bw}}(k) \). Otherwise, \( C \in \mathcal{R} \), and it is possible that we cannot route the set \( D_C \) of demands inside \( G[C] \) with congestion at most \( 2\beta(k)/\alpha_{\text{bw}}(k) \). We then proceed as follows. If, for each such cluster \( C \in \mathcal{R} \), we can route the set \( D_C \) of demands inside \( G[C] \) with congestion at most \( 2\beta(k)/\alpha_{\text{bw}}(k) \), then \( S \) is a good router. We terminate the algorithm and return \( S \). Otherwise, let \( C \in \mathcal{R} \) be any cluster, for which such flow does not exist. Consider the instance of the sparsest cut problem defined on the graph \( G[C] \cup \text{out}_{\hat{G}}(C) \), where the edges of \( \text{out}_{\hat{G}}(C) \) with non-zero demand serve as terminals (recall that there are at most \( 2k_1 \) such edges). Then the value of the sparsest cut in this instance is at most \( \alpha_{\text{bw}}(k) \), and so by applying algorithm \( \mathcal{A}_{\text{ARV}} \) to this instance of sparsest cut, we will obtain a \((k,\alpha(k))\)-violating cut \((Y,Z)\) for set \( C \). We then remove \( C \) from \( \mathcal{R} \), and add \( Y \) and \( Z \) to \( \hat{\mathcal{C}} \) instead. We also update \( \hat{\mathcal{C}} \) and \( \hat{G} \) accordingly, and end the current iteration. This concludes the description of the algorithm. Notice that the final set \( \mathcal{R} \) was obtained from \( X'_j \) by running the bandwidth-decomposition algorithm from Theorem 2.5 on \( X'_j \), as in each step we computed a \((k,\alpha(k))\)-violating cut of some cluster in the current partition \( \mathcal{R} \) of \( X'_j \). Therefore, from Theorem 2.5, \( \sum_{C \in \mathcal{R}} |\text{out}_G(C)| \leq |\text{out}_G(X'_j)| \left( 1 + \frac{1}{64\gamma} \right) \). If the algorithm does not terminate with a good router \( S \subseteq X'_j \), then it is guaranteed to terminate after \( O(|V(G)|) \) iterations, with a partition \( \mathcal{R} \) of \( X'_j \), and a corresponding partition \( \hat{\mathcal{C}} \) of \( V(G) \), such that \( \hat{\mathcal{C}} \) contains no candidate clusters.

Step 2 In this step, we transform the partition \( \hat{\mathcal{C}} \) of \( V(G) \) computed in Step 1 into a good clustering. We start with the following claim.

**Claim 3.9** Let \( \hat{\mathcal{C}} \) be the partition of \( V(G) \) computed in Step 1. Then there is an efficient algorithm to find a collection \( \mathcal{S} \) of disjoint small clusters, such that each cluster \( S \in \mathcal{S} \) is contained in \( V(G) \setminus \mathcal{T} \),
and for each large cluster \(C \in \tilde{C}\), there is some cluster \(S \in \mathcal{S}\) with \(C \subseteq S\). Moreover, for every pair \(C' \in \tilde{C}\) and \(S \in \mathcal{S}\) of clusters, either \(C' \subseteq S\), or \(C' \cap S = \emptyset\).

**Proof:** Throughout the algorithm, we maintain a collection \(\mathcal{S}\) of disjoint subsets of non-terminal vertices of \(G\), where each set \(S \in \mathcal{S}\) is a small cluster. We will ensure throughout the algorithm that for each pair \(C' \in \tilde{C}\) and \(S \in \mathcal{S}\) of clusters, either \(C' \subseteq S\), or \(C' \cap S = \emptyset\). We say that a large cluster \(C \in \tilde{C}\) is **covered** by \(\mathcal{S}\), if there is a cluster \(S \in \mathcal{S}\) with \(C \subseteq S\). We start with \(S = \emptyset\).

While there is a large cluster in \(\tilde{C}\), that is not covered by \(\mathcal{S}\), let \(C\) be any such cluster. Recall that there is a small canonical separator \((A, B)\) for \(C\) in \(G\). Let \(\mathcal{L} \subseteq \tilde{C}\) be the set of large clusters covered by \(\mathcal{S} \cup \{A\}\). We now consider clusters \(S \in \mathcal{S}\) one-by-one. For each such cluster \(S\), if \(S \setminus A\) is a small cluster, then we replace \(S\) with \(S \setminus A\) in \(\mathcal{S}\). Notice that all clusters of \(\mathcal{L}\) remain covered by the new set \(\mathcal{S} \cup \{A\}\). Otherwise, if \(S \setminus A\) is a large cluster, then, from sub-modularity of cuts, \(|\text{out}(S)| + |\text{out}(A)| \geq |\text{out}(S \setminus A)| + |\text{out}(A \setminus S)|\), and since \(S\) and \(A\) are small clusters, \(A \setminus S\) is a small cluster. We replace \(A\) with \(A \setminus S\), and continue. Notice that all clusters of \(\mathcal{L}\) remain covered by the new set \(\mathcal{S} \cup \{A\}\). Once all clusters of \(\mathcal{L}\) are processed, we add the final set \(A\) to \(\mathcal{S}\), and continue to the next iteration. It is easy to see that in each iteration, the number of large clusters in \(\tilde{C}\) that are not covered by \(\mathcal{S}\) decreases by at least 1, and we maintain the property that for each pair \(C' \in \tilde{C}\) and \(S \in \mathcal{S}\) of clusters, either \(C' \subseteq S\), or \(C' \cap S = \emptyset\). Once all large clusters of \(\tilde{C}\) are covered, we obtain the desired set \(\mathcal{S}\) of small clusters.

We assume without loss of generality, that for each cluster \(S \in \mathcal{S}\), there is some large cluster \(C_S \in \tilde{C}\) with \(C_S \subseteq S\), since otherwise we can remove \(S\) from \(\mathcal{S}\). Let \(\tilde{C}'\) be a partition of \(V(G)\), obtained from \(\tilde{C}\) as follows: for each cluster \(S \in \mathcal{S}\), we delete all clusters \(C \subseteq S\) from \(\tilde{C}\), and we add \(S\) to \(\tilde{C}'\) instead. Notice that every cluster in \(\tilde{C}'\) is now a small cluster. We let \(\tilde{W}_1 = \tilde{C}' \cap \mathcal{W}_1\), and \(\tilde{W}_2 = \tilde{C}' \setminus \mathcal{W}_1\). Notice that for every cluster \(S \in \tilde{W}_2\), either \(S \in \mathcal{R}\), or there is some large cluster \(C_S \in \mathcal{R}\), such that \(C_S \subseteq S\). In the latter case, since \(C_S\) is a large cluster, and \(S\) is a small cluster, \(|\text{out}_G(S)| < |\text{out}_G(C_S)|\).

Therefore,

\[
\sum_{S \in \tilde{W}_2} |\text{out}_G(S)| \leq \sum_{C \in \mathcal{R}} |\text{out}_G(C)| \leq |\text{out}_G(X_j')| \left(1 + \frac{1}{64\gamma}\right)
\]

Our final step is to compute, for each cluster \(C \in \tilde{W}_2\), a partition \(\mathcal{W}(C)\) into small clusters, that have the \(\alpha_{\text{bw}}(k)\)-bandwidth property, such that \(\sum_{C' \in \mathcal{W}(C)} |\text{out}_G(C')| \leq |\text{out}_G(C)| \left(1 + \frac{1}{64\gamma}\right)\), using Corollary 2.6. Let \(\mathcal{R}' = \bigcup_{C \in \tilde{W}_2} \mathcal{W}(C)\). Then:

\[
\sum_{C' \in \mathcal{R}'} |\text{out}_G(C')| \leq \sum_{C \in \tilde{W}_2} |\text{out}_G(C)| \left(1 + \frac{1}{64\gamma}\right) \leq |\text{out}_G(X_j')| \left(1 + \frac{1}{64\gamma}\right)^2 < |\text{out}_G(X_j')| \left(1 + \frac{1}{20\gamma}\right).
\]

Our final good partition \(\mathcal{C}'\) of \(V(G)\) is the union of \(\tilde{W}_1\) and \(\mathcal{R}'\). It is immediate to verify that this is indeed a good partition. Moreover,
\[ \sum_{C \in \mathcal{C}} |\text{out}_G(C)| \leq \sum_{C \in \mathcal{W}_1} |\text{out}_G(C)| + \sum_{C \in \mathcal{R}} |\text{out}_G(C)| \]
\[ < \sum_{C \in \mathcal{W}_1} |\text{out}_G(C)| + |\text{out}_G(X_j)'\rangle \left(1 + \frac{1}{20\gamma}\right) \]
\[ < \sum_{C \in \mathcal{W}_1} |\text{out}_G(C)| + \sum_{C \in \mathcal{W}_2} |\text{out}_G(C)| \]
\[ = \sum_{C \in \mathcal{C}} |\text{out}_G(C)|, \]

since \( \sum_{C \in \mathcal{W}_2} |\text{out}_G(C)| = |\text{out}_G(X_j)'\rangle | + 2|E_G'(X_j)| \geq |\text{out}_G(X_j)'\rangle \left(1 + \frac{1}{10\gamma}\right) \).

### 3.2 Finding the Routing

In this section, we exploit the family \( \mathcal{F} \) of good routers, in order to find a routing of \( \Omega\left(\frac{k}{\log^{2+\delta} k \log \log k}\right) \) pairs in \( \mathcal{M} \) with congestion at most 14.

We assume that we are given a family \( \mathcal{F} = \{S_1, \ldots, S_\gamma\} \) of good routers in \( G \). For each \( 1 \leq j \leq \gamma \), we are also given a subset \( \Gamma_j \subseteq \text{out}_G(S_j) \) of edges, such that \( S_j \) has the \( \alpha_{bw}(k) \)-bandwidth property for \( \Gamma_j \), and there is a flow \( F_j : \Gamma_j \sim \eta \mathcal{T} \), where each edge \( e \in \Gamma_j \) sends one flow unit to a distinct terminal \( t_e \), and the total congestion due to \( F_j \) is at most \( \eta = 2\beta(k)/\alpha_{bw}(k) \).

In order to find the final routing, we build an expander over a subset of terminals and embed it into graph \( G \). More precisely, we select an arbitrary subset \( \mathcal{M}' \subseteq \mathcal{M} \) of \( k' \) source-sink pairs, for some \( k' = k/\text{poly} \log k \), whose precise value is defined later. Let \( \mathcal{T}' \subseteq \mathcal{T} \) be the subset of terminals participating in pairs in \( \mathcal{M}' \), and assume that \( \mathcal{T}' = \{t_1, \ldots, t_{k'}\} \). We construct an expander \( X \) over the set \( \{v_1, \ldots, v_{k'}\} \) of vertices, which is then embedded into the graph \( G \) as follows. For each \( 1 \leq i \leq k' \), we define a connected sub-graph \( C_i \) of \( G \), that represents the vertex \( v_i \) of the expander. For each edge \( e = (v_i, v_j) \in E(X) \), we define a path \( P_e \), connecting a vertex of \( C_i \) to a vertex of \( C_j \) in \( G \). We will ensure that each edge of \( G \) may only appear in a small constant number of the sub-graphs \( C_i \), and a small constant number of the paths \( P_e \). We will also ensure that for each \( 1 \leq i \leq k' \), terminal \( t_i \in C_i \). We will think about the expander vertex \( v_i \) as representing the terminal \( t_i \). The idea is that any *vertex-disjoint* routing of the terminal pairs in the expander \( X \) can now be translated into a low-edge-congestion routing in the original graph \( G \).

We now turn to describe the construction of the expander \( X \) and the connected sub-graphs \( C_1, \ldots, C_{k'} \) that we use to embed \( X \) into \( G \). The construction exploits the family \( \mathcal{F} = \{S_1, \ldots, S_\gamma\} \) of good routers. We construct a collection \( T_1, \ldots, T_{k'} \) of trees in graph \( G \). Each such tree \( T_i \) contains, for each \( 1 \leq j \leq \gamma \), an edge \( e_{i,j} \in \Gamma_j \). For each \( 1 \leq j \leq \gamma \), the edges \( e_{1,j}, e_{2,j}, \ldots, e_{k',j} \) are all distinct, and we think of the edge \( e_{i,j} \) as the representative of the vertex \( v_i \in V(X) \) for the set \( S_j \). In other words, each tree \( T_i \) spans \( \gamma \) representatives of the vertex \( v_i \); one representative \( e_{i,j} \) for each set \( S_j \in \mathcal{F} \). We will ensure that each edge of graph \( G \) only participates in a constant number of such trees. Additionally, we build a set \( \mathcal{P} = \{P_t \mid t \in \mathcal{T}'\} \) of paths, where path \( P_t \) connects the terminal \( t \) to a distinct tree \( T_i \) (so if \( t \neq t' \), then \( t \) and \( t' \) are connected to different trees), and the total congestion caused by paths in \( \mathcal{P} \) is at most 4. We rename the terminals in \( \mathcal{T}' \), so that \( t_i \) denotes the terminal that is connected to the tree \( T_i \). The final sub-graph \( C_i \) of \( G \) is simply the union of the tree \( T_i \) and the path \( P_{t_i} \).

In order to construct the expander \( X \) over the set \( \{v_1, \ldots, v_{k'}\} \) of vertices, we use the cut-matching
game of [KRV09], where we use the sub-graph $G[S_j]$ of $G$ to route the $j$th matching between the corresponding representatives $e_{1,j}, e_{2,j}, \ldots, e_{k',j}$ of the vertices $v_1, \ldots, v_{k'}$, respectively. Recall that we are only guaranteed that sets $\{S_j\}_{j=1}^\gamma$ have the $\alpha_{bw}(k)$-bandwidth property for the edges in $\Gamma_j$, and so in order to route these matchings, we may have to incur the congestion of $\Omega(1/e)$. In the corresponding representatives $e_{1,j}, e_{2,j}, \ldots, e_{k',j}$ of the vertices $v_1, \ldots, v_{k'}$, respectively. Recall that we are only guaranteed that sets $\{S_j\}_{j=1}^\gamma$ have the $\alpha_{bw}(k)$-bandwidth property for the edges in $\Gamma_j$, and so in order to route these matchings, we may have to incur the congestion of $\Omega(1/e)$. However, this problem is easy to overcome by performing a suitable grouping of the edges of $\Gamma_j$.

The rest of the algorithm proceeds in three steps. In the first step, we perform groupings of the edges in the subsets $\Gamma_j$ for $1 \leq j \leq \gamma$. In the second step, we construct the trees $T_1, \ldots, T_{k'}$. In the third step, we finish the construction of the expander $X$ and its embedding into $G$, and produce the final routing of a subset of demand pairs in $M'$.

**Step 1: Edge grouping.** In this step we compute, for each $1 \leq j \leq \gamma$, a grouping of the edges in $\Gamma_j$. We then establish some properties of these groupings. We use the following two parameters: $p = 8\beta(k)/\alpha_{bw}(k) = O(\log^{4.5} k)$ is the grouping parameter for the sets $\Gamma_j$. The second parameter, $k' = \lfloor \frac{1}{4\gamma} \cdot \frac{k}{3p} \rfloor = \Omega\left(\frac{k}{\log^{16} k \log \log k}\right)$ is the number of the vertices in the expander $X$ that we will eventually construct. We assume without loss of generality that $k'$ is an even integer; otherwise we round it down to the closest even integer.

Fix some $1 \leq j \leq \gamma$. Since $G[S_j] \cup \text{out}_G(S_j)$ is a connected graph, we can find a spanning tree $T_j$ of this graph, and partition the edges of $\Gamma_j$ along this tree into groups whose size is at least $p$ and at most $3p$ using Theorem 2.7. Let $G_j$ be the resulting collection of groups, and let $k^* = \lfloor \frac{k}{3p} \rfloor$. For each group $U \in G_j$, let $T_j(U)$ be the sub-tree of the tree $T_j$ spanning the edges of $U$. For each group $U \in G_j$, we select one arbitrary representative edge, and we let $\Gamma_j'$ denote this set of representative edges. For each $e \in \Gamma_j'$, we denote by $U_e$ the group to which $e$ belongs. Additionally, let $U_e^* \subseteq U_e$ be an arbitrary subset of $p$ edges of $U_e$, including $e$ itself. Notice that $|U_e^*| \geq k^*$ must hold. If $|U_e^*| > k^*$, then we discard edges from $\Gamma_j'$ arbitrarily, until $|U_e^*| = k^*$ holds. This finishes the description of the grouping. The next theorem establishes some properties of the resulting groupings that will be used later.

**Theorem 3.10**

- For each $1 \leq j \leq \gamma$, for any pair $X, Y \subseteq \Gamma_j'$ of edge subsets, where $|X| = |Y|$, there is a collection $\mathcal{P}(X, Y) : X \overset{i_2}{\rightarrow} Y$ of paths contained in $G[S_j]$, where each path connects a distinct edge of $X$ to a distinct edge of $Y$, and the paths cause congestion at most 2.

- For all $1 \leq i \neq j \leq \gamma$, there is a set $\mathcal{P}_{i,j} : \Gamma_i \overset{i_2}{\rightarrow} \Gamma_j'$ of $k^*$ paths in graph $G$. That is, each path connects a distinct edge of $\Gamma_i'$ to a distinct edge of $\Gamma_j'$, with total congestion at most 2.

- Let $\Gamma_1^* \subseteq \Gamma_1'$ be any subset of $k'$ edges, $M' \subseteq M$ any subset of $k'/2$ source-sink pairs, and $T'$ the subset of terminals participating in pairs in $M'$. Then there is a set $\mathcal{P} : T' \overset{i_4}{\rightarrow} \Gamma_1^*$ of paths in $G$, each path connecting a distinct terminal of $T'$ to a distinct edge of $\Gamma_1^*$, with total congestion at most 4.

**Proof:** In order to prove the first assertion, fix some $1 \leq j \leq \gamma$. From the integrality of flow, it is enough to prove that there is a flow $F_j(X, Y)$, where each edge in $X$ sends one flow unit, each edge in $Y$ receives one flow unit, the flow congestion is at most 2, and the flow is contained in $G[S_j]$. We start by defining two subsets $X', Y' \subseteq \Gamma_j$ of edges, as follows: $X' = \bigcup_{e \in X} U_e^*$, and $Y' = \bigcup_{e \in Y} U_e^*$. Observe...
that \(|X'| = |Y'| = |X| \cdot p\). Since set \(S_j\) has the \(\alpha_{bw}(k)\)-bandwidth property for \(\Gamma_j\), there is a flow \(F_j(X', Y')\) contained in \(G[S_j]\), where every edge in \(X'\) sends one flow unit, every edge in \(Y'\) receives one flow unit, and the congestion due to this flow is at most \(1/\alpha_{bw}(k)\). We are now ready to define the flow \(F_j(X, Y)\). Each edge \(e \in X\) spreads one flow unit uniformly among the edges of \(U_e'\) along the tree \(T_j(U_e)\). Next, all this flow is sent along the flow-paths in \(F_j(X', Y')\), where we scale this flow down by factor \(p\). Finally, each edge \(e \in Y\) collects all flow from edges in \(U_e'\) along the tree \(T_j(U_e)\).

Since all trees \(\{T_j(U)\}_{U \in G_j}\) are edge-disjoint, and since the congestion caused by \(F_j(X', Y')\) is at most 1/\(\alpha_{bw}(k)\) < \(p\), the resulting flow \(F_j(X, Y)\) causes congestion at most 2.

We now turn to prove the second assertion. From the integrality of flow, it is enough to prove that there is a flow \(F_{i,j}: \Gamma_i \xrightarrow{1:2} \Gamma'_j\), where every edge \(e \in \Gamma_i\) sends one flow unit and every edge in \(\Gamma'_j\) receives one flow unit. As before, we construct two edge subsets, \(X \subseteq \Gamma_j\) and \(Y \subseteq \Gamma_i\), as follows: \(X = \bigcup_{e \in \Gamma_i} U_e', \) and \(Y = \bigcup_{e \in \Gamma_j} U_e'.\) Notice that \(|X| = |Y| = k^* \cdot p\).

Recall that from the definition of good routers, we already have a flow \(F_j\), where each edge \(e \in \Gamma_j\) sends one flow unit to a distinct terminal in \(T\), with total congestion at most \(\eta = 2\beta(k)/\alpha_{bw}(k)\). We discard all flow-paths except those originating at the edges of \(X\). As a result, we obtain a flow \(F_j\), where each edge \(e \in X\) sends one flow unit to a distinct terminal \(t_e \in T\), and \(F_j\) causes congestion at most \(\eta\) in \(G\). Let \(T_j\) be the subset of terminals that receive flow in \(F_j\), \(|T_j| = |X|\). Similarly, we can define a flow \(F_j\), where each edge \(e \in Y\) sends one flow unit to a distinct terminal \(t_e \in T\), and \(F_j\) causes congestion at most \(\eta\) in \(G\). Subset \(T_j\) of terminals is defined similarly. Notice that \(T_i\) and \(T_j\) are not necessarily disjoint. But since the set \(T\) of terminals is \(1/2\)-well-linked in \(G\), there is a flow \(F : T_i \xrightarrow{1:2} T_j\), where each terminal in \(T_i\) sends one flow unit, each terminal in \(T_j\) receives one flow unit, and the total edge-congestion is at most 2. We concatenate the three flows, \(F_j, F_j, \) to obtain a flow \(F' : X \xrightarrow{1:2} Y\). In this flow, each edge in \(X\) sends one flow unit, each edge in \(Y\) receives one flow unit, and the total congestion is at most \(2\eta + 2\).

We are now ready to define the flow \(F_{i,j}\). Each edge \(e \in \Gamma'_j\) sends one flow unit along the tree \(T_i(U_e)\), which is evenly split among the edges of \(U_e'\). We then use the flow \(F'\), scaled down by factor \(p\), to route this flow to the edges of \(Y\). Finally, each edge \(e \in \Gamma'_i\) collects the flow that the edges of \(U_e'\) receive, along the tree \(T_j(U_e)\), so that after collecting all that flow, edge \(e\) receives 1 flow unit. In order to analyze the total congestion due to flow \(F_{i,j}\), observe that all trees \(\{T_i(U)\}_{U \in G_i} \cup \{T_j(U)\}_{U \in G_j}\) are edge-disjoint. So the routing along these trees causes a congestion of at most 1. Since flow \(F_{i,j}\) causes congestion of at most \(2\eta + 2\), and \(p\) is selected so that \(p \geq 2\eta + 2\), the congestion due to the scaled-down flow \(F'\) is at most 1. The total congestion is therefore at most 2.

Finally, we prove the third assertion. Let \(\Gamma_i' \subseteq \Gamma_i\) be any subset of \(k'\) edges, \(\mathcal{M}' \subseteq \mathcal{M}\) any subset of \(k'/2\) source-sink pairs, and \(\mathcal{T}'\) the set of all terminals participating in the pairs in \(\mathcal{M}'\). Let \(X = \bigcup_{e \in \Gamma_i'} U_e'\), so \(|X| = k'p\). As before, we make use of the previously defined flow \(F_1\), where each edge \(e \in \Gamma_1\) sends one flow unit to a distinct terminal in \(T\), with total congestion at most \(\eta = 2\beta(k)/\alpha_{bw}(k)\). We discard all flow-paths except those that originate at the edges of \(X\). As a result, we obtain a flow \(F\), where each edge \(e \in X\) sends one flow unit to a distinct terminal \(t_e \in T\), and \(F\) causes congestion at most \(\eta < p\) in \(G\). We now define a new flow \(F' : \Gamma_i' \xrightarrow{1:2} \mathcal{T}\), where each edge in \(\Gamma_i'\) sends one flow unit, and each terminal in \(\mathcal{T}\) receives at most one flow unit. Flow \(F'\) is defined as follows. Each edge \(e \in \Gamma_i'\) sends one flow unit to the edges in \(U_e'\) along the tree \(T_i(U_e)\), distributing it evenly among these edges. Each edge in \(U_e'\) then sends the \(1/p\) flow unit it receives from \(e\) to the terminals via the flow \(F\), so the flow \(F'\) is scaled down by factor \(p\). Since the congestion caused by flow \(F'\) is \(\eta < p\), and the trees \(\{T_i(U_e)\}_{e \in \Gamma_i'}\) are edge-disjoint, the total congestion caused by \(F'\) is at most 2. Moreover, each terminal receives at most one flow unit in \(F'\). From the integrality of flow, there is a subset \(\mathcal{T}'' \subseteq \mathcal{T}\) of \(k'\) terminals, and a collection \(\mathcal{P}_1 : \Gamma_i' \xrightarrow{1:2} \mathcal{T}''\) of paths in \(G\). Since the set \(\mathcal{T}\) of terminals is \(1/2\)-well-linked, using the integrality of flow, there is a collection \(\mathcal{P}_2 : \mathcal{T}'' \xrightarrow{1:2} \mathcal{T}'\) of paths in \(G\). We then
obtain the desired collection \( P \) of paths by concatenating the paths in \( P_1 \) with the paths in \( P_2 \).

\[
\text{Step 2: Constructing the Trees.} \quad \text{The goal of this step is to find a collection } T_1, \ldots, T_{k'} \text{ of trees in graph } G, \text{ such that each edge of } G \text{ belongs to at most 8 trees. For each tree } T_i, \text{ we will find a subset } E_i \subseteq E(T_i) \text{ of special edges, such that the sets } E_1, \ldots, E_{k'} \text{ are pairwise disjoint, and for each } 1 \leq i \leq k', \text{ } E_i = \{e_{i,1}, e_{i,2}, \ldots, e_{i,\gamma}\}, \text{ where for } 1 \leq j \leq \gamma, e_{i,j} \in \Gamma_j. \text{ Notice that an edge } e \in \Gamma_j \text{ may belong to several trees, but only to one of them as a special edge. For each } 1 \leq j \leq \gamma, \text{ we denote } \Gamma_j = \{e_{1,j}, \ldots, e_{k',j}\}, \text{ the subset of edges of } \Gamma_j \text{ that the trees } T_1, \ldots, T_{k'} \text{ contain as special edges. We summarize Step 2 in the next theorem.}

\begin{align*}
\text{Theorem 3.11} & \quad \text{Given a family } F \text{ of good routers, and a subset } \Gamma_j' \subseteq \text{out}_G(S_j) \text{ of edges for each } 1 \leq j \leq \gamma, \text{ as computed in Step 1, we can efficiently find } k' \text{ trees } T_1, \ldots, T_{k'} \text{ in graph } G, \text{ and for each tree } T_i \text{ a subset } E_i \subseteq E(T_i) \text{ of special edges, such that:} \\
& \quad \text{• Each edge of } G \text{ belongs to at most 8 trees;} \\
& \quad \text{• Subsets } E_1, \ldots, E_{k'} \text{ of edges are pairwise disjoint; and} \\
& \quad \text{• For all } 1 \leq i \leq k', \text{ } E_i = \{e_{i,1}, \ldots, e_{i,\gamma}\}, \text{ where for all } 1 \leq j \leq \gamma, e_{i,j} \in \Gamma_j'.
\end{align*}

\text{Proof:} \quad \text{In order to prove the theorem, we start by augmenting the graph } G \text{ as follows. First, replace each edge of } G \text{ with two parallel edges. Next, for each } 1 \leq j \leq \gamma, \text{ add a new vertex } s_j, \text{ and for each edge } e \in \Gamma_j, \text{ we sub-divide one of the copies of } e, \text{ by adding a new vertex } v_e, \text{ which is then connected to the vertex } s_j. \text{ Notice that from Theorem 3.10, for each } 1 \leq j \neq j' \leq \gamma, \text{ there are exactly } k^* \text{ edge-disjoint paths connecting } s_j \text{ to } s_{j'} \text{ in the resulting graph. Finally, we replace each edge in the resulting graph by two bi-directed edges, thus obtaining a directed Eulerian graph that we denote by } G^+. \text{ From Theorem 3.10, for each pair } 1 \leq j \neq j' \leq \gamma \text{ of indices, there are } k^* \text{ edge-disjoint paths connecting } s_j \text{ to } s_{j'}, \text{ and } k^* \text{ edge-disjoint paths connecting } s_{j'} \text{ to } s_j. \text{ Notice also that each vertex } s_j \text{ has exactly } k^* \text{ incoming edges and exactly } k^* \text{ outgoing edges.}

\text{As a next step, we use the standard edge splitting procedure in graph } G^+. \text{ Our goal is to eventually obtain a graph } \tilde{H} \text{ over the set } \{s_1, \ldots, s_\gamma\} \text{ of vertices, such that each pair } s_j, s_{j'} \text{ is } k^*-\text{edge connected, and each edge } e = (s_j, s_{j'}) \in E(\tilde{H}) \text{ is associated with a path } P_e \text{ connecting } s_j \text{ to } s_{j'} \text{ in } G^+, \text{ while all paths in } \{P_e \mid e \in E(\tilde{H})\} \text{ are edge-disjoint in } G^+.

\text{Let } D = (V, A) \text{ be any directed multigraph with no self-loops. For any pair } (v, v') \in V \text{ of vertices, their connectivity } \lambda(v, v'; D) \text{ is the maximum number of edge-disjoint paths connecting } v \text{ to } v' \text{ in } D. \text{ Given a pair } a = (u, v), b = (v, w) \text{ of edges, a splitting-off procedure replaces the two edges } a, b \text{ by a single edge } (u, w). \text{ We denote by } D^{a,b} \text{ the resulting graph. We use the extension of Mader’s theorem [Mad78] to directed graphs, due to Frank [Fra89] and Jackson [Jac98]. Following is a simplified version of Theorem 3 from [Jac98]:}

\begin{align*}
\text{Theorem 3.12} & \quad \text{Let } D = (V, A) \text{ be an Eulerian digraph, } v \in V \text{ and } a = (v, u) \in A. \text{ Then there is an edge } b = (w, v) \in A, \text{ such that for all } y, y' \in V \setminus \{v\}: \lambda(y, y'; D) = \lambda(y, y'; D^{a,b})
\end{align*}

\text{We apply Theorem 3.12 repeatedly to all vertices of } G^+ \text{ except for the vertices in set } \{s_1, \ldots, s_\gamma\}, \text{ until we obtain a directed graph } \tilde{H}, \text{ whose vertex set is } \{s_1, \ldots, s_\gamma\}, \text{ and for each } 1 \leq j, j' \leq \gamma, \text{ there are } k^* \text{ edge-disjoint paths connecting } s_j \text{ to } s_{j'} \text{ and } k^* \text{ edge-disjoint paths connecting } s_{j'} \text{ to } s_j. \text{ Clearly, each edge } e = (s_j, s_{j'}) \in E(\tilde{H}) \text{ is associated with a path } P_e \text{ connecting } s_j \text{ to } s_{j'} \text{ in } G^+, \text{ and all paths } \{P_e \mid e \in E(\tilde{H})\} \text{ are edge-disjoint. Let } \tilde{H}' \text{ denote the undirected multi-graph identical to } \tilde{H}, \text{ except}
that now all edges become undirected. Notice that each vertex \( s_j \) must have \( 2k^* \geq \gamma \) edges adjacent to it in \( \tilde{H}' \), so the graph contains many parallel edges. For each pair \( s_j, s_{j'} \) of vertices, there are exactly \( 2k^* \) edge-disjoint paths connecting \( s_j \) to \( s_{j'} \) in \( \tilde{H}' \). For convenience, let us denote \( 2k^* \) by \( \ell \).

As a next step, we build an auxiliary undirected graph \( Z \) on the set \( \{s_1, \ldots, s_*\} \) of vertices, as follows. For each pair \( s_j, s_{j'} \) of vertices, there is an edge \((s_j, s_{j'})\) in graph \( Z \) if there are at least \( \ell/\gamma^3 \) edges connecting \( s_j \) and \( s_{j'} \) in \( \tilde{H}' \). If edge \( e = (s_j, s_{j'}) \) is present in graph \( Z \), then its capacity \( c(e) \) is set to be the number of edges connecting \( s_j \) to \( s_{j'} \) in \( \tilde{H}' \). For each vertex \( s_j \), let \( C(s_j) \) denote the total capacity of edges incident on \( s_j \) in graph \( Z \). We need the following simple observation.

**Observation 3.13**

- For each vertex \( v \in V(Z) \), \( (1-\gamma^2)\ell \leq C(v) \leq \ell \).
- For each pair \((u, v)\) of vertices in graph \( Z \), we can send at least \((1-\gamma)\ell \) flow units from \( u \) to \( v \) in \( Z \) without violating the edge capacities.

**Proof:** In order to prove the first assertion, recall that each vertex in graph \( \tilde{H}' \) has \( \ell \) edges incident to it (this is since, in graph \( G' \), each vertex \( s_1, \ldots, s_* \) had exactly \( k^* \) incoming and \( k^* \) outgoing edges, and we did not perform edge splitting on these vertices). So \( C(v) \leq \ell \) for all \( v \in V(Z) \).

Call a pair \((s_j, s_{j'})\) of vertices bad iff there are fewer than \( \ell/\gamma^3 \) edges connecting \( s_j \) to \( s_{j'} \) in \( \tilde{H}' \). Notice that each vertex \( v \in V(Z) \) may participate in at most \( \gamma \) bad pairs, as \(|V(Z)| = \gamma \). Therefore, \( C(v) \geq \ell - \gamma\ell/\gamma^3 = \ell(1 - 1/\gamma^2) \) must hold.

For the second assertion, assume for contradiction that it is not true, and let \((u, v)\) be a violating pair of vertices. Then there is a cut \((A, B)\) in \( Z \), with \( u \in A \), \( v \in B \), and the total capacity of edges crossing this cut is at most \((1-\gamma)\ell \). Since \( u \) and \( v \) were connected by \( \ell \) edge-disjoint paths in graph \( \tilde{H}' \), this means that there are at least \( \ell/\gamma \) edges in graph \( \tilde{H}' \) that connect bad pairs of vertices. But since we can only have at most \( \gamma^2 \) bad pairs, and each pair has less than \( \ell/\gamma^3 \) edges connecting them, this is impossible.

We now proceed in two steps. First, we show that we can efficiently find a spanning tree of \( Z \) with maximum vertex degree at most 3. Next, using this spanning tree, we show how to construct the collection \( T_1, \ldots, T_{k^*} \) of trees.

**Claim 3.14** There is an efficient algorithm to find a spanning tree \( T^* \) of \( Z \) with maximum vertex degree at most 3.

**Proof:** We use the algorithm of Singh and Lau [SL15] for constructing bounded-degree spanning trees. Suppose we are given a graph \( G = (V, E) \), and our goal is to construct a spanning tree \( T \) of \( G \), where the degree of every vertex is bounded by \( B \). For each subset \( S \subseteq V \) of vertices, let \( E(S) \) denote the subset of edges with both endpoints in \( S \), and \( \delta(S) \) the subset of edges with exactly one endpoint in \( S \). Singh and Lau consider a natural LP-relaxation for the problem. We note that their algorithm works for a more general problem where edges are associated with costs, and the goal is to find a minimum-cost tree that respects the degree requirements; since we do not need to minimize the tree cost, we only discuss the unweighted version here. For each edge \( e \in E \), we have a variable \( x_e \) indicating whether \( e \) is included in the solution. We are looking for a feasible solution to the following LP.

\[
\begin{align*}
\text{minimize} & \quad \sum_{e \in E} c(e) x_e \\
\text{subject to} & \quad \sum_{e \in \delta(S)} x_e \leq B & \forall S \subseteq V \\
\end{align*}
\]
Recall that $|V(Z)| = \gamma$. The solution is defined as follows. Let $e = (u, v)$ be any edge in $E(Z)$. We set the LP-value of $e$ to be $x_e = \frac{\gamma - 1}{\gamma} \cdot \left( \frac{c(e)}{C(v)} + \frac{c(e)}{C(u)} \right)$. We say that $\frac{\gamma - 1}{\gamma} \cdot \frac{c(e)}{C(u)}$ is the contribution of $v$ to $x_e$, and $\frac{\gamma - 1}{\gamma} \cdot \frac{c(e)}{C(v)}$ is the contribution of $u$. We now verify that all constraints of the LP hold.

First, it is easy to see that $\sum_{e \in E} x_e = |V| - 1$, as required. Indeed,

$$\sum_{e \in E} x_e = \sum_{e = (u, v) \in E} \frac{\gamma - 1}{\gamma} \cdot \left( \frac{c(e)}{C(v)} + \frac{c(e)}{C(u)} \right)$$

$$= \sum_{v \in V} \frac{\gamma - 1}{\gamma} \cdot \left( \sum_{e \in \delta(v)} \frac{c(e)}{C(v)} \right)$$

$$= \frac{\gamma - 1}{\gamma} \cdot |V|$$

$$= |V| - 1,$$

as $|V| = \gamma$.

Next, consider some subset $S \subset V$ of vertices. Notice that it is enough to establish Constraint (3) for subsets $S$ with $|S| \geq 2$. From Observation 3.13, the total capacity of edges in $E_Z(S, \bar{S})$ must be at least $(1 - 1/\gamma)\ell$. Since for each $v \in S$, $C(v) \leq \ell$, the total contribution of the vertices in $S$ towards the LP-weights of edges in $E_Z(S, \bar{S})$ is at least $\frac{\gamma - 1}{\gamma} \cdot (1 - 1/\gamma) = (1 - 1/\gamma)^2$. Therefore,

$$\sum_{e \in E(S)} x_e \leq \frac{\gamma - 1}{\gamma} |S| - (1 - 1/\gamma)^2 = |S| - |S|/\gamma - 1 - 1/\gamma^2 + 2/\gamma \leq |S| - 1$$

since we assume that $|S| \geq 2$. This establishes Constraint (3). Finally, we show that for each $v \in V(Z)$, $\sum_{e \in \delta(v)} x_e \leq 2$. First, the contribution of the vertex $v$ to this summation is bounded by 1. Next, recall that for each $u \in V(Z)$, $C(u) \geq (1 - 1/\gamma^2)\ell$, while the total capacity of edges in $\delta(v)$ is at most $\ell$. Therefore, the total contribution of other vertices to this summation is bounded by $\frac{\ell}{(1 - 1/\gamma^3)\ell} \cdot \frac{\gamma - 1}{\gamma} \leq \frac{\gamma - 1}{\gamma} \leq 1$. The algorithm of Singh and Lau can now be used to obtain a spanning tree $T^*$ for $Z$ with maximum vertex degree at most 3.

Root the tree $T^*$ at any degree-1 vertex $r$. Let $e = (s_i, s_j)$ be some edge of the tree, where $s_i$ is the parent of $s_j$. Recall that there are at least $\ell/\gamma^3$ edges $(s_i, s_j)$ in graph $H'$. Let $A(e)$ be any collection of exactly $\lfloor \ell/\gamma^3 \rfloor$ such edges. Recall that for each edge $\hat{e} \in A(e)$ in graph $H'$, there is a path $P$, connecting either $s_i$ to $s_j$ or $s_j$ to $s_i$ in graph $G^+$ (recall that graph $G^+$ is directed). Since the direction of the edges in $G^+$ will not play any role in the following argument, we will assume without
loss of generality that $P$ is directed from $s_j$ towards $s_i$. Recall that the first edge on path $P$ must connect $s_j$ to some vertex $v_\ell$, where $\ell \in \Gamma'_j$, and similarly, the last edge on path $P$ connects some vertex $v_\ell_\gamma$, for $\ell_\gamma \in \Gamma'_\gamma$ to $s_i$. So by removing the first and the last edges from path $P$, we obtain a path $P_\hat{e}$ in graph $G$, that connects edge $\hat{e} \in \Gamma'_j$ to edge $\hat{e}' \in \Gamma'_\gamma$. Since $s_i$ is the parent of $s_j$ in tree $T^*$, we will think of $P_\hat{e}$ as being directed from $S_j$ towards $S_i$. We call $\hat{e}$ the first edge of $P_\hat{e}$, and $\hat{e}'$ the last edge of $P_\hat{e}$. Going back to the edge $e = (s_i, s_j)$ in tree $T^*$, we can now define a set $\mathcal{P}(e) = \{ P_\hat{e} \mid \hat{e} \in A(e) \}$ of exactly $\lceil \ell/\gamma^2 \rceil$ paths in graph $G$, associated with $e$. We let

$$B_1(e) = \{ \hat{e} \in \Gamma'_j \mid \hat{e} \text{ is the first edge on some path } P_\hat{e} \in \mathcal{P}(e) \}$$

and

$$B_2(e) = \{ \hat{e} \in \Gamma'_\gamma \mid \hat{e} \text{ is the last edge on some path } P_\hat{e} \in \mathcal{P}(e) \}$$

Both sets $B_1(e)$, $B_2(e)$ are multi-sets, that is, if some edge $\hat{e} \in \Gamma'_j$ appears as a first edge on two paths in $\mathcal{P}(e)$, then we add two copies of $\hat{e}$ to $B_1(e)$. (From the construction of $G^+$, it is easy to see that $\hat{e}$ may appear as the first edge on at most two such paths). We then have that $\mathcal{P}(e) : B_1(e) \overset{1:1}{\cup} B_2(e)$ in graph $G$, since, from the construction of graphs $G^+$ and $\hat{H}'$, every edge of graph $G$ may appear on at most four paths of $\bigcup_{e \in E(T^*)} \mathcal{P}(e)$.

We call the sets $B_1(e)$, $B_2(e)$ of edges bundles corresponding to $e$, and we view $B_1(e)$ as a bundle that belongs to $S_j$, while $B_2(e)$ is a bundle that belongs to $S_i$. Since the degree of tree $T^*$ is at most 3, every set $S_j$ has at most three bundles that belong to it. From the construction of graph $G^+$, for every vertex $s_i : 1 \leq i \leq \gamma$, each edge in $\Gamma'_i$ may appear at most twice in the multi-set defined by the union of the bundles that belong to $S_i$. In particular, it is possible that it appears twice in the same bundle. We need to make sure that this never happens. In order to achieve this, we will define, for each edge $e \in E(T^*)$, smaller bundles, $B'_1(e) \subseteq B_1(e)$ and $B'_2(e) \subseteq B_2(e)$, such that each edge appears at most once in each bundle, and there is a subset $\mathcal{P}'(e) \subseteq \mathcal{P}(e)$, where $\mathcal{P}'(e) : B'_1(e) \overset{1:1}{\cup} B'_2(e)$. We will also ensure that $|B'_1(e)| = |B'_2(e)| = \lfloor \ell/4\gamma^3 \rfloor$ (see Figure 2(b)).

This is done as follows. Consider some edge $e = (s_i, s_j)$ in tree $T^*$, and assume that $s_i$ is the parent of $s_j$ in the tree. Consider first $B_1(e)$. For each edge $\hat{e} \in B_1(e)$, if two copies of $\hat{e}$ appear in $B_1(e)$, then we remove one of the copies from $B_1(e)$. If $P \in \mathcal{P}(e)$ is one of the two paths for which $\hat{e}$ is the first edge, then we remove $P$ from $\mathcal{P}(e)$, and we also remove its last edge from $B_2(e)$. It is easy to see that we remove at most half the edges of $B_1(e)$. We then perform the same operation for $B_2(e)$. In the end, both $B_1(e)$ and $B_2(e)$ must contain at least a 1/4 of the original edges, and $\mathcal{P}(e)$ contains at least a 1/4 of the original paths. We now let $\mathcal{P}'(e)$ be any subset of exactly $\lfloor \ell/4\gamma^3 \rfloor$ remaining paths, and we set $B'_1(e)$ to be the set of all edges $\hat{e}$ that appear as the first edge on some path in $\mathcal{P}'(e)$, and similarly $B'_2(e)$ the set of all edges that appear as the last edge on some path in $\mathcal{P}'(e)$. We perform this operation for all edges $e$ of tree $T^*$. We then obtain the following corollary.

**Corollary 3.15** Our algorithm finds a tree $T^*$ with maximum vertex degree at most 3, rooted at a degree-1 vertex, with vertex set $V(T^*) = \{ v_1, \ldots, v_\gamma \}$. Additionally, for each edge $e = (v_i, v_j) \in E(T^*)$, where $v_i$ is a parent of $v_j$ in $T^*$, the algorithm finds a collection $\mathcal{P}'(e)$ of paths in graph $G$, a subset $B'_1(e) \subseteq \Gamma(S_j)$ of $\lfloor \ell/4\gamma^3 \rfloor$ edges, and a subset $B'_2(e) \subseteq \Gamma(S_i)$ of $\lfloor \ell/4\gamma^3 \rfloor$ edges, such that:

- Each path in $\mathcal{P}'(e)$ connects a distinct edge of $B'_1(e)$ to a distinct edge of $B'_2(e)$ in $G$; and
- The paths in $\bigcup_{e \in E(T^*)} \mathcal{P}'(e)$ cause total edge-congestion at most 4 in $G$.


We are now ready to define the subsets $\Gamma^*_j \subseteq \Gamma'_j$ of $k'$ edges, $\Gamma^*_j = \{e_{1,j}, \ldots, e_{k',j}\}$, that our trees will span. Fix some index $1 \leq j \leq \gamma$. If $s_j$ is not the root of the tree $T^*$, then we let $\Gamma^*_j = B_1(e)$, where $e$ is the edge connecting $s_j$ to its parent in $T^*$. If $s_j$ is the root of the tree, then $\Gamma^*_j = B_2(e)$, where $e$ is the unique edge incident on $s_j$ in tree $T^*$. Notice that $|\Gamma^*_j| = \lceil \frac{k}{1+\gamma} \rceil = \lceil \frac{k'}{2\gamma} \rceil = k'$.

Finally, we construct the trees $T_1, \ldots, T_{k'}$. In order to construct these trees, for each vertex $s_j \in V(T^*)$, we define a set $Q_j$ of paths, and a corresponding set $U_j \subseteq S_j$ of special vertices in graph $G$, as follows. If the degree of $s_j$ is 1 in $T^*$, then let $e$ be the unique edge of $T^*$ incident on $s_j$. We let $U_j$ be the set of vertices of $S_j$ that serve as endpoints of the paths in $P'(e)$, and we set $Q_j = \emptyset$. If the degree of $s_j$ is 2 in $T^*$, then let $e_1$ and $e_2$ be the two edges of $T^*$ incident on $s_j$, and assume that $e_1$ is the edge connecting $s_j$ to its parent. Let $U_1$ be the set of vertices of $S_j$ that serve as endpoints of the paths in $P'(e_1)$, and let $U_2$ be defined similarly for $e_2$. We then define $U_j = U_1 \cup U_2$. Notice that from Theorem 3.10, we can find a set $Q_j : U^1 \uplus U^2$ of paths contained in $G[S_j]$, where each path in connects a distinct vertex of $U^1$ to a distinct vertex of $U^2$. Finally, assume that $s_j$ has degree 3 in $T^*$. Let $e_1$ be the edge incident on $s_j$ that connects it to its parent, and let $e_2, e_3$ be the other two edges incident on $s_j$ in $T^*$. Let $U^1$ and $U^2$ be defined exactly as before, with respect to $e_1$ and $e_2$. Similarly, we let $U^3$ be the set of vertices of $S_j$ that serve as endpoints of the paths in $P'(e_3)$. We let $U_j = U^1 \cup U^2 \cup U^3$. As before, from Theorem 3.10, we can find a set $Q^1 : U^1 \uplus U^2$ and a set $Q^2 : U^1 \uplus U^3$ of paths in $G[S_j]$. We let $Q_j = Q^1 \uplus Q^2$ (see Figure 3(a)). Let $P = \bigcup_{e \in E(T^*)} P'(e)$, and $Q = \bigcup_{v_j \in V(T^*)} Q_j$. Recall that every edge of $G$ belongs to at most 4 paths in $P$, and it is easy to see that it belongs to at most 4 paths in $Q$.

It is now easy to complete the construction of the trees. Let $U$ be the graph obtained from disjoint union of the paths in $P$ and $Q$ - that is, if a vertex, or an edge, belong to several paths in $P \cup Q$, then we create several copies of this edge or vertex. For each $s_j \in V(T^*)$, for each vertex $v \in U_j$, we unify all copies of $v$, that serve as endpoints of the paths in $P \cup Q$ (there can be at most 3 such copies), and we call the resulting copy of $v$ a special copy. (Notice that $v$ may belong to additional paths in $P \cup Q$ as an inner vertex, and we do not unify such copies of $v$.) It is easy to see that the resulting graph is a collection of $k'$ disjoint trees $T_1, \ldots, T_{k'}$, where for each edge $e \in E(T^*)$, each path $P \in P'(e)$ is contained in a distinct tree $T_i$ (see Figure 3(b)). Consider now some vertex $v_j \in V(T^*)$, and the corresponding set $\Gamma^*_j$ of $k'$ edges. If $s_j$ is not the root of $T^*$, then let $e$ be the edge of $T^*$ connecting
s_j \text{ to its parent; otherwise, let } e \text{ be the unique edge of } T^* \text{ incident on } s_j. \text{ For each edge } \tilde{e} \in \Gamma_j^*, \text{ the unique path } P \in \mathcal{P}'(e) \text{ that originates from } \tilde{e} \text{ is contained in some tree } T_i, \text{ for } 1 \leq i \leq k', \text{ and we add } \tilde{e} \text{ to the set } E_i \text{ of special edges for } T_i. \text{ It is easy to see that each edge in } \Gamma_j^* \text{ is added as a special edge to a distinct tree in } \{T_1, \ldots, T_{k'}\}. \text{ As observed before, } \text{every edge of } G \text{ may belong to at most 8 trees } T_i, \text{ for } 1 \leq i \leq k'. \text{ Notice that for } 1 \leq i \leq k', \text{ tree } T_i \text{ defines a connected sub-graph of } G, \text{ that may not necessarily be a tree (since the same edge or vertex may appear twice in } T_i \text{ in graph } U). \text{ We simply delete edges from the resulting sub-graph of } G, \text{ until it becomes a tree, and we still denote that tree by } T_i. \text{ } \square

\textbf{Step 3: Constructing the Expander and Finding the Routing.} \text{ In this step, we construct the expander } X, \text{ together with its embedding into the graph } G, \text{ and find the final routing of a subset of demands in } \mathcal{M}. \text{ Let } \mathcal{M}' \subseteq \mathcal{M} \text{ be any subset of } k'/2 \text{ demand pairs, and let } \mathcal{T}' \text{ be the subset of terminals participating in the pairs of } \mathcal{M}'.

Let } \mathcal{P}^* = \mathcal{T}' \xleftarrow{1,1} \Gamma_1^* \text{ be the collection of paths connecting the terminals of } \mathcal{T}' \text{ to the edges of } \Gamma_1^* \subseteq \Gamma_1 \text{ (where } \Gamma_1^* = \{e_{1,1}, \ldots, e_{k',1}\}), \text{ guaranteed by Theorem 3.10. Denote } \mathcal{P}^* = \{P_i \mid t \in \mathcal{T}'\}, \text{ where } P_i \text{ is the path originating from terminal } t. \text{ Rename the terminals in } \mathcal{T}' \text{ as } \mathcal{T}' = \{t_1, \ldots, t_{k'}\}, \text{ where for each } 1 \leq i \leq k', t_i \text{ is the terminal whose path } P_i \text{ terminates at the edge } e_{i,1} \text{ (the unique edge of } \Gamma_1^* \text{ that belongs to the tree } T_i \text{ as a special edge). For } 1 \leq i \leq k', \text{ let } C_i \text{ be the connected subgraph of graph } G, \text{ that consists of the union of the tree } T_i \text{ and the path } P_i. \text{ Since each edge of graph } G \text{ participates in at most 8 trees } T_i, \text{ and at most 4 paths in } \mathcal{P}^*, \text{ each edge of } G \text{ participates in at most 12 connected sub-graphs } C_i.

\text{We now construct the expander } X \text{ and embed it into the graph } G. \text{ The set of vertices of } X \text{ is } V(X) = \{v_1, \ldots, v_{k'}\}, \text{ where we view each vertex } v_i \text{ as representing the terminal } t_i \in \mathcal{T}'. \text{ We view the connected sub-graph } C_i \text{ of } G \text{ as the embedding of the vertex } v_i \text{ into } G. \text{ Finally, we need to define the set of the edges of } X \text{ and specify their embedding into } G. \text{ In order to do so, we use the cut-matching game from Theorem 2.2. Recall that in each iteration } j, \text{ the cut player produces a partition } (A_j, B_j) \text{ of } V(X), \text{ with } |A_j| = |B_j|. \text{ The matching player then returns some matching } M_j \text{ between the vertices of } A_j \text{ and } B_j, \text{ and the edges of } M_j \text{ are added to graph } X. \text{ We use the graphs } G[S_j] \text{ to route the matchings } M_j. \text{ Specifically, let } (A_1, B_1) \text{ be the partition of } V(X) \text{ produced by the cut player in the first iteration. Consider the set } \Gamma_1^* = \{e_{1,1}, \ldots, e_{k',1}\} \text{ of edges. Partition } (A_1, B_1) \text{ of } V(X) \text{ defines a partition } (A'_1, B'_1) \text{ of these edges, where } A'_1 = \{e_{i,1} \mid v_i \in A_1\} \text{ and } B'_1 = \{e_{i,1} \mid v_i \in B_1\}. \text{ From Theorem 3.10, we can find a set } \mathcal{R}_1 : A'_1 \xleftarrow{1} B'_1 \text{ of } |A'_1| \text{ paths contained in } G[S_1], \text{ where each path in } \mathcal{R}_1 \text{ connects a distinct}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{tree_construction.png}
\caption{Constructing the trees}
\end{figure}
edge of $A'_1$ to a distinct edge of $B'_1$. Set $R_1$ of paths then defines a matching $M'_1$ between the sets $A'_1$ and $B'_1$, which in turn defines a matching $M_j$ between the sets $A_j$ and $B_j$ of vertices of $V(X)$. We then treat $M_j$ as the response of the matching player. For each edge $e = (v_i, v_v) \in M_1$ of the matching, we let $P_e$ be the unique path of $R_1$ connecting $v_i$ to $v_v$. We view $P_e$ as the embedding of $e$ into graph $G$. We continue similarly to execute the remaining iterations, where in each iteration $j : 1 \leq j \leq \gamma$, we use the set $S_j \in \mathcal{F}$ to find the matching $M_j$. That is, we define the partition $(A'_j, B'_j)$ of $\Gamma'_j$ based on the partition $(A_j, B_j)$ of $V(X)$ as before, find a collection $R_j : A'_j \rightarrow B'_j$ of paths contained in $G[S_j]$. These paths give us the matching $M_j'$ between the sets $A'_j$ and $B'_j$ of edges, which in turn gives us the matching $M_j$ between the sets $A_j$ and $B_j$ of vertices of $V(X)$. For each edge $e = (v_i, v_v) \in M_j$, we let $P_e$ be the unique path of $R_j$ connecting $e_{i,j}$ to $e_{v,j}$. We view $P_e$ as the embedding of $e$ into graph $G$. The final graph $X$ is the graph obtained after $\gamma$ iterations, with $E(X) = \bigcup_{j=1}^\gamma M_j$, and we are guaranteed that with constant probability it is an $\Omega(\log k')$-expander. For each edge $e = (v_i, v_v) \in E(X)$, we have defined an embedding $P_e$ of $e$ into $G$, where $P_e$ is a path connecting some vertex in $C_i$ to some vertex in $C_v$. Let $P_X = \{P_e \mid e \in E(X)\}$. Then $P_X = \bigcup_{j=1}^\gamma R_j$, and the total congestion caused by paths in $P_X$ in $G$ is at most 2. This finishes the definition of the expander $X$ and of its embedding into $G$.

We now use the expander $X$ and its embedding into $G$, to route a subset of demand pairs. We identify from now on the vertices of $X$ with the terminals of $\mathcal{T}$ they represent, that is, $V(X) = \mathcal{T}$.

We use Theorem 2.1 to find a collection $\hat{P}$ of $r = \Omega \left( \frac{k}{\gamma} \right)$ vertex-disjoint paths in the expander $X$, routing $r$ distinct demand pairs. Let $\mathcal{M}'' \subseteq \mathcal{M}'$ be the set of these demand pairs, and assume, by renumbering the terminals as necessary, that $\mathcal{M}'' = \{(t_1, t_2), (t_3, t_4), \ldots, (t_{2r-1}, t_{2r})\}$. For each $1 \leq i \leq r$, let $P_i \in \hat{P}$ be the path connecting $t_{2i-1}$ to $t_{2i}$. In order to complete the routing, we transform each such path $P_i$ into a path $Q_i$ in graph $G$, connecting the same pair $(t_{2i-1}, t_{2i})$ of terminals.

Fix some $1 \leq i \leq r$. We now show how to transform the path $P_i$ connecting $t_{2i-1}$ to $t_{2i}$ in graph $X$ to a path $Q_i$, connecting the same pair of terminals in graph $G$. In order to do so, we will replace the edges and the vertices of path $P_i$ by paths in graph $G$. We will think of $P_i$ as directed from $t_{2i-1}$ to $t_{2i}$. First, each edge $e = (t_a, t_b) \in P_i$ is replaced by the path $P_e \subseteq G$, connecting some vertex $v \in C_a$ to some vertex $u \in C_b$ (where $P_e$ is the path into which $e$ is embedded). Next, consider some inner vertex $t_x \in P_i$, and let $e, e'$ be the two edges appearing immediately before and immediately after $t_x$ on the original path $P_i$, respectively. Let $v_x \in C_a$ be the last vertex on path $P_e$, and let $v'_x \in C_b$ be the first vertex on path $P_{e'}$. Then we replace the vertex $t_x$ with an arbitrary path $P_x$ connecting $v_x$ to $v'_x$ in the connected sub-graph $C_x$ of $G$. It now only remains to take care of the endpoints of path $P_i$. Let $e$ be the first edge on the original path $P_i$, and recall that the first vertex on $P_i$ is $t_{2i-1}$. Let $v_{2i-1} \in C_{2i-1}$ be the first vertex on the path $P_e$. Then we replace $t_{2i-1}$ by any path connecting $t_{2i-1}$ to $v_{2i-1}$ in the connected sub-graph $C_{2i-1}$ of $G$. The last vertex of $P_i$ is taken care of similarly. Let $Q_i$ denote the resulting path. Notice that $Q_i$ consists of two types of segments: the first type are the paths $P_e$ for edges $e \in P_i$, and the second type is the paths $P_x$ for vertices $x \in P_i$. Let $Q_1, \ldots, Q_r$ be the resulting set of paths. We now bound the congestion due to paths in $Q_1, \ldots, Q_r$ in graph $G$. Recall that the paths $\{P_i\}_{i=1}^r$ are edge- and vertex-disjoint. Recall also that each edge of graph $G$ participates in at most 2 paths of the set $P_X = \{P_e \mid e \in E(X)\}$. Therefore, the congestion due to type-1 segments in $\{Q_i\}_{i=1}^r$ is at most 2. Since the paths in $\{P_i\}_{i=1}^r$ are vertex-disjoint, and every edge of graph $G$ participates in at most 12 components $C_1, \ldots, C_{k'}$, the congestion due to type-2 segments is bounded by 12. Overall, the paths in $\{Q_i\}_{i=1}^r$ cause congestion at most 14. The number of demand pairs routed is $r = \Omega \left( \frac{k}{\gamma} \right) = \Omega \left( \frac{k}{\log k \cdot k \log \log k} \right)$.

To conclude, we have started with a graph $G$, a collection $\mathcal{M}$ of $k$ source-sink pairs, and the set $\mathcal{T}$
of terminals participating in pairs in $M$, such that $G$ is $\frac{1}{2}$-well-linked for $T$. We have constructed a routing for the subset $M'' \subseteq M$ of $\Omega \left( \frac{k}{\log^{\omega} k \log \log k} \right)$ pairs with congestion at most 14.

We note that our algorithm uses randomization in two places: first, in constructing a random partition of vertices of the legal contracted graph into clusters in Claim 3.5, and second, in the cut-matching game, in order to embed an expander into our graph. As observed before, the random partitioning of the legal contracted graph can be run $\text{poly}(n)$ times, to ensure that the procedure succeeds with high probability. The cut-matching game only returns an $\Omega(\log k')$-expander with constant probability. If the graph returned by the cut-matching game is not an expander, then we may fail to route the desired number of the demand pairs on node-disjoint paths in the expander. In this case, we can terminate the algorithm execution, and repeat it, starting from the step where the cut-matching game is executed. Clearly, after $\text{poly}(n)$ iterations, we are guaranteed that the algorithm will succeed with high probability.

Since we lose an additional $O(\log^2 k)$ factor on the number of pairs routed due to the pre-processing step that ensures $\frac{1}{2}$-well-linkedness of the terminals, our algorithm routes $\Omega \left( \frac{\text{OPT}}{\log^{\omega} k \log \log k} \right)$ pairs with congestion at most 14 with high probability.

4 Routing with Grouping

The goal of this section is to prove Theorem 1.2. We roughly follow the algorithm from Section 3, except that we use a slightly different theorem for routing on expanders, summarized below. Its proof is similar to some arguments from [BFU94], and it uses the new constructive proof of the Lovasz Local Lemma by Moser and Tardos [MT10]. The proof is deferred to Section C of the Appendix.

**Theorem 4.1** Let $G = (V, E)$ be any $n$-vertex $\alpha$-expander (for $\alpha \leq 1$) with maximum degree $d_{\max}$, and let $c \geq 1$ be any integer. Then there is a value $m = \Theta \left( \frac{d_{\max}^{1+3/c} (\log n)^{1+5/c}}{\alpha^{1+3/c}} \right)$, such that, for any partition $G = (V_1, \ldots, V_r)$ of the vertices of $G$ into groups of size at least $m$, and for any partial matching $M \subseteq ([r] \times [r])$, we can efficiently find, for each pair $(i, j) \in M$, a path $P_{i,j}$ connecting a vertex of $V_i$ to a vertex of $V_j$, such that with high probability, the set of paths $\{P_{i,j} \mid (i, j) \in M\}$ causes vertex congestion at most $c$ in $G$.

Assume that we are given a graph $G = (V, E)$ and a set $T \subseteq V$ of $k_0$ terminals, such that $T$ is $\alpha_0$-well-linked in $G$. We will construct an expander $X$ on a subset of terminals in $T$ as in Section 3, where $X$ is a $\frac{1}{2}$-expander, $|V(X)| \leq k_0$, and the maximum degree of $X$ is bounded by $\gamma_{\text{CMC}}(k_0) = O(\log^2 k_0)$. We denote by $m = \Theta \left( (\log k_0)^{3+11/c} \right)$ the corresponding parameter from Theorem 2.1 for this setting.

We also use the following parameters. Let $\tilde{k} = \frac{k_0 \alpha_0}{12}$. Recall that we have defined, in Section 3, a parameter $k'$, whose value is $\Omega \left( \frac{k_{\text{OPT}}}{\log^{\omega} k \log \log k} \right)$, where $k$ is the number of the terminals. We define a function $q(k) = O(\log^{16.5} k \log \log k)$, so that for any integer $k$, $k' = \frac{k}{q(k)}$. We then set $\tilde{k}' = \frac{\tilde{k}}{q(\tilde{k})} = \Omega \left( \frac{k_{\text{OPT}}}{\log^{\omega} k} \right) = \Omega \left( \frac{k_{0} \alpha_{0}}{\log^{\omega} k_{0}} \right)$. Intuitively, we will define a grouping $G''$ of the terminals in $T$ into groups of size roughly $k_0 \alpha_0$, and select one representative terminal from each group. Let $T''$ denote the resulting set of terminals. We will show that the set $T''$ is $\frac{1}{2}$-well-linked in graph $G$, and $|T''| \geq \tilde{k}$. We can then apply the algorithm from Section 3 to construct an expander $X$ over a subset $T' \subseteq T''$ of $\tilde{k}'$ terminals. These terminals are in turn grouped into groups of size at least $m$, and we then apply Theorem 2.1 to route these terminals in the expander $X$. We now proceed with a formal description of the algorithm.
We define three hierarchical groupings of the terminals in $T$. Let $T$ be any spanning tree of the graph $G$. Our first step is to group the terminals in $T$ into groups of size roughly $k_0m/k'$. To do so, we use the grouping technique with the parameter $6[k_0m/k']$ on the set $T$ of terminals and the tree $T$. As a result, we obtain a partition $G$ of the set $T$ of terminals into groups of size at least $6[k_0m/k']$, and at most $18[k_0m/k']$. For each group $U \in G$, there is a tree $T_U$ spanning the terminals of $U$, and the all trees in $\{T_U\}_{U \in G}$ are edge-disjoint. The final grouping of the terminals returned by the algorithm is $G$. The size of each group in $G$ is bounded by $18[k_0m/k'] = O \left( k_0 (\log k_0)^{3+11/c} \cdot \log^{17} k_0 / \alpha_0 \right) = O \left( \left( \log k_0 \right)^{21+11/c} / \alpha_0 \right)$, as required. Assume now that we are given a set $M$ of integral $(1, G)$-restricted demands on $T$. We now show an algorithm to integrally route the demands in $M$.

For each group $U \in G$, we further partition the terminals in $U$ into at least $m$ groups of roughly equal size, using the tree $T_U$. Let $n_U = |U|$. We use the grouping technique with the parameter $\frac{1}{3} [n_U/m]$ for $U$ and tree $T_U$. We then obtain at least $m$ groups, whose sizes are at least $\frac{k_0}{k}$ and at most $\frac{36k_0}{k}$. For each group $U \in G$, let $P(U)$ denote the resulting partition of $U$, and let $G' = \bigcup_{U \in G} P(U)$ be the corresponding grouping of the terminals. Notice that again for each group $U' \in G'$, we have a tree $T_{U'}$ spanning the terminals of $U'$, such that all trees in $\{T_{U'}\}_{U' \in G'}$ are edge-disjoint.

Finally, for each group $U' \in G'$, we further partition the terminals in $U'$ into groups of size at least $\left[ \frac{1}{\alpha_0} \right]$ and at most $3 \left[ \frac{1}{\alpha_0} \right]$, using the standard grouping technique on the tree $T_{U'}$, with the parameter $\left[ \frac{1}{\alpha_0} \right]$. For each set $U' \in G'$, let $P'(U')$ be the resulting partition of $U'$, and let $G'' = \bigcup_{U' \in G'} P'(U')$ be the resulting partition of the terminals. For each set $U'' \in G''$, let $t_{U''}$ be any representative terminal from $U''$, and let $T'' = \{ t_{U''} \mid U'' \in G'' \}$. Each group $U'' \in G''$ is again associated with a tree $T_{U''}$ spanning the terminals of $U''$, and all trees in $\{T_{U''}\}_{U'' \in G''}$ are edge-disjoint. We start with the following simple claim.

**Claim 4.2** The terminals in $T''$ are $\frac{1}{2}$-well-linked.

**Proof:** Let $T_1, T_2 \subseteq T''$ be any pair of equal-sized disjoint subsets of terminals. From Observation 2.3, it is enough to show that there is a flow $F : T_1 \overset{1:1}{\rightarrow} T_2$ in $G$. We construct two sets $X, Y$ of terminals as follows. For each terminal $t \in T_1 \cup T_2$, let $S_t$ be any subset of $[1/\alpha_0]$ vertices of the set $U'' \in G''$, where $t \in U''$. We then let $X = \bigcup_{t \in T_1} S_t$, and $Y = \bigcup_{t \in T_2} S_t$. Since the terminals in $T$ are $\alpha_0$-well-linked in $G$, there is a flow $F' : X \overset{1:1/\alpha}{\rightarrow} Y$ in $G$. Scaling this flow down by factor $[1/\alpha_0]$, we obtain a flow $F''$ where every vertex of $X$ sends $\frac{1}{[1/\alpha]}$ flow units, every vertex of $Y$ receives $\frac{1}{[1/\alpha]}$ flow units, and the total edge-congestion is at most 1. In order to obtain the final flow $F$, every terminal $t \in T_1$ spreads 1 flow unit evenly among the vertices of $S_t$, along the corresponding tree $T_{U''}$, where $t \in U''$. Every vertex of $X$ then receives $\frac{1}{[1/\alpha]}$ flow units. We then use flow $F''$ to ensure that every vertex of $Y$ receives $\frac{1}{[1/\alpha]}$ flow units. Finally, every terminal $t \in T_2$ collects 1 flow unit from the vertices of $S_t$ along the corresponding tree $T_{U''}$, where $t \in U''$. It is easy to see that the congestion due to flow $F$ is at most 2.

Notice that the number of terminals in $T''$ is $|T''| \geq \frac{k_0}{[1/\alpha_0]} \geq \frac{k_0 \alpha_0}{12} \geq \tilde{k}$. Therefore, we now have a graph $G$ and a subset $T''$ of at least $\tilde{k}$ terminals, that are $\frac{1}{2}$-well-linked in $G$. As before, we perform the standard transformation that ensures that the degree of every vertex is at most 4, and the degree of every terminal in $T''$ is 1. It is easy to verify that the terminals of $T''$ remain $\frac{1}{2}$-well-linked in the resulting graph $G'$, since for any pair $T_1, T_2$ of equal-sized subsets of $T''$, there is a flow $F : T_1 \overset{1:1}{\rightarrow} T_2$ in $G$ if it exists in $G'$. Therefore, we obtain precisely the starting point of the algorithm in Section 3. We can now use the algorithm from Section 3 to construct the expander $X$ on a subset $T' \subseteq T''$ of $\tilde{k}$ terminals. The only difference is that, instead of selecting an arbitrary subset $T'$ of terminals as in the algorithm, we select $T'$ as follows. Consider the grouping $G'$ of the terminals. Let $G'$ be the grouping of
the terminals in $T''$ that $G'$ induces. We select one representative terminal from each group in $G^*$, and we let $T'$ be the set of all selected terminals. By our construction, $|T'| = |G'| \leq 6m|G| \leq \frac{6km}{6km/k'} \leq k'$. We now use the algorithm from Section 3 to construct a $\frac{1}{2}$-expander $X$ on the set $T'$ of terminals, and embed it into $G$. Recall that for each terminal $t \in T'$, we have a connected sub-graph $C_t$ of $G$, and each edge of $G$ participates in at most 12 such sub-graphs. Each edge $e$ of $X$ is mapped to a path $P_e$ in $G$, and each edge of $G$ participates in at most two such paths.

Consider the grouping $G^{**}$ of the terminals in $T'$, induced by $G$. In order to obtain $G^{**}$, we start from $G$, and we ignore terminals that do not belong to $T'$. By our construction, each group in $G^{**}$ contains at least $m$ terminals from $T'$. We assume without loss of generality that the input set of demands is $M = \{(t_1, t_2), \ldots, (t_{2r-1}, t_{2r})\}$, and for each $1 \leq i \leq 2r$, $t_i \in U_i$, where $U_i \in G$. Let $U^*_i = U_i \cap T'$, and recall that $|U^*_i| \geq m$, and $U^*_i \in G^{**}$. We now use Theorem 2.1 on graph $X$, set $T'$ of terminals, grouping $G^{**}$, and matching $M = \{(1, 2), (3, 4), \ldots, (2r-1, 2r)\}$. Let $P''$ be the set of paths returned by the theorem, where for each $1 \leq i \leq k$, there is a path $P''(i) \in P$ connecting some terminal $t'_{2j-1} \in U_{2j-1}$ to some terminal $t'_{2j} \in U_{2j}$. The paths in $P''$ cause vertex congestion at most $c$ in $X$. We transform these paths into a set $P'$ of paths connecting the same pairs of terminals in graph $G$. Since each edge of $G$ participates in at most 12 sub-graphs $C_i$, and at most two paths in set $\{P_e \mid e \in E(X)\}$, the total congestion caused by paths in $P'$ is at most $14c$. For each $1 \leq j \leq r$, let $P''(j) \in P'$ be the path connecting $t'_{2j-1}$ to $t'_{2j}$.

We then construct a path $P_j$ connecting $t_{2j-1}$ to $t_{2j}$ as follows: first connect $t_{2j-1}$ to $t'_{2j-1}$ via the tree $T_{2j-1}$, then use the path $P''(j)$ to connect $t'_{2j-1}$ to $t'_{2j}$, and finally connect $t'_{2j}$ to $t_{2j}$ via the tree $T_{2j}$. Let $P = \{P_j\}_{j=1}^r$ be the final routing. Then the total edge congestion caused by $P$ is bounded by $14c + 1$.

5 Integral Sparsifiers

In this section we prove Theorem 1.3. Notice that we can assume without loss of generality that the degree of every terminal is 1, and the number of terminals is $d$: for each terminal $t \in T$, we can simply sub-divide every edge $e$ incident on $t$ with a new vertex $v_t$, let $S_t$ be the set of these new vertices, and set $T' = \bigcup_{t \in T} S_t$. Let $G'$ be the sub-graph of the resulting graph induced by $(V \setminus T) \cup T'$, and let $T'$ be the new set of terminals. Then every terminal in $T'$ has degree 1, and $|T'| = d$. Moreover, if $H'$ is a quality $(q_1, q_2)$ integral sparsifier for $G'$, we can obtain a sparsifier $H$ for $G$ by unifying, for each $t \in T$, all vertices in $S_t$ into a single vertex $t$ in graph $H'$. It is immediate to verify that the resulting graph $H$ is a quality $(q_1, q_2)$-sparsifier for $G$.

For convenience, from now on we assume that every terminal in $T$ has degree 1, and we denote by $k$ the number of terminals in $T$. We now show a construction of a sparsifier for $G$ of size $O(k)$.

We first consider a special case where the set $T$ of terminals is $\alpha_{bw}(k)$-well-linked in graph $G$. We use Theorem 1.2 with $c = 1$ and $\alpha_0 = \alpha_{bw}(k)$ to find a partition $G$ of the terminals into subsets of size $z = O(\log^{32} k / \alpha_{bw}(k)) = O(\log^{35.5} k)$. Recall that in the proof of Theorem 1.2, we have constructed, for each group $U \in G$, a tree $T_U \subseteq G$ containing all terminals of $U$, such that the trees $\{T_U\}_{U \in G}$ are edge-disjoint.

Consider some group $U \in G$ and its corresponding tree $T_U$. We construct a new tree $T'_{U'}$, which is a minor of $T_U$, as follows. Root $T_U$ at some arbitrary vertex $r_U$. While $T_U$ contains a leaf $v$ that does not belong to $U$, delete $v$ from $T_U$. Assume now that every leaf of $T_U$ belongs to $U$. While $T_U$ contains any degree-2 vertex $v' \neq r_U$, we replace the two edges incident on $v'$ with a single edge. Let $T'_U$ be the resulting tree. It is easy to see that $T'_U$ is a minor of $T_U$, and it contains at most $2|U|$ vertices, since
Indeed, let $D$ be any set of demands over $T$. By scaling $D$ appropriately, we can assume without loss of generality that $\eta(G,D) = 1$. Let $D'$ be the demand set obtained from $D$ by scaling all demands down by factor $z$. We show that $\eta(H,D') \leq 1$. For each group $U \in \mathcal{G}$, the total demand originating from the terminals of $U$ is at most 1. For each pair $t,t' \in U$, we route the demand $D'(t,t')$ along the tree $T_U'$. For each pair $(t,t')$ with $t \in U$, $t' \in U'$, where $U \neq U'$, we route $D'(t,t')$ flow units from $t$ to $r_U$ along the tree $T_U'$, then use the edges $(r_U,r)$ and $(r,r_U)$, and finally we route $D'(t,t')$ flow units from $r_U$ to $t'$ along the tree $T_U'$. This gives a routing of $D'$ with congestion at most 1. Therefore, $\eta(H,D) \leq z$.

Assume now that we are given some collection $\mathcal{M}$ of pairs of terminals, and a set $\mathcal{P}$ of paths that connects the pairs of terminals in $\mathcal{M}$ with congestion at most $\eta$ in graph $H$. We show a collection $\mathcal{P}'$ of paths connecting the same pairs of terminals with congestion at most $31\eta$ in graph $G$.

We decompose $\mathcal{M}$ into two subsets: $\mathcal{M}_1 \subseteq \mathcal{M}$ containing pairs $(s,t)$ where both $s$ and $t$ belong to the same group $U$, and $\mathcal{M}_2 = \mathcal{M} \setminus \mathcal{M}_1$. For each group $U \in \mathcal{G}$, let $\mathcal{M}_1(U) \subseteq \mathcal{M}_1$ be the set of pairs that belong to group $U$. Then we can assume without loss of generality that all pairs in $\mathcal{M}_1(U)$ are routed along the tree $T_U'$ in $H$, and the total congestion of this routing is at most $\eta$. We can then route these pairs along the tree $T_U$ in graph $G$. We therefore obtain an integral routing of all pairs in $\mathcal{M}_1$ with congestion at most $\eta$ in graph $G$.

We now turn to pairs in $\mathcal{M}_2$. Since the pairs in $\mathcal{M}_2$ can be routed in graph $H$ with congestion at most $\eta$, for each $U \in \mathcal{G}$, the terminals of $U$ participate in at most $\eta$ pairs in $\mathcal{M}_2$. We decompose $\mathcal{M}_2$ into $2\eta$ subsets $\mathcal{M}_2^1, \ldots, \mathcal{M}_2^{2\eta}$, such that for each $1 \leq j \leq 2\eta$, for each $U \in \mathcal{G}$, at most one terminal of $U$ participates in pairs in $\mathcal{M}_2^j$. Such a decomposition can be found greedily. We consider each pair $(s,t) \in \mathcal{M}_2$ in turn. Assume that $s \in U$, $t \in U'$. We select any index $j$, such that no terminal of $U \cup U'$ participates in any pair of $\mathcal{M}_2^j$, and add $(s,t)$ to $\mathcal{M}_2^j$. Since for each $U \in \mathcal{G}$, the terminals of $U$ participate in at most $\eta$ pairs in $\mathcal{M}_2$, it is easy to see that this greedy process will give the desired decomposition. For each $1 \leq j \leq 2\eta$, the pairs in $\mathcal{M}_2^j$ now define a set $D_j$ of $(1,\mathcal{G})$-restricted demands. Using Theorem 1.2, there is an efficient algorithm that with high probability finds a routing of $D_j$ in $G$ with congestion at most 15. Therefore, we obtain a routing of all pairs in $\mathcal{M}_2$ with congestion at most $30\eta$. Overall, we route all pairs in $\mathcal{M}$ with congestion at most $31\eta$. This concludes the proof that $H$ is a quality-$(z,31)$ integral flow sparsifier for $G$. Notice that $|V(H)| \leq 2k$.

We now consider a general case, where we are given a graph $G = (V,E)$, and set $T$ of $k$ terminals, such that the degree of every terminal is 1 in $G$, but $T$ is not necessarily $\alpha_{\text{bw}}(k)$-well-linked in $G$. We compute a bandwidth-decomposition $\mathcal{W}$ of $V(G) \setminus T$ using Corollary 2.6. Let $G'$ be the graph obtained from $G$ by subdividing every edge $e \in \bigcup_{W \in \mathcal{W}} \text{out}(W)$ by a vertex $v_e$. For each cluster $W \in \mathcal{W}$, let $T_W = \{v_e | e \in \text{out}_G(W)\}$, and let $G_W = G'[W \cup T_W]$. Notice that since $W$ has the $\alpha_{\text{bw}}(k)$-bandwidth property, we are guaranteed that the set $T_W$ of terminals is $\alpha_{\text{bw}}(k)$-well-linked in graph $G_W$. We can then compute a sparsifier $H_W$ for $G_W$ as before.

In order to obtain the final sparsifier $H$, we replace, for each $W \in \mathcal{W}$, graph $G_W$ with graph $H_W$ in $G'$. In order to do so, we delete all vertices of $W$ from $G'$, and add the vertices and the edges of $H_W$ to it. Finally, for each $t \in T_W$, we identify the two copies of $t$ in the resulting graph. It is immediate to verify that the resulting graph $H$ is a quality-$(z,31)$ integral flow sparsifier for $(G,T)$, using the fact that for each $W \in \mathcal{W}$, graph $H_W$ is a quality-$(z,31)$ integral flow sparsifier for $(G_W,T_W)$.
Acknowledgements  The author thanks Matthew Andrews and Sanjeev Khanna for many inspiring discussions about routing problems. She also thanks the anonymous reviewers for many helpful comments.

References


A Proofs Omitted from Section 2

A.1 Proof of Theorem 2.1

Let \( \ell = 4d\beta(n)/\alpha \), where \( \beta(n) = O(\log n) \) is the flow-cut gap for undirected graphs. The algorithm greedily selects a source-sink pair \((s_i, t_i)\) that has a path \( P \) of length at most \( \ell \) connecting \( s_i \) to \( t_i \) in the current graph \( G \). We then remove all vertices of \( P \) from the graph \( G \) and continue. The algorithm terminates when for each remaining source-sink pair \((s_i, t_i)\), every path connecting \( s_i \) to \( t_i \) has length at least \( \ell \).

Note that in each iteration of the algorithm, we route one demand pair, and remove at most \((\ell + 1)d\) edges from the graph. The key to the algorithm analysis is to show that when the algorithm terminates, we have removed many edges from the graph, and therefore we have routed many of the demand pairs.

Let \( E' \) be the subset of edges removed from the graph by the algorithm, and let \( E'' \) be the subset of remaining edges. We first claim that there is a multicut in graph \( G \) whose value is at most \(|E'| + |E''| \cdot \beta(n)/\ell\). Indeed, let \( G' = G[E''] \) be the graph obtained when the algorithm terminates, and let \( M' \) be the set of the surviving source-sink pairs. Consider the instance of the multicut problem on graph \( G' \) with the set \( M' \) of demand pairs. Setting the weight of each edge in \( E'' \) to \( 1/\ell \), we obtain a feasible fractional solution to this multicut instance, since the length of every path connecting every pair of terminals is at least \( \ell \). Therefore, there is an integral solution to this multicut instance of value \(|E''| \cdot \beta(n)/\ell\). Adding the subset \( E' \) of edges, we obtain a feasible solution to the multicut problem on the original graph \( G \) of value \(|E'| + |E''| \cdot \beta(n)/\ell\). (Notice that all edges incident on the terminals that participate in pairs in \( M \setminus M' \) belong to \( E' \)).

On the other hand, the value of any multicut in graph \( G \) is at least \( \alpha|V|/2 \). Indeed, if \( E^* \) is any feasible solution to the multicut problem, then each connected component \( C \) of \( G \setminus E^* \) contains at most \( |V|/2 \) vertices, and therefore has at least \( \alpha|V(C)| \) out-going edges. Since each edge is counted at most twice, we get that \(|E^*| \geq \alpha|V|/2\).

We conclude that \(|E'| + |E''| \cdot \beta(n)/\ell \geq \alpha|V|/2\), and so

\[
|E'| \geq \frac{\alpha|V|}{2} - \frac{|E''| \cdot \beta(n)}{\ell} \geq \frac{\alpha|V|}{2} - \frac{|E| \cdot \beta(n)}{\ell} \geq \frac{\alpha|V|}{2} - \frac{d|V|\beta(n)}{2\ell} \geq \frac{\alpha|V|}{4}
\]

since \( \ell = 4d\beta(n)/\alpha \). Therefore, at least \( \alpha|V|/4 \) edges have been deleted from the graph. Since in each iteration we only delete at most \( d(\ell + 1) \) edges, overall the number of pairs routed is at least \( \frac{\alpha|V|}{4d(\ell + 1)} \). Setting \( \frac{\alpha|V|}{4d(\ell + 1)} = \Omega\left(\frac{\alpha^2|V|}{d^2 \log n}\right) \).

A.2 Proof of Theorem 2.7

Let \( T \) be any spanning tree of the graph \( G \), and assume that it is rooted at some vertex \( r \). We perform a number of iterations, where in each iteration we delete some edges and vertices from \( T \). For each vertex \( v \) of the tree \( T \), let \( T_v \) denote the sub-tree rooted at \( v \), and let \( w(T_v) \) denote the total weight of all vertices in \( T_v \). We build the partition \( \mathcal{G} \) of \( V \) gradually. At the beginning, \( \mathcal{G} = \emptyset \). While \( w(T) > 3p \), we perform the following iteration:

- Let \( v \) be the lowest vertex in the tree \( T \), such that \( w(T_v) > p \).
- If \( w(T_v) \leq 2p \), then we add a new group \( U \) to \( \mathcal{G} \), containing all vertices of \( T_v \), and we delete \( T_v \) from the tree \( T \), setting \( T_U = T_v \).
• Otherwise, let \( u_1, \ldots, u_k \) be the children of \( v \), and let \( j \) be the smallest index, such that \( \sum_{i=1}^j w(T_{u_i}) \geq p \). We add a new group \( U \) to \( G \), consisting of all vertices in trees \( T_{u_1}, \ldots, T_{u_j} \). Notice that \( w(U) \leq 2p \) must hold. We let \( T_U \) be the sub-tree of \( T \) consisting of \( v \) and the trees \( T_{u_1}, \ldots, T_{u_j} \). We delete the trees \( T_{u_1}, \ldots, T_{u_j} \) from the tree \( T \).

Notice that if, at the beginning of the current iteration, \( w(T) > 3p \), then at the end of the current iteration, \( w(T) > p \) must hold. In the last iteration, when \( w(T) \leq 3p \), we add a final group \( U \) to \( G \), containing all vertices currently in the tree \( T \), and we let \( T_U \) be the current tree \( T \). It is easy to verify that all conditions of the theorem hold for the final partition \( G \) of \( V \).

## B Proof of Theorem 3.1

For the proof of this theorem, we need a more general definition of well-linkedness, that was used in \([\text{CKS05}]\). Suppose we are given a graph \( G = (V, E) \), and for each vertex \( v \in V \), we are given a weight \( \pi(v) \). For a subset \( S \subseteq V \) of vertices, let \( \pi(S) = \sum_{v \in S} \pi(v) \). We say that \( G \) is \( \pi \)-flow well-linked, iff each pair \((u, v)\) of vertices can simultaneously send \( \frac{\pi(u) \cdot \pi(v)}{\pi(V)} \) flow units to each other with no congestion. We start with the following theorem, that was proved in \([\text{CKS05}]\), using a flow-well-linked graph decomposition.

**Theorem B.1 (Theorem 2.1 in [CKS05])** Let \( G = (V, E) \) be any graph and let \( M \) be a set of \( k \) source-sink pairs in \( G \). We can efficiently find a partition \( G_1, \ldots, G_k \) of \( G \) into vertex-disjoint induced subgraphs, and for each \( 1 \leq i \leq k \), find a weight function \( \pi_i : V(G_i) \to \mathbb{R}^+ \), with the following properties. Let \( M'_i \subseteq M \) be the set of source-sink pairs contained in \( G_i \), and let \( T'_i \) be the set of all terminals participating in \( M'_i \). Then:

- For all \( 1 \leq i \leq k \):
  - for all \( u \in T'_i \), \( \pi_i(u) \leq 1 \).
  - for all \( (u, v) \in M'_i \), \( \pi_i(u) = \pi_i(v) \).
  - Graph \( G_i \) is \( \pi_i \)-flow well-linked.

- \( \sum_{i=1}^k \pi_i(T'_i) = \Omega(\text{OPT}/(\beta(k) \cdot \log \text{OPT})) = \Omega(\text{OPT}/\log^2 k) \).

In order to complete the proof of the theorem, it is enough to show that we can find, for each \( 1 \leq i \leq k \), a subset \( M_i \subseteq M'_i \) of source-sink pairs, with \( |M_i| = \Omega(\pi_i(T'_i)) \), such that the set \( T_i \) of all terminals participating in pairs in \( M_i \) is \( \frac{1}{2} \)-well-linked in \( G_i \).

Fix some \( 1 \leq i \leq k \). We find a grouping \( G_i \) of the terminals in set \( T'_i \), using the weights \( \pi_i \) and the grouping parameter \( p = 2 \), as in Theorem 2.7, so for each group \( U \in G_i \), \( 2 \leq \pi_i(U) \leq 6 \). Next, we will gradually construct the set \( M_i \) of source-sink pairs, starting from \( M_i = \emptyset \). In each iteration, we will add one source-sink pair to \( M_i \), and remove some source-sink pairs from \( M'_i \), charging their weights to the pair that was added to \( M_i \). While \( M'_i \) is non-empty, we perform the following procedure:

- Let \((s, t) \in M'_i \) be any source-sink pair. Add \((s, t) \) to \( M_i \).

- If both \( s \) and \( t \) belong to the same group \( U \in G \), then for each pair \((u, v) \in M'_i \), where \( u \in U \) or \( v \in U \), remove \((u, v) \) from \( M'_i \), and charge the weight \( \pi_i(v) \) and \( \pi_i(v) \) to \((s, t) \). Notice that the total weight charged to \((s, t) \) is at most 12.
The procedure stops when \( \mathcal{M}' = \emptyset \). Let \( \mathcal{M}_i \) be the resulting set of source-sink pairs, and let \( T_i \) be the set of terminals participating in them. From the above charging scheme, it is clear that \( |\mathcal{M}_i| = \Omega(\pi_i(T_i)) \), as required. Observe also that for each group \( U \in G_i \), at most one terminal \( v \in U \) belongs to \( T_i \). Finally, we need to show that \( T_i \) is \( \frac{1}{2} \)-well-linked in \( G_i \). For each vertex \( v \in T_i \), let \( U_v \in G_i \) be the group to which \( v \) belongs.

Let \( (A, B) \) be any partition of \( V(G_i) \), and assume without loss of generality that \( |A \cap T_i| \geq |B \cap T_i| = k' \). Let \( T_A \) be any subset of \( k' \) vertices of \( A \cap T_i \), and let \( T_B = B \cap T_i \). Let \( \mathcal{M}^* \) be any matching between the vertices of \( T_A \) and the vertices of \( T_B \). We show how to route this matching with congestion at most 2 in \( G_i \), proving that \( |E_{G_i}(A, B)| \geq k'/2 \), as required.

We find the routing in two steps. In the first step, we construct a flow \( F_1 \), where for each pair \((v, v') \in \mathcal{M}^* \), the vertices in \( U_v \) send 1 flow unit in total to the vertices in \( U_{v'} \), each vertex \( x \in U_v \) sends at most \( \pi_i(x) \) flow units and each vertex \( y \in U_{v'} \) receives at most \( \pi_i(y) \) flow units, with total congestion at most 1. This flow is defined as follows. Recall that graph \( G_i \) is \( \pi_i \)-well-linked. Therefore, every pair \((x, y) \) of vertices can send \( \frac{\pi_i(x)}{\pi_i(V(G_i))} \) flow units to each other with no congestion. Let \( F \) denote this flow. Fix some pair \((v, v') \in \mathcal{M}^* \). In flow \( F \), there are \( \pi_i(U_v) \) flow units originating from the vertices in \( U_v \), that are then distributed among the vertices of \( G \), and the amount of flow each vertex \( z \) of \( G \) receives is \( \frac{\pi_i(z) \cdot \pi_i(U_v)}{\pi_i(V(G_i))} \). We scale the flow originating from the vertices in \( U_v \) down by factor \( \frac{\pi_i(U_v)}{2} \), so that every vertex \( z \) of \( G \) now receives \( 2 \cdot \frac{\pi_i(z)}{\pi_i(V(G_i))} \) flow units from \( U_v \). We perform a similar transformation for the flow originating at the vertices of \( U_{v'} \), and we concatenate both flows. As a result, we obtain a flow where the vertices in \( U_v \) send two flow units in total to the vertices in \( U_{v'} \). Taking the union of these flows over all \((v, v') \in \mathcal{M}^* \), and scaling them down by factor 2, gives us the flow \( F_1 \). It is easy to see that the total congestion caused by \( F_1 \) is at most 1. This is since each flow-path in \( F \) is used at most twice: once for each of its end-points. Finally, in order to route the matching \( \mathcal{M}^* \), consider any pair \((v, v') \in \mathcal{M}^* \). Vertex \( v \) will distribute one flow unit to the vertices in \( U_v \), along the tree \( T_{U_v} \), where the amount of flow each vertex \( x \in T_{U_v} \) receives equals to \( \frac{1}{2} \) of flow it sends out in \( F_1 \). We then use the flow \( F_1 \) to route this one flow unit to the vertices of \( U_{v'} \). Finally, vertex \( v' \) collects one flow unit from the vertices of \( U_{v'} \) along the tree \( T_{U_{v'}} \). It is easy to see that the total congestion caused by this flow is at most 2, since all trees \( \{T_U\}_{U \in G_i} \) are edge-disjoint. We conclude that \( |E_{G_i}(A, B)| \geq k'/2 \), and \( T_i \) is \( \frac{1}{2} \)-well-linked in \( G_i \).

### C Proof of Theorem 4.1

We use the result of Leighton and Rao [LR99], who have shown that any demand that is routable on an expander graph with no congestion, can also be routed on relatively short paths with small congestion. Specifically, following is a slightly rephrased statement of Theorem 18 from [LR99], and its immediate corollary.

**Theorem C.1 (Theorem 18 from [LR99])** Let \( G \) be any \( n \)-vertex \( \alpha \)-expander with maximum vertex degree \( d_{\text{max}} \). Then every pair of vertices in \( G \) can send \( \Omega(\alpha/(n \log n)) \) flow units to each other with no congestion, on flow-paths of length \( O(d_{\text{max}} \log n/\alpha) \). Moreover, such flow can be found efficiently.

**Corollary C.2** Let \( G \) be any \( n \)-vertex \( \alpha \)-expander with maximum vertex degree \( d_{\text{max}} \), and let \( M \) be any partial matching over the vertices of \( G \). Then there is an efficient randomized algorithm that finds,
for every pair \((u,v) \in M\), a set \(P_{u,v}\) of \(h = \lceil \log n \rceil\) paths length \(O(d_{\text{max}} \log n/\alpha)\) each, such that the set \(P = \bigcup_{(u,v) \in M} P_{u,v}\) of paths causes congestion \(O(\log^2 n/\alpha)\) in \(G\). The algorithm succeeds with high probability.

**Proof:** We start by showing that there is a multi-commodity flow \(f\), where every pair \((u,v) \in M\) of vertices sends one flow unit to each other simultaneously, on flow-paths of length \(O(d_{\text{max}} \log n/\alpha)\), with total congestion \(O(\log n/\alpha)\). Let \(f'\) be the flow guaranteed by Theorem C.1, scaled up by factor \(O(\log n/\alpha)\), so that every pair of vertices now sends \(1/n\) flow units to each other, with total congestion \(O(\log n/\alpha)\). Let \((u,v) \in M\) be any pair of vertices. The flow between \(u\) and \(v\) is defined as follows: \(u\) will send \(1/n\) flow units to each vertex of \(G\), using the flow \(f'\), and \(v\) will collect \(1/n\) flow units from each vertex in \(G\), using the flow \(f'\). In other words, the flow \(f\) between \(u\) and \(v\) is obtained by concatenating all flow-paths in \(f'\) originating at \(u\) with all flow-paths in \(f'\) terminating at \(v\). It is easy to see then that every flow-path in \(f'\) is used at most twice: once by each of its endpoints; all flow-paths in \(f\) have length \(O(d_{\text{max}} \log n/\alpha)\), and the total congestion of flow \(f\) is \(O(\log n/\alpha)\).

In order to find the sets \(P_{u,v}\) of paths for each pair \((u,v) \in M\), we perform the standard randomized rounding: for each pair \((u,v) \in M\), we perform \(h\) independent random trials. In each trial, we randomly choose one of the flow-paths connecting \(u\) to \(v\), with probability equal to the amount of flow sent via this path in \(f\), and add this path to \(P_{u,v}\). This ensures that all paths in set \(P_{u,v}\) have length \(O(d_{\text{max}} \log n/\alpha)\). The expected congestion on any edge is \(O(h \log n/\alpha)\), and using the standard Chernoff bounds, it is easy to see that with high probability, the congestion on every edge is \(O(\log^2 n/\alpha)\).

We are now ready to prove Theorem 4.1.

**Proof of Theorem 4.1** Let \(L = O(d_{\text{max}} \log n/\alpha)\) be the bound on the path length, and \(\eta = O(\log^2 n/\alpha)\) the bound on the congestion guaranteed by Corollary C.2. We set \(m = \frac{(4e^{-\eta^2/2}d_{\text{max}}^2 L)^{1/c}}{h} = O\left(d_{\text{max}}^{1+3/c}(\log n)^{1+5/c}/\alpha^{1+3/c}\right)\).

Given the matching \(M \subseteq ([r] \times [r])\), we assume without loss of generality, that \(M = \{(2i−1, 2i)\}_{i=1}^{[r/2]}\). We extend the matching \(M\) to the vertices of \(G\), by defining a matching \(M'\), as follows. For each \(1 \leq i \leq \lceil r/2 \rceil\), we select any maximal matching \(M_i\) between the vertices of \(V_{2i−1}\) and the vertices of \(V_{2i}\), and add it to \(M'\). Notice that \(M_i\) must contain at least \(m\) pairs of vertices. Next, we use Corollary C.2, to find a collection \(P\) of paths of length at most \(L\) each, that cause a total congestion of at most \(\eta\) in \(G\), such that for each \((u,v) \in M'\), there is a set \(P_{u,v}\) of \(h\) paths connecting \(u\) to \(v\) in \(P\). For each \(i : 1 \leq i \leq \lfloor r/2 \rfloor\), let \(B_i\) denote the union of the sets \(P_{u,v}\) of paths, for \(u \in V_{2i−1}, v \in V_{2i}\), and we call \(B_i\) the \(i\)th bundle. Notice that \(|B_i| \geq mh\). If \(B_i\) contains more than \(mh\) paths, we discard paths from \(B_i\), until \(|B_i| = mh\) holds for all \(i\).

Finally, we will select one path from each bundle, such that the resulting set of paths causes vertex congestion at most \(c\). We do so, using the constructive version of the Lovasz Local Lemma by Moser and Tardos [MT10]. The next theorem summarizes the symmetric version of the result of [MT10].

**Theorem C.3 ([MT10])** Let \(X\) be a finite set of mutually independent random variables in some probability space. Let \(A\) be a finite set of bad events determined by these variables. For each event \(A \in A\), let \(vbl(A) \subseteq X\) be the unique minimal subset of variables determining \(A\), and let \(\Gamma(A) \subseteq A\) be a subset of bad events \(B\), such that \(A \neq B\), but \(vbl(A) \cap vbl(B) \neq \emptyset\). Assume further that for each \(A \in A\), \(|\Gamma(A)| \leq D\), \(\Pr[A] \leq p\), and \(\text{ep}(D+1) \leq 1\). Then there is an efficient randomized algorithm that with high probability finds an assignment to the variables of \(X\), such that none of the events in \(A\) holds.
For each bundle $B_i$ we choose one of its paths $P_i$ independently at random. We let $x_i$ be the variable indicating which path has been chosen. For each vertex $v \in V$, we let $\beta_v$ be the bad event that $v$ belongs to more than $c$ of the chosen paths. Since the congestion on every edge due to paths in $P$ is at most $\eta$, and the maximum vertex degree is $d_{\text{max}}$, we get that there are at most $(\eta d_{\text{max}})^{c+1}$ potential $(c+1)$-tuples of paths containing $v$ (where we only consider $(c+1)$-tuples containing at most one path from each bundle), and each tuple is chosen with probability $1/((hm)^{c+1})$. Therefore, $\Pr[\beta_v] \leq \frac{(\eta d_{\text{max}})^{c+1}}{(hm)^{c+1}}$. We denote $p = \frac{(\eta d_{\text{max}})^{c+1}}{(hm)^{c+1}}$.

The set $vbl(\beta_v)$ of variables contains all variables $x_i$, where the bundle $B_i$ contains a path $P \in P$, such that $v \in P$. Therefore, $|vbl(\beta_v)| \leq \eta d_{\text{max}}$. For each such variable $x_i$, there are $mh$ paths participating in the bundle $B_i$, each of which contains at most $(L+1)$ vertices. Therefore, $|\Gamma(\beta_v)| \leq mh(L+1)\eta d_{\text{max}}$. We denote this value by $D$.

It now only remains to show that $(D+1)ep \leq 1$, which follows from the choice of $m = \frac{(4e\eta^{c+2}d_{\text{max}}^{c+2}L)^{1/c}}{h}$.

\[\Box\]

### D Table of Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma_{\text{CMG}}(k)$</td>
<td>$\Theta(\log^2 k)$</td>
</tr>
<tr>
<td>$\alpha_{\text{ARV}}(k)$</td>
<td>$O(\sqrt{\log k})$</td>
</tr>
<tr>
<td>$\alpha(k)$</td>
<td>$\frac{1}{2^{1-\gamma_{\text{CMG}}(k)-0.5}} = \Omega\left(\frac{1}{\log^4 k}\right)$</td>
</tr>
<tr>
<td>$\alpha_{\text{BW}}(k)$</td>
<td>$\frac{\alpha(k)}{\alpha_{\text{ARV}}(k)} = \Omega\left(\frac{1}{\log^{0.5} k}\right)$</td>
</tr>
<tr>
<td>$\beta(k)$</td>
<td>$\Theta(\log k)$</td>
</tr>
<tr>
<td>$k_1$</td>
<td>$\frac{k}{192\gamma^3 \log \gamma} = \Omega\left(\frac{k}{\log^5 k \log \log k}\right)$</td>
</tr>
<tr>
<td>$p$</td>
<td>$\frac{8\beta(k)}{\alpha_{\text{BW}}(k)} = O(\log^{1.5} k)$</td>
</tr>
<tr>
<td>$k'$</td>
<td>$\left\lfloor \frac{k}{2^{1-\gamma}} \cdot \left\lfloor \frac{k}{6p} \right\rfloor \right\rfloor = \Omega\left(\frac{k}{\log^{0.5} \eta k \log \log k}\right)$</td>
</tr>
<tr>
<td>$k^*$</td>
<td>$\left\lfloor \frac{k}{6p} \right\rfloor$</td>
</tr>
</tbody>
</table>

Parameter from the cut-matching game of [KRV09], from Theorem 2.2. Is also denoted by $\gamma$.

Approximation factor of the algorithm of [ARV09] for Sparsest Cut.

Well-linkedness parameter from Theorem 2.5.

Well-linkedness parameter from Corollary 2.6.

Flow-cut gap for concurrent flow on $k$ terminals.

Parameter from the definition of legal contracted graphs.

Grouping parameter for the sets $\Gamma_j$.

Number of vertices in the expander $X$.

Size of sets $\Gamma'_j$ (that contain at most one edge from each group of $G_j$).