SIAM J. COMPUT.
Vol. 53, No. 3, pp. 588–623

CLUSTER BEFORE YOU HALLUCINATE: NODE-CAPACITATED NETWORK DESIGN AND ENERGY EFFICIENT ROUTING*

RAVISHANKAR KRISHNASWAMY†, VISWANATH NAGARAJAN‡, KIRK PRUHS§, AND CLIFFORD STEIN¶

Abstract. We consider the following node-capacitated network design problem. The input is an undirected graph, a set of demands, uniform node capacity, and arbitrary node costs. The goal is to find a minimum node-cost subgraph that supports all demands concurrently subject to the node capacities. We consider both single- and multicommodity demands and provide the first polylogarithmic approximation guarantees. For single-commodity demands (i.e., all request pairs have the same sink node), we obtain an $O(\log^2 n)$ approximation to the cost with an $O(\log^3 n)$ factor violation in node capacities. For multicommodity demands, we obtain an $O(\log^4 n)$ approximation to the cost with an $O(\log^{10} n)$ factor violation in node capacities. We use a variety of techniques, including single-sink confluent flows, low-load set cover, random sampling, and cut-sparsification. We also develop new techniques for clustering multicommodity demands into (nearly) node-disjoint clusters, which may be of independent interest. Moreover, this network design problem has applications to energy-efficient virtual circuit routing. In this setting, there is a network of routers that are speed scalable and that may be shut down when idle. We assume the standard model for power: the power consumed by a router with load (speed) s is $\sigma + s^\alpha$, where σ is the static power and the exponent $\alpha > 1$. We obtain the first polylogarithmic approximation algorithms for this problem when speed-scaling occurs on nodes of a network.

Key words. network design, approximation algorithms, routing

MSC codes. 68W25, 68W20

DOI. 10.1137/20M1360645

1. Introduction.

Network design problems involve finding a minimum-cost subgraph of a given graph while satisfying certain demand requirements. Classic examples include Steiner tree, Steiner forest, survivable network design, and buy-at-bulk network design. Good approximation algorithms are known for all these basic network design problems [13, 1, 27, 33, 29, 18]. However, these problems become significantly harder in the presence of capacities, and much less is known for *capacitated* network design problems. In this paper, we study a natural node-capacitated network design problem and provide the first polylogarithmic approximation algorithms for it.

*Received by the editors August 18, 2020; accepted for publication (in revised form) February 20, 2024; published electronically May 20, 2024. A preliminary version of this paper appeared in the Proceedings of the ACM Symposium on Theory of Computing (STOC) 2014.

https://doi.org/10.1137/20M1360645

Funding: The second author was supported in part by NSF grants CCF-2006778 and CMMI-1940766. The third author was supported in part by NSF grants CCF-1907673, CCF-2036077, and CCF-2209654 and an IBM Faculty Award. The fourth author was supported in part by NSF grant CCF-2218677 and ONR grant ONR-13533312 and by the Wai T. Chang Chair in Industrial Engineering and Operations Research at Columbia University.

†Microsoft Research Vigyan Building, 9, Lavelle Road, Bangalore 560018 India (ravishankar.k@gmail.com).

‡Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI 48109 USA (viswa@umich.edu).

§Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260 USA (krp2@pitt.edu).

¶Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027 USA (cliff@ieor.columbia.edu).

In the *multicommodity node-capacitated network design problem* (MCNC), there is an undirected graph $G=(V,E)$, where each node $v \in V$ has cost $c_v \geq 0$ and uniform capacity q. There are also k request-pairs of the form (s_i, t_i, d_i), where $s_i \in V$ is the source, $t_i \in V$ is the sink, and $1 \leq d_i \leq q$ is the demand. A feasible solution is a subset of nodes $U \subseteq V$ such that the graph $G[U]$ induced on nodes U (where each node has capacity q) can *concurrently* support d_i units of unsplittable flow between s_i and t_i for each request-pair $i \in [k]$. The objective is to minimize the total cost $c(U) := \sum_{v \in U} c_v$ of the solution. Instead of requiring unsplittable flows, one could alternatively ask for a splittable (i.e., fractional) flow for the demands. However, this does not change the problem significantly. In fact, our approximation guarantees also hold relative to an optimal solution for splittable flows.

Our algorithms will find bicriteria approximations, where the solution is allowed to violate the capacity constraints by some factor. A (β, γ) bicriteria approximation algorithm for MCNC finds a solution U such that (i) the cost $c(U)$ is at most β times the optimum and (ii) all request-pairs can be routed concurrently in $G[U]$ using capacity at most $\gamma \cdot q$ at each node.

Other than being a natural theoretical model, MCNC has applications in energy-efficient routing. Indeed, this was our primary motivation to study MCNC. Improving the energy efficiency of telecommunication (telecom) networks is an important practical issue. In their 2020 report [39] "The Case for Committing to Greener Telecom Networks" McKinsey reported that telecom operators account for 2 to 3 percent of total global energy demand, often making them some of the most energy-intensive companies in their geographic markets. But the report noted that all operators have considerable scope to cut energy costs and consumption, with many operators having the potential to reduce energy consumption by at least 15 to 20 percent. Further improved optimization policies was listed as one of the four key energy reduction opportunities. In this paper, we consider virtual circuit routing, which is used by several network protocols to achieve reliable communication [38].

Formally, we consider virtual circuit routing protocols (where each connection is assigned a fixed route in the network) with an objective of minimizing energy, in a network of routers that (i) are speed scalable, and (ii) may be shut down when idle. We use the standard model for a router's power-rate curve, which is the same as in [4, 3, 10, 7]. In this model, the energy cost incurred by a router operating at speed x (which is assumed to be the total traffic passing through the router) is given by

$$f(x) = \begin{cases} 0 & \text{if } x = 0, \\ \sigma + x^{\alpha} & \text{if } x > 0. \end{cases} \tag{1.1}$$

Above, parameter σ is the static power (that is always used when the router is turned on) and parameter $\alpha > 1$ specifies the energy inefficiency of the router. The value of α is in the range $[1.1, 3]$ for essentially all technologies [12, 48]. We assume that all network components are homogeneous: so σ and α are uniform across all routers. This is also the setting in several prior works [3, 10, 7].

In this paper, we obtain the first polylogarithmic approximation algorithms for virtual circuit routing with speed-scalable components at *nodes* of the network. All previous papers considered the simpler setting where speed-scaling occurred at edges. Although speed-scalable edges (corresponding to network links) are plausible, it is more realistic that speed-scaling occurs at nodes (corresponding to routers).

Formally, in the *node-cost energy efficient routing problem* (NEERP), we are given an undirected graph $G=(V,E)$, with nonnegative multipliers on nodes $\{c_v\}_{v \in V}$ and a uniform energy cost function (1.1). We are also given a collection of k request-pairs

of the form (s_i, t_i, d_i), where, for each $i \in [k]$, $s_i \in V$ is the source node, $t_i \in V$ is the sink node, and $d_i \geq 1$ is the demand. The goal in NEERP is to find a path P_i connecting s_i and t_i for each $i \in [k]$ so as to minimize the overall energy cost:

$$\sum_{v \in V} c_v \cdot f\left(\sum_{i: v \in P_i} d_i\right).$$

It turns out that NEERP reduces to the capacitated network design problem MCNC, as stated in the following result. This reduction is implicit in [3], where it was applied to the edge-version, and we provide a proof in Appendix A for completeness.

THEOREM 1.1 ([3]). *If there is a (β, γ) bicriteria approximation algorithm for* MCNC, *then there is an $O(\beta \cdot \gamma^{\alpha})$-approximation algorithm for* NEERP.

Preliminary simplifications. We refer to the set $\{s_i\}_{i=1}^k \cup \{t_i\}_{i=1}^k$ of all sources and sinks in MCNC as *terminals.* We assume (without loss of generality) that (i) all terminals are distinct, i.e., each node in V is the source or sink of at most one request and (ii) each terminal is a leaf node, i.e., has degree one. This can be ensured by adding $2k$ new terminals of cost zero, where each new terminal node (s_i or t_i) is connected only to the original terminal. So the number of nodes $n \geq 2k$. In some applications, we may also have $n \gg k$ (which is common in network design problems). So, we state our approximation ratios in terms of both n and k.

We also note that, without loss of generality, *zero-cost* nodes in MCNC may have capacity that is any integral multiple of q. To see this, consider any zero-cost node $v \in V$ (i.e., with $c_v = 0$) having capacity $z \cdot q$, where $z \geq 1$ is an integer. Then, we simply introduce z copies $v_1, \ldots, v_z$ of node v, each having uniform capacity q and zero cost. As all the copies have zero cost, it is clear that the two MCNC instances are equivalent. We note that this reduction increases the number of nodes, but we always have $z \leq k$ as the total demand in any instance is at most kq; so the number of nodes in the new instance is at most nk.

*Single-sink node-capacitated network design (*SSNC*).* We also consider separately the *single-sink* special case of MCNC, where there is a common node $t \in V$ with $t_i = t$ for all $i \in [k]$. The sink node t is assumed to have zero cost; this is without loss of generality as t must be included in any feasible solution. Moreover, the capacity of t is kq so that all demands can be routed into it. We also assume that each source is a distinct leaf node. The single-sink problem serves as a simpler setting to explain our techniques and is also used in the multicommodity algorithm.

1.1. Our results and techniques. Our first main result is the following theorem.

THEOREM 1.2. *There is an $(O(\log^2 n), O(\log^3 n))$ bicriteria approximation algorithm for single-sink node-capacitated network design.*

In order to motivate our approach, we illustrate two corner cases, which are interesting in their own right. If the total demand is smaller than the capacity, i.e., $\sum_i d_i < q$, then the problem reduces to computing a minimum-cost node-weighted Steiner tree, for which there are $O(\log k)$-approximation algorithms [36, 28]. At the other end of the spectrum, if each demand d_i is large, i.e., $\min_i d_i = \Omega(q)$, then each of these requests essentially has to route its demand on a disjoint path, and this problem can be solved by using techniques from low-congestion routing [46]. (See also Appendix B.1.) Prior results for the *edge-capacitated* problem [10, 3, 7] were based on a combination of these ideas and can be summarized as follows: (i) choose an

approximately min-cost Steiner tree T connecting all the sources and the sink, (ii) partition T into *edge-disjoint* subtrees (which we call clusters) having total demand $\approx q$ in each, (iii) choose one "leader" in each cluster and aggregate all demand in the cluster at the leader, and (iv) route q units of flow from each leader to the sink t using disjoint paths. The overall edge-congestion is bounded because the clusters are edge-disjoint and the (disjoint) path chosen by each leader suffices to route the entire demand in that cluster (which is at most q). A crucial ingredient in this approach is that any tree can be partitioned into *edge-disjoint* subtrees/clusters containing $\approx q$ demand each.

However, in the node-capacitated setting, there may not exist a *node-disjoint* clustering of the minimum Steiner tree into subtrees of $\approx q$ demand each! For example, the tree T could just be a star with all the sources and sink as leaves, which means that the center node will appear in every cluster. So, instead of partitioning a min-cost Steiner tree into clusters (which may not be possible), we directly aim to find low-cost node-disjoint clusters. However, it is not a priori clear that such a clustering must always exist. Our first step in Theorem 1.2 is to prove the *existence* of node-disjoint clusters of cost at most the optimal SSNC value where each cluster has $O(\log n) \cdot q$ demand. This proof relies on the existence of single-sink *confluent flows* [20]. In fact, we show that each such cluster can be rooted at a neighbor of the sink so that routing from each cluster to the sink is trivial. Our second step in Theorem 1.2 is to *efficiently find* such a clustering. We achieve this by formulating the single-sink clustering problem as an instance of *low load set cover* [9]. Here, each subtree with $O(\log n) \cdot q$ demand is a "set" and we need to pick a min-cost collection of sets such that the number of sets containing any node is bounded (which will ensure approximate node-disjointness). The approximation algorithm for low load set cover from [9] requires a subroutine for the related "minimum ratio" problem, for which we obtain an $O(\log n)$-approximation algorithm using the *partial node weighted Steiner tree* problem [37, 43]. These details are presented in section 2.

Our second main result is as follows.

THEOREM 1.3. *There is an* $(O(\log^2 n \log^2 k), O(\log^6 n \log^4 k))$ *bicriteria approximation algorithm for multicommodity node-capacitated network design.*

We note that an $\Omega(\frac{\log \log n}{\log \log \log n})$ factor violation in the node-capacity is necessary for any nontrivial approximation on the cost, due to the hardness of the undirected congestion minimization problem [5].

Our high-level approach is similar to that for the single-sink case. First, we find a *clustering* of all source and sink nodes into nearly node-disjoint subtrees of small cost such that each cluster has at most $q \cdot polylog(n)$ demand inside. Next, we find a *routing* of demands across different clusters (from sources to sinks) while incurring low node-congestion. However, both these steps are significantly more complicated than the single-sink case, as outlined next.

For multicommodity clustering, as in the single-sink case, we need to prove both the existence and computation of (nearly) node-disjoint clusters. However, there is no multicommodity notion of confluent flow, which was used crucially in the single-sink existence proof. Moreover, the low-load set-cover approach is not applicable either because the "minimum ratio" problem in the multicommodity case is at least as hard to approximate as the *dense-k-subgraph* problem, which is believed to not admit any polylogarithmic approximation [25, 11, 42]. We also need to modify the notion of an allowed cluster in the multicommodity case. Ideally, we would like each cluster to be "heavy," i.e., having demand at least q (and at most $q \cdot polylog(n)$), which is useful

in the subsequent routing step. However, this may not always be possible, so we also allow "internal" clusters where a constant fraction of the demand in the cluster comes from requests with *both* source and sink in that cluster. Then, we obtain a low cost clustering where each cluster is either heavy or internal. We also ensure that the clusters have low node congestion, i.e., each node appears in at most $polylog(n)$ many clusters. Our algorithm constructs these clusters in an iterative manner, where we use the single-sink algorithm (Theorem 1.2) in each iteration. We start off with each terminal being a singleton cluster and continue merging clusters until each cluster is either heavy or internal. Crucially, we prove that the SSNC instances solved in each iteration have low cost by producing a "witness solution" using the optimal MCNC solution. We then use the SSNC solutions to merge clusters so that the number of clusters reduces by a constant factor in each iteration: this implies that $O(\log k)$ iterations suffice. The actual algorithm is more subtle, and we only end up clustering a constant fraction of the total demand. See section 3.1 for a more detailed overview of the clustering algorithm.

For multicommodity routing, we consider two cases depending on whether there are more demands in internal or heavy clusters. If a constant fraction of the demand is contained in internal clusters, then we do not have to route across clusters: we just route all "internal" demands using the respective subtrees. The harder case is when a constant fraction of the demand is in heavy clusters. Here, we find a low-cost routing across heavy clusters using a sampling/hallucination based approach from the edge-capacitated problem [7]. However, unlike the edge version [7], in the node version we need to drop some demands in the routing step. This is required to ensure that the min-cut in the demand graph is large, which in turn is needed for the cut-sparisification result [34] that we use. See section 3.2 for a more detailed overview of the routing algorithm.

Finally, after combining the clustering and routing steps, we obtain a solution that can support a constant fraction of the total demands. So, we need to apply these steps recursively on the remaining demands to complete the proof of Theorem 1.3.

We also note that the approximation ratios in Theorems 1.2 and 1.3 can be strengthened to be relative to an optimal *splittable* routing: see Appendix A.2.

Using Theorems 1.2 and 1.3 along with the reduction in Theorem 1.1, we obtain the following corollaries.

COROLLARY 1.4. *There is an $O(\log^{3\alpha+2} n)$-approximation algorithm for the single-sink node-cost energy-efficient routing problem.*

COROLLARY 1.5. *There is an $O(\log^{10\alpha+4} n)$-approximation algorithm for the multicommodity node-cost energy-efficient routing problem.*

1.2. Related work. Approximation algorithms for the edge-capacitated version of MCNC have been studied previously in [4, 3, 10, 7]. The node-capacitated problem that we study is more general, and we obtain the first approximation algorithms. A key challenge that needs to be addressed in these results is that the problem has similarities to both convex and concave cost flows. When the capacity q is small, the MCNC problem is similar to convex-cost flow, where it is preferable to spread flow over disjoint paths. On the other hand, when the capacity q is large, MCNC is similar to concave-cost flow, where one prefers to aggregate flow. In [3], the authors showed that these competing forces (to spread out or aggregate flow) can be "poly-log-balanced" by giving a bicriteria polylogarithmic approximation algorithm for the multicommodity edge version of the problem. Moreover, [4] showed an $\Omega(\log^{1/4} n)$ inapproximability

result for the edge version, under standard complexity theoretic assumptions. Later, [7] obtained an improved $(O(\log n), O(\log n))$ bicriteria approximation algorithm for edge-capacitated MCNC. In fact, [7] also studied the online version (where requests arrive over time) and obtained an $(O(\log n), O(\log^2 n))$ bicriteria competitive ratio. A key technique in [7] was a random-sampling idea, where each request i "hallucinates" that it wants to route q units with probability $\approx \frac{d_i}{q}$. We also make use of this idea in our paper.

The NEERP problem has also been studied in the special case that speed scaling occurs on the edges instead of the nodes. As noted earlier, it is more realistic to have speed-scalable nodes rather than edges. Presumably, the assumption in these previous papers that speed scaling occurs on the edges was motivated by reasons of mathematical tractability, as network design problems with edge costs are usually easier to solve than the corresponding problems with node costs. The paper [3] obtained a $\log^{O(\alpha)} n$-approximation algorithm for the edge-cost NEERP. The paper [10] considered the single-sink special case (with edge costs) and obtained an $O(1)$-approximation algorithm and $O(\log^{2\alpha+1} n)$-competitive randomized online algorithm. Later, [7] obtained a simple $O(\log^{\alpha} n)$-approximation algorithm for the multicommodity edge version, which was also extended to an $\tilde{O}(\log^{3\alpha+1} n)$-competitive randomized online algorithm.

We note that the MCNC problem is a special case of a very general model, called *fixed-charge network design,* that has been studied extensively in the operations research literature; see, e.g., [35, 22, 32]. The focus in these papers has been on solving the problem exactly, which is different from our goal of polynomial-time approximation algorithms.

An $O(\log k)$-approximation algorithm for the basic *node-weighted* Steiner tree problem was obtained in [36], which is also the best possible approximation ratio (as set cover is a special case). This contrasts with the usual (edge-weighted) Steiner tree, for which constant-factor approximations are known [13]. Our algorithm also makes use of the *partial* node-weighted Steiner tree (PNWST) problem, where we only want to connect a certain number of terminals. An $O(\log n)$ approximation algorithm for PNWST was obtained in [37, 43].

Buy-at-bulk network design is also somewhat related to our model. Here, the cost on a network element (edge or node) is a *concave* function of the load through it. Polylogarithmic approximation algorithms are known for both edge-weighted [8, 29, 18] and node-weighted cases [19, 6]. The paper [2] also showed polylogarithmic hardness of approximation for buy-at-bulk network design. From a technical standpoint, the hallucination idea used in [7] and also in our algorithm is similar to the sample-augment framework in [30] for solving buy-at-bulk problems. However, our algorithm analysis is quite different from those for buy-at-bulk and is more similar in spirit to the analysis of cut-sparsification algorithms [34, 47, 26].

The *survivable network design* problem (SNDP) is a different (but well-studied) multicommodity network design problem. Here, the goal is to select a minimum-cost subgraph that can route a set of demands *individually;* i.e., each demand should be routable in the subgraph (just by itself). A 2-approximation algorithm is known for SNDP with edge-connectivity requirements [33]. The node-connectivity SNDP has also been studied extensively, with the best approximation ratio being $O(k^3 \log n)$ for edge costs [21] and $O(k^4 \log^2 n)$ for node costs [45]; here k is the largest demand. Vertex-connectivity SNDP is also $\Omega(k^{\epsilon})$-hard to approximate [17]. There has also been some work on capacitated SNDP [14, 15, 16, 31]. We note that capacitated

SNDP differs significantly from MCNC because the goal in SNDP is to route each request-pair in isolation, whereas our goal is to route all requests concurrently.

Very recently (after the conference version of this paper), [24, 44] considered the NEERP problem with *nonuniform* cost functions, where the σ and α parameters in (1.1) are different across nodes. In fact, their results apply to a larger class of "generalized network design" problems, which includes multicommodity routing on directed graphs. The paper [24] gave an $O(\max_{v \in V} \sigma_v^{1/\alpha_v})$-approximation algorithm for nonuniform NEERP, where σ_v and α_v are the cost parameters for each node (and α_v's are constant). Moreover, [44] obtained an online algorithm for nonuniform NEERP with the same competitive ratio. We note that these results are incomparable to Corollary 1.5: we obtain approximation ratios that are polylogarithmic in the input size (n and k), whereas these results in [24, 44] have a polynomial dependence (albeit on the cost parameters).

2. Single-sink node-capacitated network design. The input to the SSNC problem consists of an undirected graph $G = (V, E)$, with $|V| = n$, and a collection of k sources $\mathcal{D} = \{s_i \,|i \in [k]\}$ with respective demands $\{1 \le d_i \le q \,|i \in [k]\}$. Recall that each source node has degree one. There is a specified sink $t \in V$ to which each source s_i wants to send d_i units of flow unsplittably. Each node $v \in V \setminus \{t\}$ has a cost c_v and uniform capacity q; the sink t has zero cost and capacity kq (so all demands can be routed into it). Recall that zero-cost nodes in MCNC (and SSNC) are allowed to have larger capacity than q. The output is a subset of nodes $V' \subseteq V$ such that the graph $G[V']$ induced by the nodes V' can concurrently support an unsplittable flow of d_i units from each source s_i to the sink t. The objective is to minimize the total cost $c(V') = \sum_{v \in V'} c(v)$. We will also refer to the nodes $\{s_i \,|i \in [k]\}$ as *terminals*. In our analysis, we use Opt to denote the cost of the optimal SSNC solution.

A simple but important notion is that of a single-sink cluster, defined below.

Definition 2.1 (SSNC cluster). *A cluster is any subtree of graph G containing the sink t. The demand of the cluster is the total demand of all sources in it.*

The key step in our single-sink algorithm is to find a collection of nearly node-disjoint clusters, each assigned roughly q demand. An important step is to even show the existence of such clusters, which we do in section 2.1. The existence argument is based on *single-sink confluent flows* [20]. We then give an algorithm for finding such clusters in section 2.2. This algorithm relies (in a black-box fashion) on two other results: an $O(\log n)$-approximation algorithm for *partial node-weighted Steiner tree* [37, 43], and a logarithmic bicriteria approximation for *low load set cover* [9]. At a high level, we model a set cover instance on the graph, where any cluster is a set, and the goal is to find a minimum cost set cover of all terminals such that no node is in too many sets. The algorithm of [9] requires a *min-ratio* oracle, for which we use the partial node-weighted Steiner tree algorithm. Finally, we just select all the nodes in the clusters as our solution. The node congestion can be bounded using the fact that each cluster has roughly q demand and that the clusters are nearly disjoint.

Confluent flow. Consider any n-node directed graph with sink node t, sources $\{s_i\}_{i=1}^k$ with demands $\{d_i\}_{i=1}^k$, and uniform node capacity q at all nodes except the sink (which has infinite capacity). Again, we assume that each demand is at most q. A flow is said to be *confluent* if for every node u there is at most one edge (u, v) out of u that carries positive flow. Note that the edges carrying positive flow in any confluent flow correspond to an arborescence directed toward the sink t.

THEOREM 2.2 (Theorem 20 in [20]). *Consider any directed graph as above with a splittable routing $\mathcal{F}^*$ that sends d_i units from each source s_i (for $i \in [k]$) to sink t, while respecting node capacities. Then, there is a confluent flow $\mathcal{F}$ that routes all demands where the total flow through any node (other than t) is at most $(1+\ln n)q$.*

The multiple sinks referred to in [20] correspond to the in-neighbors of our single sink t (which are at most n in number).

2.1. Existence of good clustering. We first show that there exists a "good" clustering of the source nodes into node-disjoint clusters.

LEMMA 2.3. *Given any instance of* SSNC *with optimal cost* Opt*, there exists a collection $\{T_i\}_{i=1}^{g}$ of clusters such that the following hold:*

(i) *The demand of each cluster T_i is at most $(1+\ln n)\cdot q$.*
(ii) *Every source lies in some cluster.*
(iii) *The clusters are node-disjoint except at t.*
(iv) *The total cost is $\sum_{i=1}^{g}\sum_{v\in T_i} c_v \le$ Opt.*

Proof. Let $V^* \subseteq V$ denote the set of nodes in an optimal solution and $\mathcal{F}^*$ denote an optimal flow for the sources $\mathcal{D}$. Note that $\mathcal{F}^*$ sends at most q flow through each node (except t). We now apply Theorem 2.2 on the graph induced on V^* to obtain a *confluent flow* $\mathcal{F}$ where the flow through each node (other than t) is at most $q(1+\ln n)$. Recall that $\mathcal{F}$ corresponds to an arborescence $\mathcal{T}$ directed toward the sink t. Let $\{r_i\}_{i=1}^{g}$ denote all neighbors of t contained in arborescence $\mathcal{T}$. For each $i \in [g]$, let T_i denote the subtree of $\mathcal{T}$ rooted at r_i, along with the edge (t, r_i). Note that $\{T_i\}_{i=1}^{g}$ are node-disjoint except at t. Moreover, the total demand in each subtree T_i is at most $q(1+\ln n)$ because all of these demands pass through node r_i. Finally, $\mathcal{T} = \cup_{i=1}^{g} T_i$ contains all the sources as the confluent flow $\mathcal{F}$ routes every demand.

We claim that the clusters $\{T_i\}_{i=1}^{g}$ satisfy all the conditions in the lemma. Conditions (i)–(iii) follow directly from the above construction. Condition (iv) follows from (iii) and the fact that all nodes of T_i are contained in V^*. ∎

2.2. Finding a good clustering. The previous subsection only establishes the existence of a good clustering; in this subsection we explain how to efficiently find such a clustering.

LEMMA 2.4. *There is an efficient algorithm that, for any instance of* SSNC *with optimal cost* Opt*, finds a collection of clusters $\{T_i\}_{i=1}^{g}$ such that the following hold:*

(i) *The demand of each cluster T_i is at most $(1+\ln n)\cdot q$.*
(ii) *Every source lies in some cluster.*
(iii) *Every node in $V \setminus \{t\}$ appears in at most $O(\log^2 n)$ clusters.*
(iv) *The total cost is $\sum_{i=1}^{g}\sum_{v\in T_i} c_v \le O(\log^2 n)\cdot$* Opt*.*

Given this clustering, our final solution to the SSNC instance is just $\cup_{i=1}^{g} T_i$. As each cluster contains the sink t, there is no need for a separate routing step (from clusters to t). By Lemma 2.4 property (iv), the cost is $O(\log^2 n)\cdot$ Opt. Moreover, by properties (i), (ii), and (iii), all demands can be routed unsplittably with a total flow of $O(\log^3 n)\cdot q$ through any node (other than t). This completes the proof of Theorem 1.2. It remains to prove Lemma 2.4, which we will do now. Our algorithm will use an approximation algorithm for *low load set cover* (LLSC) [9], defined next.

Low load set cover (LLSC). In this problem, we are given a set system $(U, \mathcal{C})$ with elements U and sets $\mathcal{C} \subseteq 2^U$, costs $\{c_v : v \in U\}$, and bound $p \ge 1$. The cost of any set $S \in \mathcal{C}$ is $c(S) := \sum_{v\in S} c_v$, the sum of its element costs. We note that the collection

$\mathcal{C}$ may be exponentially large and specified implicitly. The cost of any collection $\mathcal{C}' \subseteq \mathcal{C}$ is $c(\mathcal{C}') := \sum_{S \in \mathcal{C}'} c(S) = \sum_{S \in \mathcal{C}'} \sum_{v \in S} c_v$, the sum of its set costs. We are also given two special subsets of elements: *required* elements $W \subseteq U$ that need to be covered, and *capacitated* elements $L \subseteq U$.[1] The goal is to find a minimum cost set cover $\mathcal{C}' \subseteq \mathcal{C}$ for the required elements W (i.e., $\cup_{S \in \mathcal{C}'} S \supseteq W$) such that each capacitated element $e \in L$ appears in at most p sets of $\mathcal{C}'$. An approximation algorithm for LLSC is given in [9], which relies on the following subproblem. The *min-ratio oracle* for LLSC takes, as input, nonnegative element-costs $\{\eta_v : v \in U\}$ and a subset $X \subseteq W$ (of already covered required elements) and outputs a set $S \in \mathcal{C}$ that minimizes $\frac{\sum_{v \in S} \eta_v}{|S \cap (W \setminus X)|}$. We use the following result on LLSC.

THEOREM 2.5 ([9]). *Assuming a ρ-approximate min-ratio oracle, there is an algorithm for the LLSC problem that finds a solution of cost $O(\rho \log |U|)$ times the optimum and which covers each capacitated element $O(\rho \log |U|)\, p$ times.*

In other words, this is a bicriteria approximation algorithm that violates both the cost and capacities by an $O(\rho \log |U|)$ factor.

The SSNC problem as LLSC. We now prove Lemma 2.4 by casting the desired clustering problem as an instance of LLSC. The elements are the nodes V of the original SSNC problem. The costs $\{c_v : v \in V\}$ are the node-costs in SSNC. The required elements are all the sources $W = \{s_i\}_{i=1}^{k}$. For any $v \in W$ its demand $d_v := d_i$ where $v = s_i$ is the corresponding source node. The capacitated elements are $L := V \setminus \{t\}$ and the bound $p = 1$. Let $Q := (1 + \ln n) \cdot q$. The collection $\mathcal{C}$ of sets is defined as follows. There is a set corresponding to each cluster (Definition 2.1), having demand at most Q. To reduce notation, we use T to denote the subtree representing the cluster as well as the nodes in this cluster. By Lemma 2.3, the optimal value of this LLSC instance is at most Opt.

Next, we provide an approximation algorithm for the min-ratio oracle for such LLSC instances. Our min-ratio algorithm relies on another known problem.

Partial node-weighted Steiner tree (PNWST). The input is an undirected graph $G = (V, E)$ with node-weights $\{\eta_v : v \in V\}$, sink $t \in V$, rewards $\{\pi_v : v \in V\}$, and target τ. Both the node-weights and rewards are nonnegative. The objective is to find a minimum node cost Steiner tree containing t having total reward at least τ. We will use the following known result.

THEOREM 2.6 ([37, 43]). *There is an $O(\log |V|)$-approximation algorithm for the partial node-weighted Steiner tree problem.*

LEMMA 2.7. *There is an $O(\log n)$-approximate min-ratio oracle for the* SSNC *clustering problem.*

Proof. In the min-ratio oracle of the SSNC clustering problem, we are given nonnegative node weights $\{\eta_v : v \in V\}$ and subset $X \subseteq W$. The goal is to find:

$$\min_{T \in \mathcal{C}} \frac{\sum_{v \in T} \eta_v}{|T \cap (W \setminus X)|}.$$

We will refer to the nodes $W \setminus X$ as *new* nodes. Our min-ratio oracle involves solving several PNWST instances, as defined below.

[1] Our LLSC definition is slightly different from that in [9] due to the presence of element costs and having to cover only a subset of elements. However, the algorithm and analysis from [9] extend to our formulation in a straightforward way.

For each $\ell = 1, 2, \ldots |W \setminus X|$, we define an instance $\mathcal{I}_\ell$ of PNWST as follows.

- The node-weights are $\{\eta_v : v \in V\}$.
- The rewards are

$$\pi_v = \begin{cases} \frac{1}{\ell} - \frac{d_v}{2Q} & \text{if } v \in W \setminus X, \\ 0 & \text{otherwise} \end{cases} \qquad \forall v \in V.$$

  Note that some node-rewards may be negative. Let $V' := \{v \in V : \pi_v \geq 0\}$ denote the nodes with nonnegative reward. Note that all nodes in $V \setminus V'$ are leaf nodes (by our assumption that all sources are leaf nodes).
- The input graph G' is the subgraph of G induced on nodes V'.
- The target reward is $\tau = \frac{1}{2}$.

Let T_ℓ denote the tree obtained from the $\rho = O(\log n)$ approximation algorithm for the PNWST instance $\mathcal{I}_\ell$. (If instance $\mathcal{I}_\ell$ is infeasible, then $T_\ell = \mathsf{NIL}$ and we skip the following steps.) Let $N \subseteq W \setminus X$ denote the new nodes covered by tree T_ℓ and let $F = \sum_{v \in N} d_v$ be their total demand. So the reward of tree T_ℓ is $\frac{|N|}{\ell} - \frac{F}{2Q} \geq \frac{1}{2}$. In other words,

$$|N| \geq \frac{Q + F}{2Q} \cdot \ell. \tag{2.1}$$

Note that subtree T_ℓ may not be in the set-collection $\mathcal{C}$ as its demand F may be more than Q. To fix this issue, we will select a subset $N' \subseteq N$ of nodes in T_ℓ with demand at most Q and construct a subtree $T'_\ell \in \mathcal{C}$ that contains N'.

If $F \leq Q$, then $N' = N$ and $T'_\ell = T_\ell$. Note that T'_ℓ is in $\mathcal{C}$ as its demand is at most Q. Moreover, $|N'| = |N| \geq \frac{\ell}{2}$ by (2.1). So, the cost-to-coverage ratio of T'_ℓ is at most $2c(T_\ell)/\ell$.

If $F > Q$, we do the following. Starting with a partition of N into singletons (so each part has demand at most Q), we repeatedly merge any two parts into one if the resulting part has total demand at most Q. At the end of this process, we will have $h \leq \frac{2F}{Q}$ parts each with demand at most Q. We set N' to be the largest cardinality part in the final partition. So $|N'| \geq \frac{|N|}{h} \geq \frac{Q}{2F}|N| \geq \frac{\ell}{4}$ by (2.1). Further, let T'_ℓ be the subtree obtained from T_ℓ by removing nodes $N \setminus N'$. Note that T'_ℓ is indeed a tree because the deleted nodes $N \setminus N' \subseteq W$ are all leaves (by our assumption that all sources are leaf nodes). Crucially, T'_ℓ is in $\mathcal{C}$ because its demand is at most Q. Moreover, it covers $|N'| \geq \frac{\ell}{4}$ new nodes. So, the cost-to-coverage ratio of T'_ℓ is at most $4c(T_\ell)/\ell$.

Finally, the min-ratio oracle returns the subtree having the minimum ratio among $\{T'_\ell : 1 \leq \ell \leq |W \setminus X|\}$. We now show that this is a 4ρ-approximation algorithm.

Let $T^* \in \mathcal{C}$ denote the min-ratio cluster and $\ell^* = |T^* \cap (W \setminus X)|$ be the number of new nodes in T^*. Then, by definition of the rewards $\{\pi_v\}$ in PNWST instance $\mathcal{I}_{\ell^*}$, we have $\pi(T^*) \geq \frac{1}{2}$ as the total demand in T^* is at most Q. However, T^* may not itself be feasible to $\mathcal{I}_{\ell^*}$ as it may not be a subtree of graph G' (which is restricted to the nodes V'). Let T' be the subtree of T^* obtained by restricting to the nodes V' of graph G'; note that T' is indeed a tree because all nodes $V \setminus V'$ are leaves. Moreover, the reward of T' is at least that of T^* as nodes in $V \setminus V'$ have negative reward. So, T' is a feasible solution to instance $\mathcal{I}_{\ell^*}$, and the optimal value of $\mathcal{I}_{\ell^*}$ is at most $c(T') \leq c(T^*)$. Hence, $c(T_{\ell^*}) \leq \rho \cdot c(T^*)$ as T_{ℓ^*} is a ρ-approximate solution to $\mathcal{I}_{\ell^*}$. So, the ratio of our algorithm's solution T'_{ℓ^*} is at most $\frac{4\rho}{\ell^*} c(T^*)$. Using $\rho = O(\log n)$ from Theorem 2.6, we obtain the lemma. ∎

Finally, the proof of Lemma 2.4 follows from Theorem 2.5 along with Lemma 2.7.

2.3. Good clustering from SSNC solution. In our multicommodity algorithm, we will utilize approximate solutions to SSNC instances to come up with a good clustering. The desired properties of this clustering are stated in Theorem 2.9 below. Informally, this result says that given any solution to an SSNC instance with some set of sources X and sink t, we can peel out node-disjoint subtrees such that (a) the total demand in any cluster is bounded and (b) each cluster either has at least two sources or contains a neighbor of the sink t. Our clustering algorithm makes use of the following known result on single-sink unsplittable flow.

THEOREM 2.8 (Theorem 3.5 in [23]). *Consider any directed graph with single sink t, sources X having demands $\{d_s : s \in X\}$, and a splittable flow $\mathcal{F}'$ that sends d_s units from each source $s \in X$ to sink t, while respecting node capacities. Then, there is an unsplittable flow $\mathcal{F}$ that routes all demands where the total flow in $\mathcal{F}$ through any node exceeds its original flow (in $\mathcal{F}'$) by at most* $\max_{s \in X} d_s$.

THEOREM 2.9. *Consider any* SSNC *instance with source-nodes X, maximum demand d_{max}, and sink t. Let $N \subseteq V$ denote the neighbors of t. Suppose $V' \subseteq V$ is a solution of cost B such that it supports the demand flow from X to t with maximum node capacity of C. Then, we can find in polynomial time, a node-disjoint collection of rooted subtrees $\{(r_j, T_j)\}_{j=1}^{g}$ such that the following hold:*

1. *Every source node appears in some subtree.*
2. *Each subtree $T_j \subseteq V' \setminus \{t\}$; so the total cost of these subtrees is at most B.*
3. *The total demand in any subtree is at most $C + d_{max}$.*
4. *Every subtree T_j with root $r_j \notin N$ contains at least two sources.*
5. *For every subtree T_j with root $r_j \notin N$ we have $T_j \cap N = \emptyset$.*

Proof. Consider the network induced on the nodes V' (from the SSNC solution) where each node has capacity C. By feasibility of this SSNC solution, there is a splittable flow $\mathcal{F}'$ that sends d_s units of flow from each source $s \in X$ to sink t. As this is a single-sink flow, we can ensure that there are no directed cycles in $\mathcal{F}'$. Moreover, we can assume (without loss of generality) that every neighbor of t (i.e., node in N) sends flow *only* to t: this is because we only have capacities at nodes. Applying Theorem 2.8 to $\mathcal{F}'$, we obtain a flow $\mathcal{F}$ that sends d_s units unsplittably from each source $s \in X$, where the flow through each node (other than t) is at most $C + d_{max}$. Moreover, $\mathcal{F}$ does not have any directed cycles because $\mathcal{F}'$ doesn't. We may also assume (without loss of generality) that in $\mathcal{F}$, each node in N (neighbors of t) only carries nonzero flow to t. Let $E' \subseteq E$ denote the arcs used in flow $\mathcal{F}$; note that (V', E') is a directed acyclic graph. We index the nodes V' in topological sort order with the sink t having the smallest index 1. Let $\mathcal{F}(s)$ be the $s - t$ path used to route demand from source $s \in X$. We construct the desired collection of trees as described in Algorithm 2.1. Let g denote the number of trees produced. We now show that the rooted subtrees $\{(T_j, r_j)\}_{j=1}^{g}$ satisfy the claimed properties.

We first prove that the subtrees T_j are node-disjoint. Note that step 7 in the while-loop only occurs when the sink t is the only node containing flow from more than one source of Y. So, this can only happen in the last iteration. Consider any subtree T_j produced in step 3. By the choice of root r_j, for each $s \in Z_j$ the portion of path $\mathcal{F}(s)$ from s to r_j is disjoint from the paths of the remaining sources $Y \setminus Z_j$. That is, subtree T_j is (node) disjoint from all subtrees $T_{j+1}, \ldots, T_g$ found in later iterations. As noted above, step 7 only occurs in the last iteration, at which point every node in $V' \setminus \{t\}$ carries flow from at most one source of Y. So the subtrees produced in this step are also node-disjoint.

Algorithm 2.1. Computing SSNC Clusters.

initialize sources $Y \leftarrow X$ and $j \leftarrow 1$.
while $Y \neq \emptyset$ **do**
  let root $r_j \in V' \setminus \{t\}$ denote the maximum index node that carries flow from *at least two* sources in Y.
  let $Z_j \subseteq Y$ denote all remaining sources s whose paths $\mathcal{F}(s)$ contain r_j.
  subtree T_j consists of root node r_j, and for each source $s \in Z_j$, the $s - r_j$ prefix of path $\mathcal{F}(s)$.
  update $Y \leftarrow Y \setminus Z_j$ and $j \leftarrow j + 1$.
  if there is no root node (from $V' \setminus \{t\}$) satisfying the condition in step 3 **then**
    for each $u \in N$ (neighbor of t) **do**
      set $r_j = u$ and $Z_j \subseteq Y$ is the singleton set containing the source whose path contains u. ▷ *If there is no such source node then* $Z_j = \emptyset$
      subtree T_j consists of root node r_j, and the $s - r_j$ prefix of path $\mathcal{F}(s)$ for the source $s \in Z_j$.
      update $Y \leftarrow Y \setminus Z_j$ and $j \leftarrow j + 1$.
    end for
  end if
end while

It is clear that each source appears in some subtree, as the while-loop continues until $Y = \emptyset$: this proves property 1. It is also clear than each subtree $T_j \subseteq V' \setminus \{t\}$: combined with node-disjointness of the subtrees, we obtain property 2.

We now bound the total demand in each subtree T_j. For any tree T_j produced in step 3, we have $r_j \in \mathcal{F}(s)$ for all $s \in Z_j$. So the flow through node r_j in the unsplittable flow $\mathcal{F}$ is at least $\sum_{s \in Z_j} d(s)$, the total demand in T_j. Using the fact that $\mathcal{F}$ sends at most $C + d_{max}$ flow through any node (other than t), the total demand in T_j is at most $C + d_{max}$. As noted above, step 7 only occurs when every node in $V' \setminus \{t\}$ carries flow from at most one source of Y. So each subtree produced in step 7 contains at most one source, which has demand at most d_{max}. This proves property 3.

Each subtree produced in step 3 contains at least two sources: this follows from the choice of node r_j. All remaining subtrees (produced in step 7) have as their root some node of N (neighbors of t). This proves property 4.

For property 5, consider any subtree T_j with root $r_j \notin N$. Clearly, T_j must be produced in step 3. Moreover, T_j consists of the prefixes of certain paths until node r_j. As nodes of N only send flow to sink t and the root $r_j \notin N$, subtree T_j does not contain any node of N. ∎

3. Multicommodity node-capacitated network design. We now discuss the general multicommodity case of the problem. Recall that the input is an undirected graph $G = (V, E)$ with k request-pairs $\{(s_i, t_i, d_i) \mid i \in [k]\}$, where the ith request has source s_i, sink t_i, and demand $1 \leq d_i \leq q$. All nodes have capacity q. The output is a subset of nodes $V' \subseteq V$ such that the graph $G[V']$ induced by V' can simultaneously support d_i units of flow (unsplittably) between nodes s_i and t_i, for each $i \in [k]$. The objective is to minimize the total cost $c(V') = \sum_{v \in V'} c_v$. As mentioned earlier, we assume (without loss of generality) that all terminals are distinct and each terminal is a leaf node. For any terminal s, we define its *mate* to be the unique terminal t such that (s, t) is a request-pair. We also use $d(s)$ to denote the demand associated with any terminal s; so we have $d(s_i) = d(t_i) = d_i$ for all $i \in [k]$.

Roadmap. Our algorithm first clusters the terminals into nearly node-disjoint subtrees of low total cost. We need a new notion of "allowed clusters" in the multicommodity case, and the clustering algorithm is based on iteratively solving several instances of the single-sink problem (SSNC). The details appear in section 3.1. Next, the algorithm routes demands across different clusters while respecting node capacities. The routing algorithm relies on random-sampling and cut-sparsification; see section 3.2 for details. After combining the intercluster routing with the clusters themselves, we are able to route a *constant fraction* of the demands with small node congestion. Finally, we need to apply the above clustering and routing algorithms recursively on all unsatisfied demands, so we repeat the main algorithm a logarithmic number of times.

3.1. Clustering. Here, we describe the multicommodity clustering algorithm that finds a collection of nearly disjoint clusters, where each cluster has either a large fraction of "induced" demands (*internal* clusters) or a large number of "crossing" demands (*heavy* clusters). During our clustering algorithm, we will drop some request-pairs and maintain a *current* set of requests K. At any point in the algorithm, the terminals are the sources/sinks of *only* the current request-pairs. We will ensure that requests remaining at the end of the clustering algorithm have a constant fraction of the total demand $D := \sum_{i=1}^{k} d_i$.

DEFINITION 3.1 (MCNC cluster). *Let $K \subseteq [k]$ denote a subset of requests. The following definitions are relative to K, where the nodes $\{s_i, t_i\}_{i \in K}$ are called terminals. A cluster is any subtree T in graph G.*

- *The set of terminals contained in cluster T is denoted* $\mathsf{terms}(T)$.
- *The demand of cluster T is* $\mathsf{load}(T) = \sum_{s \in \mathsf{terms}(T)} d(s)$, *the sum of demands over all its terminals.*
- *A terminal $s \in \mathsf{terms}(T)$ is called* internal *if its mate is also in* $\mathsf{terms}(T)$*; the terminal s is called* external *otherwise.*
- *The internal (resp., external) demand of cluster T is the total demand of its internal (resp., external) terminals.*

Note that "internal requests" (with both source and sink in T) contribute twice to $\mathsf{load}(T)$, whereas "external requests" (with exactly one terminal in T) contribute just once to $\mathsf{load}(T)$.

DEFINITION 3.2 (cluster categories). *Let $K \subseteq [k]$ be a subset of requests and T be any cluster. Then, T is said to be*

- heavy *if its demand* $\mathsf{load}(T)$ *is at least q;*
- internal *if its internal demand is more than* $\mathsf{load}(T)/2$*;*
- active *if it is neither internal nor heavy.*

We note that some clusters may be both internal and heavy. In our algorithm, we explicitly maintain collections of different cluster types, and any ties will be broken according to the algorithm.

We will maintain and grow active clusters until all clusters are heavy or internal. We further classify active clusters into two types depending on how much of their external demand goes to other active clusters. This distinction is important because the algorithm needs to deal with these clusters differently.

DEFINITION 3.3 (active cluster types). *Let $K \subseteq [k]$ be a subset of requests and T an active cluster. T is a* type 1 *active cluster if the total demand of terminals in T with their mates in other active clusters is less than* $\mathsf{load}(T)/4$. *Otherwise, T is a*

type 2 *active cluster. Moreover, a type* 1 *active cluster is called* dangerous *if it has nonzero demand going to other active clusters.*

We will refer to type 1 and type 2 active clusters as t1-active and t2-active, respectively. Note that the total external demand of any active cluster T is at least $\mathsf{load}(T)/2$; otherwise, T would be an internal cluster (not active). So, any t1-active cluster has at least $\mathsf{load}(T)/4$ demand crossing to internal/heavy clusters. Moreover, any t1-active cluster that is not dangerous has *all* its external demands going to internal/heavy clusters.

We can now state the main multicommodity clustering result.

THEOREM 3.4. *Suppose that there is a* (β, γ)*-bicriteria approximation algorithm for the single-sink problem (*SSNC*). Then, for any* MCNC *instance with optimal cost* Opt*, there is a polynomial-time algorithm that finds a subset* $K \subseteq [k]$ *of request-pairs and a collection* $\widehat{\mathcal{T}}$ *of clusters (relative to* K*) such that the following hold:*

(i) *Each cluster in* $\widehat{\mathcal{T}}$ *is internal or heavy.*
(ii) *Each terminal (i.e., node in* $\{s_i, t_i\}_{i \in K}$*) lies in exactly one cluster.*
(iii) *The total demand of request-pairs* K *is* $\sum_{i \in K} d_i \geq D/4$.
(iv) *Each node appears in at most* $O(\log k)$ *different clusters.*
(v) *The demand of each cluster is at most* $O(\gamma^2 \log k) \cdot q$.
(vi) *The total cost of all the clusters*

$$\sum_{T \in \widehat{\mathcal{T}}} \sum_{v \in T} c_v \leq O(\beta \cdot \log k) \cdot \mathsf{Opt}.$$

Overview of algorithm/analysis. We start with each terminal being its own cluster. The clustering algorithm aims to find clusters with $\approx q$ terminals in each; these aggregated demands can then be handled in the routing step of our algorithm. This motivates the definition of *heavy* clusters, which contain at least q demand (see Definition 3.2). In order to obtain such heavy clusters, the algorithm iteratively merges the nonheavy clusters. Moreover, to ensure that there is a low-cost solution to this "merging" step, we need each cluster to have a constant fraction of its demand going to *other* clusters: otherwise, the cost to merge may be much more than the optimal MCNC cost (denoted by Opt). This motivates the definition of *internal* clusters, where a constant fraction of the demand is induced inside the cluster (see Definition 3.2). While internal clusters cannot participate in merging anymore, we can use the subtree inside such clusters to route all its internal demand (which is a constant fraction of its total demand). So, both heavy and internal clusters are "good" in the sense that we will be able to satisfy a constant fraction of their demand in the subsequent routing step. Therefore, the revised aim of the clustering algorithm is to find a collection of heavy *or* internal clusters. All other clusters are called *active,* which the algorithm continues to merge.

The clustering algorithm proceeds in iterations. Each iteration attempts to merge active clusters and ensures that the number of active clusters reduces by a constant factor. So, the number of iterations will be at most $O(\log k)$. The merging step in each iteration is based on solving suitable instances of the *single-sink* problem SSNC. We will ensure that the optimal value of each SSNC instance is at most Opt, so the final cost of our clusters will be $O(\log k) \cdot \mathsf{Opt}$. When multiple active clusters merge together in any iteration, the resulting cluster may be internal, heavy, or active (we make progress in all cases). However, the new cluster may have demand much more than q, as we cannot control how many active clusters get merged into one. In order to handle this issue, we ensure that the SSNC instances have node-capacity

$\approx q$; the SSNC approximation guarantee then implies that the demand in any new cluster is at most $\approx \gamma \cdot q$. (In some cases we will have slightly larger node capacities in SSNC, as explained later.)

Another issue to handle is that we may be unable to merge active clusters just with each other. In particular, it is possible that a large fraction of an active cluster's demand goes to heavy/internal clusters. Then, we cannot expect to merge such a cluster with other active clusters. This motivates the classification of active clusters into types 1 and 2, corresponding to an active cluster having a low/high fraction of its demand go to other active clusters (see Definition 3.3). The two types of active clusters are dealt with separately. Initially, each singleton cluster is t2-active. Assume for now that there are no requests between active clusters of different types. We will explicitly ensure this property by *dropping some requests* and preserving only a subset $K \subseteq [k]$ of requests.

- *Merging t2-active clusters.* Intuitively, these active clusters can be merged with each other (at low cost) because most of their demands go to other active clusters. We would like to construct an SSNC instance $\mathcal{I}_2$, where each t2-active cluster is a source. However, the original MCNC requests are multicommodity; so, even requests associated with one cluster do not have a common sink. We get around this issue by restricting our attention to a "bipartite demand graph" containing at least half the total demand between t2-active clusters. This corresponds to finding an appropriate bi-partition $(\mathcal{A}^+, \mathcal{A}^-)$ of t2-active clusters, which can be done using a simple local search. Now, we treat all clusters in part $\mathcal{A}^+$ as sources and connect all clusters in $\mathcal{A}^-$ to a new sink node t. All nodes have capacity $\approx q$. See Algorithm 3.3 for the formal description. In the analysis, we need to show that the optimal cost of this SSNC instance is at most Opt. This is done by demonstrating a (fractional) routing based on the optimal MCNC solution and using the fact that at least half the total demand in t2-active clusters is "crossing" between $\mathcal{A}^+$ and $\mathcal{A}^-$. Finally, we obtain a collection of subtrees (using the approximate SSNC solution and Theorem 2.9) that are used to merge t2-active clusters.
- *Merging t1-active clusters.* These clusters have a constant fraction of their demands going to heavy/internal clusters. If any requests from a t1-active cluster go to an internal cluster, then we simply drop these requests and "charge" them to the requests in internal clusters which will definitely be preserved. Note that demand induced in any internal cluster is at least a constant fraction of its total demand. Moreover, the clustering algorithm only aims to preserve a subset K of requests (which should be a constant fraction of the total demand). So, we are left with the case that a large fraction of demand from any t1-active cluster goes to heavy clusters. Intuitively, these clusters can be merged with heavy clusters at low cost. We now construct an SSNC instance $\mathcal{I}_1$ to merge t1-active clusters with heavy clusters (or each other). We treat each t1-active cluster as a source and connect all heavy clusters to a new sink t. All nodes have capacity $\approx q$ except the nodes corresponding to heavy clusters, which have capacity $O(\gamma \cdot q)$. The reason that we have a larger capacity for heavy clusters is that the demand in a heavy cluster can be as large as $O(\gamma \cdot q)$ and the MCNC routing (which is used to demonstrate a low-cost SSNC routing) induces a corresponding load on these clusters. See Algorithm 3.2 for the formal description. Again, we merge clusters using the approximate SSNC solution and Theorem 2.9.

The demand of any new cluster formed when active clusters merge with each other is bounded by $O(\gamma \cdot q)$ as all nodes in these clusters have capacity $O(q)$ and γ is the capacity violation in our SSNC algorithm. However, when t1-active clusters merge with an existing heavy cluster H, the demand of H may increase by as much as $O(\gamma^2 \cdot q)$; this is because nodes corresponding to heavy clusters have capacity $O(\gamma \cdot q)$. Unfortunately, this increased demand in H may multiply over iterations. Specifically, if most of the new requests in H are going to other active clusters, then the SSNC instance $\mathcal{I}_1$ in the next iteration must increase the capacity of H from $\gamma \cdot q$ to $\gamma^2 \cdot q$; this is needed to demonstrate a low cost solution to $\mathcal{I}_1$. So, the capacity (and hence the demand) of cluster H will keep increasing by a multiplicative factor γ in each iteration! In order to fix this issue, we will ensure that all t1-active clusters in SSNC instance $\mathcal{I}_1$ have *zero* demand going to other active clusters (again, this property will be ensured by dropping certain requests). This motivates the definition of *dangerous* clusters, which are t1-active clusters having any request going to other active clusters (Definition 3.3). We will ensure that instance $\mathcal{I}_1$ has no dangerous clusters. Now, when t1-active clusters merge with a heavy cluster H, the demand between H and active clusters *does not* increase. So, the capacity of H in instance $\mathcal{I}_1$ of the next iteration can remain $O(\gamma \cdot q)$. We note that the total demand in H still increases *additively* by $O(\gamma^2 \cdot q)$ in each iteration. As there are only $O(\log k)$ iterations, the final demand of any heavy cluster can be bounded by $O(\log k \cdot \gamma^2 \cdot q)$.

Dropping requests. In the above description, we assumed that there are no requests between (i) active clusters of different types, (ii) internal clusters and active clusters, and (iii) t1-active clusters and t1- or t2-active clusters. As mentioned earlier, to ensure these properties, our algorithm drops certain requests and preserves only a subset $K \subseteq [k]$. Specifically, in each iteration we perform a "pruning step" after merging active clusters. This involves repeatedly choosing a (new) cluster T that is either internal or dangerous, and dropping all requests between T and other active clusters. In the analysis we will show that the dropped requests can be "charged" to preserved requests in K, so that the final set K is a constant fraction of the total demand in MCNC. No request incident to a heavy cluster is ever dropped, so heavy clusters do not change in this pruning step. We note that dropping requests incident to cluster T may cause another active cluster T' to become internal or dangerous; then, cluster T' will also be processed later in this pruning step. Therefore, at the end of each iteration, we ensure the three properties (i)–(iii). We emphasize that the cluster definitions are all relative to the current set K of requests.

Algorithm 3.1 describes the overall clustering algorithm MCcluster. We maintain separate collections of clusters: $\mathcal{T}_i$ (internal), $\mathcal{T}_h$ (heavy), $\mathcal{A}_1$ (t1-active), and $\mathcal{A}_2$ (t2-active). Algorithm 3.2 (MergeT1) and Algorithm 3.3 (MergeT2) describe the separate procedures to merge t1-active and t2-active clusters. Figures 1 and 2 illustrate the graphs G_1 and G_2 used in the two SSNC instances $\mathcal{I}_1$ (solved in MergeT1) and $\mathcal{I}_2$ (solved in MergeT2).

Analysis. Our first lemma shows that all clusters are classified correctly (relative to the current requests K) at the end of each iteration of MCcluster. During the iteration, new or modified clusters may be in the wrong collection, but these will get fixed at the end (as shown in the next lemma).

LEMMA 3.5. *At the end of each iteration of Algorithm* MCcluster*, the clusters in the current set of clusters are correctly classified into $\mathcal{T}_i$ (internal), $\mathcal{T}_h$ (heavy), $\mathcal{A}_1$ (t1-active), and $\mathcal{A}_2$ (t2-active).*

Algorithm 3.1. MCNC Clustering Algorithm MCcluster.

1: initialize $K = [k]$ to be all request-pairs and the clusters are all singletons.
2: all clusters are t2-active, i.e., $\mathcal{A}_2 \leftarrow \{\{s_i\}, \{t_i\}\}_{i \in K}$, $\mathcal{T}_i \leftarrow \emptyset$, $\mathcal{T}_h \leftarrow \emptyset$, $\mathcal{A}_1 \leftarrow \emptyset$.
3: **while** some cluster is active ($\mathcal{A}_1 \cup \mathcal{A}_2 \neq \emptyset$) **do** ▷ *Iteration begins*
4: run algorithm MergeT1($\mathcal{T}_h, \mathcal{A}_1$) which modifies t1-active and heavy clusters.
5: run algorithm MergeT2($\mathcal{A}_2$) which modifies t2-active clusters.
6: **for** all clusters T in $\mathcal{A}_1 \cup \mathcal{A}_2$ **do** ▷ *Identify heavy clusters*
7: if T is heavy then move it from $\mathcal{A}_1$ or $\mathcal{A}_2$ to $\mathcal{T}_h$.
8: **end for**
9: **while** some cluster T in $\mathcal{A}_1 \cup \mathcal{A}_2$ is dangerous or internal **do**
▷ *Identify internal clusters and ensure no dangerous clusters*
10: remove from K all requests between T and other active clusters.
11: if T is internal (resp., t1-active), then move it to $\mathcal{T}_i$ (resp., $\mathcal{A}_1$).
12: **end while**
13: if any cluster $T \in \mathcal{A}_2$ is t1-active, then move it to $\mathcal{A}_1$.
14: **end while**

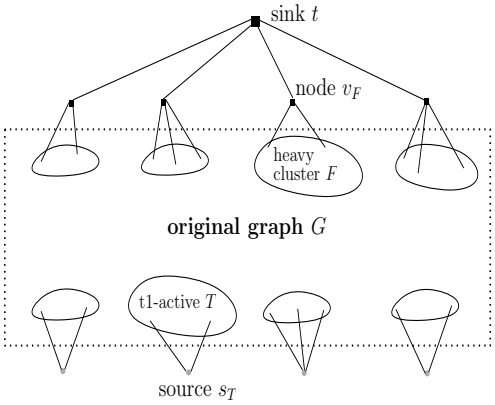

FIG. 1. *Graph G_1 for t1-active clusters.*

Proof. At the beginning of the algorithm, each cluster is a singleton terminal. Clearly, these are t2-active clusters, so the initial classification is correct.

Algorithm MCcluster first runs MergeT1 and MergeT2 to merge t1-active and t2-active clusters separately. MergeT2 creates new clusters by merging t2-active clusters, and places all new clusters in $\mathcal{A}_2$. MergeT1 modifies existing heavy clusters (which remain in $\mathcal{T}_h$) and creates new clusters by merging t1-active clusters, which are placed in $\mathcal{A}_1$. Then, in Step 7 of MCcluster, we first identify any new heavy clusters (in $\mathcal{A}_1 \cup \mathcal{A}_2$) and move them to $\mathcal{T}_h$. At this point, all heavy clusters are classified correctly. Moreover, MCcluster never removes requests incident to a heavy cluster (see Steps 9–12). So, once a cluster is heavy, it will remain heavy throughout the algorithm.

Next, in Steps 9–12 of MCcluster, we repeatedly identify dangerous/internal clusters and drop some requests (which in turn can modify other clusters). We now argue that the clusters are classified correctly after this step:

- Suppose an internal cluster $T \in \mathcal{A}_1 \cup \mathcal{A}_2$ is processed in Step 9. We remove all requests from T to other active clusters. This keeps T as an internal cluster,

Algorithm 3.2. Merging algorithm for t1-active clusters $\mathsf{MergeT1}(\mathcal{T}_h, \mathcal{A}_1)$.

let graph G_1 consist of graph G and the following new nodes/edges:
- For each cluster $T \in \mathcal{A}_1$, there is a source node s_T (of zero cost) with demand $d(s_T) = \mathsf{load}(T)$; node s_T is connected to each terminal in T.
- For each heavy cluster $F \in \mathcal{T}_h$, there is a new node v_F (of zero cost), which is connected to every terminal in F.
- There is a new sink node t, connected to the nodes $\{v_F : F \in \mathcal{T}_h\}$.

The node capacities are $\tilde{q} := 5q$. For the zero-cost nodes v_F corresponding to heavy clusters $F \in \mathcal{T}_h$, we set their capacities to $9\gamma \cdot \tilde{q}$, where γ is the approximation ratio for node-congestion from our single-sink algorithm.
let $\mathcal{I}_1$ be the SSNC instance on graph G_1, with sources $\{s_T\}_{T \in \mathcal{A}_1}$ and sink t.
solve instance $\mathcal{I}_1$ using the SSNC approximation algorithm (Theorem 1.2) to obtain solution $V_1 \subseteq V$.
using Theorem 2.9 on solution V_1, obtain a collection of rooted subtrees $\mathcal{N}_1$.
for each subtree X (with root r) in $\mathcal{N}_1$ **do** ▷ *Merge clusters using* $\mathcal{N}_1$
  let $\mathcal{X} = \{T \in \mathcal{A}_1 : s_T \in X\}$ be the t1-active clusters whose source-nodes are contained in subtree X
  update $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \setminus \mathcal{X}$
  if root r corresponds to a heavy cluster $F \in \mathcal{T}_h$ (i.e., $r = v_F$) **then**
    add clusters $\mathcal{X}$ and subtree X to heavy cluster F, i.e., $F \leftarrow F \cup X \cup \mathcal{X}$
  else
    merge clusters $\mathcal{X}$ using subtree X to get new cluster $S = X \cup \mathcal{X}$
    update $\mathcal{A}_1 \leftarrow \mathcal{A}_1 \cup \{S\}$
      ▷ *S may not be t1-active: its type will be updated in* MCcluster
  end if
end for

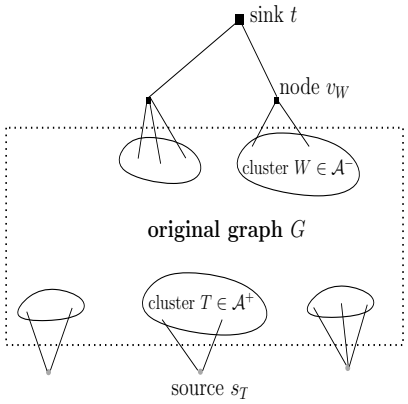

FIG. 2. *Graph G_2 for t2-active clusters.*

and it is classified correctly. Moreover, we never remove any internal requests, and we never modify an internal cluster after it is assigned to $\mathcal{T}_i$.

- Suppose a dangerous cluster $T \in \mathcal{A}_1 \cup \mathcal{A}_2$ is processed in Step 9. We remove all requests from T to other active clusters. As a result, T will become either internal (if its internal demand is more than $\mathsf{load}(T)/2$) or nondangerous

Algorithm 3.3. Merging algorithm for t2-active clusters MergeT2($\mathcal{A}_2$).

1: partition clusters $\mathcal{A}_2$ into $\mathcal{A}^+$ and $\mathcal{A}^-$ such that every cluster in $\mathcal{A}^+$ (resp., $\mathcal{A}^-$) has more demand going to clusters in $\mathcal{A}^-$ (resp., $\mathcal{A}^+$) than $\mathcal{A}^+$ (resp., $\mathcal{A}^-$)
▷ *This can be done by a simple local search*
2: relabel the parts so that $|\mathcal{A}^+| \geq |\mathcal{A}^-|$.
3: let graph G_2 consist of graph G and the following new nodes/edges:
- For each cluster $T \in \mathcal{A}^+$, there is a source node s_T (of zero cost) with demand $d(s_T) = \mathsf{load}(T)$; node s_T is connected to each terminal of T.
- For each cluster $W \in \mathcal{A}^-$, there is a new node v_W (of zero cost), connected to every terminal of W.
- There is a new sink node t, connected to the nodes $\{v_W : W \in \mathcal{A}^-\}$.

4: The node capacities are $q' = 9q$.
5: let $\mathcal{I}_2$ be the SSNC instance on graph G_2, with sources $\{s_T\}_{T \in \mathcal{A}^+}$ and sink t.
6: solve instance $\mathcal{I}_2$ using the SSNC approximation algorithm (Theorem 1.2) to obtain solution $V_2 \subseteq V$.
7: using Theorem 2.9 on solution V_2, obtain a collection of rooted subtrees $\mathcal{N}_2$.
8: **for** each subtree Y in $\mathcal{N}_2$ **do** ▷ *Merge clusters using* $\mathcal{N}_2$
9: let $\mathcal{Y} = \{T \in \mathcal{A}^+ : s_T \in Y\} \bigcup \{W \in \mathcal{A}^- : v_W \in Y\}$ be the t2-active clusters whose s-nodes or v-nodes are contained in subtree Y
10: update $\mathcal{A}_2 \leftarrow \mathcal{A}_2 \setminus \mathcal{Y}$.
11: merge clusters $\mathcal{Y}$ with each other to get new cluster $S = Y \cup \mathcal{Y}$
12: update $\mathcal{A}_2 \leftarrow \mathcal{A}_2 \cup \{S\}$ ▷ S's *type will be updated in* MCcluster
13: **end for**

t1-active (if its internal demand is at most $\mathsf{load}(T)/2$); note that T cannot become heavy or t2-active. In either case, T is classified correctly, and it will not be modified later in this iteration.

- Consider now any cluster $T' \in \mathcal{A}_1$ that is *not* processed in Step 9. Then, T' is not heavy or internal (or even dangerous). So it must be active. Also, it must have formed in algorithm MergeT1 as a result of merging some t1-active clusters, and such a cluster cannot be t2-active; its demand going to other active clusters will be less than $\mathsf{load}(T')/4$. Further, when requests are dropped, a t1-active cluster cannot become t2-active because the only requests dropped from T' are those going to other active clusters. So T' remains t1-active at the end of this iteration.
- Consider now any cluster $T' \in \mathcal{A}_2$ that is *not* processed in Step 9. Again, T' must be active and nondangerous. It could be either t1-active or t2-active and is classified correctly in Step 13. ∎

We now observe how cluster types change during algorithm MCcluster. The classification of a cluster (its assignment to $\mathcal{T}_h$, $\mathcal{T}_i$, $\mathcal{A}_1$, or $\mathcal{A}_2$) changes due to merging the cluster with other clusters (which occurs in algorithms MergeT1 and MergeT2), or removal of requests from K (which occurs in algorithm MCcluster). We note that these changes satisfy following:

- Any cluster in $\mathcal{T}_i$ will remain in $\mathcal{T}_i$.
- Any cluster in $\mathcal{T}_h$ will be (part of) some cluster of $\mathcal{T}_h$.
- Any cluster in $\mathcal{A}_1$ will be (part of) some cluster of $\mathcal{A}_1 \cup \mathcal{T}_i \cup \mathcal{T}_h$.
- Any cluster in $\mathcal{A}_2$ will be (part of) some cluster of $\mathcal{A}_2 \cup \mathcal{A}_1 \cup \mathcal{T}_i \cup \mathcal{T}_h$.

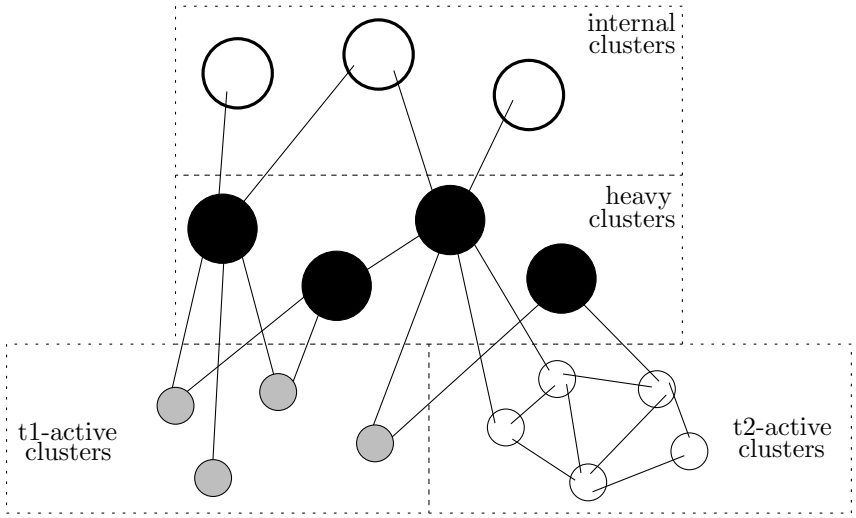

FIG. 3. *Different types of clusters at the end of an* MCcluster *iteration. The edges represent request-pairs (in K) across clusters.*

Next, we show that requests (in the current set K) between different kinds of clusters satisfy some useful properties. These properties are crucial in setting up the SSNC instances correctly and ensuring that the demand of heavy clusters does not grow too much. Figure 3 illustrates the possible requests across clusters.

LEMMA 3.6. *At the start/end of any iteration of algorithm* MCcluster*, there are no requests (in K) between*

- *internal clusters $\mathcal{T}_i$ and active clusters $\mathcal{A}_1 \cup \mathcal{A}_2$;*
- *any t1-active cluster $T \in \mathcal{A}_1$ and other active clusters $\mathcal{A}_1 \cup \mathcal{A}_2 \setminus \{T\}$.*

Moreover, the demand from any heavy cluster to active clusters is at most $10\gamma \cdot q$.

Proof. We will prove this inductively over the iterations. The lemma is clearly true at the beginning of MCcluster (all clusters are t2-active). Now consider any iteration of MCcluster: assuming the lemma at the start of the iteration, we prove that it also holds at the end.

For the first property, observe that MCcluster does not modify any existing internal cluster, so any cluster that was internal at the start of the iteration continues to satisfy this property. Let I be a (new) cluster that is found to be internal in this iteration. This must happen in Steps 9–12, and we explicitly remove all requests from I to other active clusters at this point.

For the second property, let $T \in \mathcal{A}_1$ be any t1-active cluster at the end of the iteration. By the loop condition in Steps 9–12 of MCcluster, T cannot be dangerous. So, the demand from T to other active clusters is zero, as desired.

For the third property, we consider two cases for any heavy cluster $H \in \mathcal{T}_h$:

- *H is a new heavy cluster.* Then, H must have formed due to merging t1-active (resp., t2-active) clusters in algorithm MergeT1 (resp., MergeT2). See Step 12 in MergeT1 and Step 11 in MergeT2. If H was formed in MergeT1, then it was based on SSNC instance $\mathcal{I}_1$, where node capacities are $\tilde{q} = 5q$. Using Theorems 1.2 and 2.9, it follows that the demand of cluster H must be at most $\gamma \cdot \tilde{q} + \tilde{q} \leq 10\gamma \cdot q$. If H was formed in MergeT2, the analysis is

similar—this time, using SSNC instance $\mathcal{I}_2$, which has node capacity $q' = 9q$. Using Theorems 1.2 and 2.9 again, the demand of cluster H is at most $\gamma \cdot q' + q' = (9\gamma+9) \cdot q \leq 10\gamma \cdot q$. Here, we used $\gamma \geq 9$ because γ is polylogarithmic. In either case, the total demand in cluster H is at most $10\gamma \cdot q$, which also bounds the demand from H to active clusters.

- *H is an existing heavy cluster.* Then, H will be modified in algorithm MergeT1 based on SSNC instance $\mathcal{I}_1$. Here, only t1-active clusters get added to cluster H; see Step 10 in MergeT1. Crucially, there is *zero* demand from any t1-active cluster to other active clusters (by the second property in this lemma). So the demand from cluster H to active clusters does not increase, and it remains at most $10\gamma \cdot q$. ∎

We are now ready to bound the optimal values of the SSNC instances $\mathcal{I}_1$ and $\mathcal{I}_1$. We do this by using the optimal MCNC solution and properties of the different cluster types (shown above).

LEMMA 3.7. *The optimal cost of* SSNC *instance* $\mathcal{I}_1$ *(in algorithm* MergeT1*) is at most the optimal cost* Opt *of the original* MCNC *instance.*

Proof. We first exhibit a feasible *fractional* flow using the optimal solution of the multicommodity MCNC instance. Let $V^* \subseteq V$ be the nodes used in Opt. For each request $i \in [k]$, let P_i^* denote the path from s_i to t_i in Opt. Note that $P_i^* \subseteq V^*$ for all $i \in [k]$, and $\sum_{i \in [k]: P_i^* \ni v} d_i \leq q$ for all nodes $v \in V^*$.

We will use $V^* \cup \{s_T : T \in \mathcal{A}_1\} \cup \{v_F : F \in \mathcal{T}_h\} \cup \{t\}$ as the nodes in our SSNC solution. The cost of this solution is Opt as the new nodes have zero cost. We now show how to route all the demands fractionally using these nodes.

Consider any t1-active cluster $T \in \mathcal{A}_1$. We first claim that the total demand of terminals in T having mates in heavy clusters is at least $\mathsf{load}(T)/4$. Indeed, by definition of t1-active clusters, the total demand from T to internal/heavy clusters is at least $\mathsf{load}(T)/4$. Moreover, by Lemma 3.6, there is zero demand from T to internal clusters, so the demand from T to heavy clusters $\mathcal{T}_h$ is at least $\mathsf{load}(T)/4$. We now route demand from s_T to t as follows. For each external terminal $s_i \in T$ (with its mate $t_i \in F$ for some heavy cluster F), send 4 units of flow from s_T to s_i, then from s_i to t_i along path P_i^*, then from t_i to v_F, and finally from v_F to sink t. Note that these are valid paths in graph G_1 of instance $\mathcal{I}_1$. Also, the total demand routed from s_T to t is at least $\mathsf{load}(T) = d(s_T)$, as desired.

Routing as above for each cluster $T \in \mathcal{A}_1$, we get a fractional routing that satisfies all the demand in the SSNC instance $\mathcal{I}_1$. We now argue that the load on any node is at most its capacity. Clearly, the flow through each node $v \in V^*$ is at most $4 \cdot \sum_{i \in [k]: P_i^* \ni v} d_i \leq 4q$. Moreover, the flow through each node v_F (for $F \in \mathcal{T}_h$) is at most 4 times the total demand between F and all active clusters, which is at most $40\gamma q$ by Lemma 3.6.

Let $d_{max} = \max_{T \in \mathcal{A}_1} \mathsf{load}(T)$ denote the maximum demand in $\mathcal{I}_1$; note that $d_{max} \leq q$. Now, we convert the above fractional flow into an unsplittable flow as required in SSNC. To this end, we use Theorem 2.8 on our fractional routing for $\mathcal{I}_1$. We then obtain an *unsplittable flow* for $\mathcal{I}_1$, where (i) the flow through each node of V^* is at most $4q + d_{max} \leq 5q = \tilde{q}$, and (ii) the flow through each node v_F is at most $40\gamma q + d_{max} \leq 9\gamma\tilde{q}$. The lemma now follows. ∎

LEMMA 3.8. *The optimal cost of* SSNC *instance* $\mathcal{I}_2$ *(in algorithm* MergeT2*) is at most the optimal cost* Opt *of the original* MCNC *instance.*

Proof. The high-level proof is similar to that of Lemma 3.7. We first show a feasible *fractional* routing using the MCNC optimal solution and then convert it into an unsplittable flow. Let $V^* \subseteq V$ be the nodes used in Opt. For each request $i \in [k]$, let P_i^* denote the path from s_i to t_i in Opt. Again, $P_i^* \subseteq V^*$ for all $i \in [k]$, and $\sum_{i \in [k]: P_i^* \ni v} d_i \leq q$ for all nodes $v \in V^*$.

We refer to the clusters in $\mathcal{A}^+$ as source clusters as they correspond to source nodes in the SSNC instance $\mathcal{I}_2$. We also refer to the clusters in $\mathcal{A}^-$ as sink clusters as they are directly connected to the sink t. Note that all these clusters are t2-active.

Consider any source cluster $T \in \mathcal{A}^+$. We first show that the total demand from T to sink clusters $\mathcal{A}^-$ is at least $\mathsf{load}(T)/8$. By definition of t2-active clusters, the total demand from T to other active clusters is at least $\mathsf{load}(T)/4$. Moreover, by Lemma 3.6, there are no requests between T and t1-active clusters. So, the total demand from T to other t2-active clusters is at least $\mathsf{load}(T)/4$. Further, by choice of the partition $(\mathcal{A}^+, \mathcal{A}^-)$ in Step 1 of MergeT2, the total demand from any source cluster T to all sink clusters is at least half the total demand from T to t2-active clusters. So, there is demand at least $\mathsf{load}(T)/8$ from T to $\mathcal{A}^-$.

Now, for any source cluster T, let C_T denote the set of all requests between T and sink clusters. We route demand from s_T to t as follows. For each $i \in C_T$, send 8 units from s_T to s_i, then from s_i to t_i along path P_i^*, then from t_i to v_W (where t_i lies in sink cluster $W \in \mathcal{A}^-$), and finally from v_W to t. Note that these are valid paths in graph G_2 of instance $\mathcal{I}_2$. Moreover, the net flow out of each source s_T is at least $\mathsf{load}(T)$ as desired.

Performing the above routing for all source clusters $T \in \mathcal{A}^+$, we obtain a fractional flow that satisfies all demands in $\mathcal{I}_2$. We now argue that the node capacity constraints are satisfied. Clearly, the flow through each node $v \in V^*$ is at most $8 \cdot \sum_{i \in [k]: P_i^* \ni v} d_i \leq 8q$. Moreover, the flow through each node v_W is at most 8 times the total external demand in cluster W, which is at most $8 \cdot q$ because W is an active cluster. Now, applying Theorem 2.8, we obtain an unsplittable flow for $\mathcal{I}_2$, where the flow through each node is at most $8q + \max_{T \in \mathcal{A}^+} \mathsf{load}(T) \leq 9q = q'$. This completes the proof. ∎

We now summarize some key properties of the (partial) solution built in each iteration of algorithm MCcluster.

LEMMA 3.9. *In any iteration of* MCcluster*, we have the following:*

1. *The subtrees added to clusters are $\mathcal{N}_1 \cup \mathcal{N}_2$, where $\mathcal{N}_1$ and $\mathcal{N}_2$ are found in* MergeT1 *and* MergeT2*. Each node appears in at most two of these subtrees.*
2. *The cost of the new nodes added to clusters is at most $2\beta \cdot$ Opt.*
3. *The demand of any new internal/heavy cluster is at most $10\gamma \cdot q$.*
4. *The demand of any existing heavy cluster increases by at most $54\gamma^2 \cdot q$.*
5. *The number of active clusters at the end of the iteration is at most $\frac{3}{4}$ times the number at the start of the iteration.*

Proof. We prove the claimed properties one by one.

Property 1. In any iteration, subtrees are added to clusters in both MergeT1 and MergeT2. The subtrees added in MergeT1 are exactly those in $\mathcal{N}_1$, which is the collection obtained by applying Theorem 2.9 to the SSNC solution V_1. Similarly, the subtrees added in MergeT2 are exactly those in $\mathcal{N}_2$. By Theorem 2.9, all subtrees in $\mathcal{N}_1$ (resp., $\mathcal{N}_2$) are node-disjoint. Hence, each node appears at most twice in $\mathcal{N}_1 \cup \mathcal{N}_2$.

Property 2. By Theorem 2.9 (property 2), the total cost of subtrees in $\mathcal{N}_1$ is at most $c(V_1)$. Moreover, by our SSNC approximation guarantee, $c(V_1)$ is at most

β times the optimal value of instance $\mathcal{I}_1$. Using Lemma 3.7, it now follows that $c(\mathcal{N}_1) \leq c(V_1) \leq \beta \cdot \mathsf{Opt}$. Similarly, we obtain $c(\mathcal{N}_2) \leq c(V_2) \leq \beta \cdot \mathsf{Opt}$ using Lemma 3.8 (for SSNC instance $\mathcal{I}_2$). Hence, the total cost of the new nodes added to clusters is at most $2\beta \cdot \mathsf{Opt}$.

Property 3. In any iteration, new clusters may be formed in either MergeT1 or MergeT2. Consider any new cluster formed in MergeT2; see Step 11. Note that this cluster corresponds to some subtree $Y \in \mathcal{N}_2$. So, by property 3 in Theorem 2.9, its total demand is at most $C' + d_{max}(\mathcal{I}_2)$, where C' is the maximum flow through any node in our SSNC solution for $\mathcal{I}_2$. By our SSNC approximation guarantee and the fact that all node capacities are $q' = 9q$ (in $\mathcal{I}_2$), we have $C' \leq \gamma \cdot q'$. Also, the maximum demand $d_{max}(\mathcal{I}_2) \leq q$ as each source node corresponds to an active cluster. So, the total demand of the new cluster is at most $(9\gamma + 1)q$. Now, consider a new cluster S formed in MergeT1; see Step 12. This cluster corresponds to some subtree $X \in \mathcal{N}_1$ with root $r \notin N = \{v_F : F \in \mathcal{T}_h\}$; note that N is the set of neighbors of sink t in instance $\mathcal{I}_1$. Again, by property 3 in Theorem 2.9, the demand of S is at most $C'' + d_{max}(\mathcal{I}_1)$, where C'' is the maximum flow through any node in our SSNC solution for $\mathcal{I}_1$. By our SSNC approximation guarantee and the fact that node capacities in $\mathcal{I}_1$ are $\tilde{q} = 5q$, we have $C'' \leq \gamma \cdot \tilde{q}$. (By property 5 of Theorem 2.9, cluster S does not contain any node of N, so it is not affected by the larger capacity on the v-nodes.) Again, the maximum demand $d_{max}(\mathcal{I}_1) \leq q$. So, the total demand of the new cluster is at most $(5\gamma + 1)q$.

Property 4. Consider any existing heavy cluster $F \in \mathcal{T}_h$. The clusters that get added to F correspond to subtrees $X \in \mathcal{N}_1$ with root $r = v_F$; see Step 10 in MergeT1. Moreover, node v_F has capacity 9γ (which is equivalent to having 9γ *copies* of v_F). So, there may be up to 9γ subtrees $X \in \mathcal{N}_1$ with root v_F. Each such subtree has demand at most $(5\gamma + 1)q$, as shown above. Hence, the increase in demand of F is at most $9\gamma(5\gamma + 1)q \leq 54\gamma^2 q$.

Property 5. Let m_1 (resp., m_2) be the number of t1-active (resp., t2-active) clusters at the start of the iteration. Let $\mathcal{A}'_1$ (resp., $\mathcal{A}'_2$) be the t1-active (resp., t2-active) clusters at the end of algorithm MergeT1 (resp., MergeT2). Any cluster that is active at the end of the iteration must be in $\mathcal{A}'_1 \cup \mathcal{A}'_2$. (Note that some clusters in $\mathcal{A}'_1 \cup \mathcal{A}'_2$ may become internal/heavy during the pruning step in MCcluster.) We now bound $|\mathcal{A}'_1|$ and $|\mathcal{A}'_2|$ separately.

- The clusters in $\mathcal{A}'_1$ are based on the SSNC instance $\mathcal{I}_1$ (in MergeT1), which has a source for every cluster in $\mathcal{A}_1$. In particular, each cluster in $\mathcal{A}'_1$ corresponds to some subtree $X \in \mathcal{N}_1$ with root $r \notin N = \{v_F : F \in \mathcal{T}_h\}$. See Step 12 in MergeT1. As r is not a neighbor of sink t, property 4 in Theorem 2.9 implies that subtree X contains at least two sources. It follows that $|\mathcal{A}'_1| \leq \frac{1}{2}|\mathcal{A}_1|$.
- The clusters in $\mathcal{A}'_2$ are based on the SSNC instance $\mathcal{I}_2$ (in MergeT2). Recall that we use a bi-partition $(\mathcal{A}^+, \mathcal{A}^-)$ of $\mathcal{A}_2$. Instance $\mathcal{I}_2$ has a source for each cluster in $\mathcal{A}^+$, and the neighbors M of the sink correspond to clusters in $\mathcal{A}^-$. Let $B \subseteq M$ denote the neighbors of the sink that appear in some subtree of $\mathcal{N}_2$; let $\mathcal{B} \subseteq \mathcal{A}^-$ denote the corresponding clusters. The collection $\mathcal{A}'_2$ consists of (i) clusters from $\mathcal{A}^- \setminus \mathcal{B}$, and (ii) clusters corresponding to subtrees in $\mathcal{N}_2$. By property 4 in Theorem 2.9, every subtree in $\mathcal{N}_2$ has at least two clusters from $\mathcal{A}^+ \cup \mathcal{B}$. So, the number of clusters in case (ii) above is at most $\frac{1}{2} \cdot (|\mathcal{A}^+| + |\mathcal{B}|)$. Clearly, the number of clusters in case (i) is $|\mathcal{A}^-| - |\mathcal{B}|$. So,

$$|\mathcal{A}_2'| \le \frac{|\mathcal{A}^+| + |\mathcal{B}|}{2} + |\mathcal{A}^-| - |\mathcal{B}| = \frac{|\mathcal{A}_2|}{2} + \frac{|\mathcal{A}^-| - |\mathcal{B}|}{2} \le \frac{3}{4} \cdot |\mathcal{A}_2|,$$

where we used that $|\mathcal{A}^-| \le |\mathcal{A}_2|/2$ as $|\mathcal{A}^-| \le |\mathcal{A}^+|$.
Thus, $|\mathcal{A}_1'| + |\mathcal{A}_2'| \le \frac{1}{2} \cdot |\mathcal{A}_1| + \frac{3}{4} \cdot |\mathcal{A}_2| \le \frac{3}{4} \cdot (m_1 + m_2)$. ∎

Finally, we show that the demand preserved in set K at the end of algorithm MCcluster is a constant fraction of the total demand in MCNC.

LEMMA 3.10. *The total demand at the end of* MCcluster *is* $\sum_{i \in K} d_i \ge D/4$.

Proof. Note that requests are only removed in Step 10 of MCcluster. We will account for the deleted requests by explicitly "charging" them to requests that will be preserved in K (at the end of MCcluster).

Consider any cluster T that is processed in Step 10 at any iteration of MCcluster. Note that T must be in the active set, i.e., $T \in \mathcal{A}_1 \cup \mathcal{A}_2$. Let t_i be the total demand of T's internal terminals; note that the demand of T's internal requests is $t_i/2$ as each such request has two terminals in T. Let t_h (resp., t_a) be the total demand of T's external terminals with mates in $\mathcal{T}_h$ (resp., $\mathcal{A}_1 \cup \mathcal{A}_2$). By Lemma 3.6, there are no requests (in the current set K) from T to any cluster of $\mathcal{T}_i$. So, $\mathsf{load}(T) = t_i + t_h + t_a$. In this step, we remove (from K) all requests from T to other active clusters. So, the deleted demand is exactly t_a. Furthermore, we claim that the following requests incident to T will be preserved (in K) until the end.

- *Internal requests in T.* This is because we never remove any internal request. Also, clusters only merge with each other during the algorithm, so any internal request in T will remain internal to some cluster.
- *External requests from T to $\mathcal{T}_h$.* This is because we never remove any request incident to a heavy cluster. Also, any cluster that is currently heavy will remain heavy for the rest of the algorithm.

The total demand in these "preserved" requests is $\frac{t_i}{2} + t_h$. We now show that the removed demand $t_a \le 3 \cdot (\frac{t_i}{2} + t_h)$. Consider the two cases when requests are removed:

- *T is an internal cluster.* Then, we have $t_i \ge \frac{\mathsf{load}(T)}{2} = \frac{t_i + t_h + t_a}{2}$; so $t_a \le t_i$.
- *T is a dangerous cluster.* Here, T is t1-active, which means it has at least $\frac{\mathsf{load}(T)}{4}$ demand going to heavy clusters (recall that T has no requests to internal clusters). Then, we have $t_h \ge \frac{t_i + t_h + t_a}{4}$, which implies $t_a \le 3t_h$.

Thus, the total removed demand incident to T is at most 3 times the total preserved demand incident to T.

Finally, we show that the preserved requests incident to different clusters T that are processed in Step 10 (MCcluster) are *disjoint.* To this end, we will show that once a cluster T is processed in Step 10, it will not be part of any cluster T' that is processed in Step 10 (and removes requests) at a later iteration. Indeed, after cluster T is processed in Step 10, there are no requests from T to other active clusters. If cluster T becomes internal, then it remains unchanged in later iterations. If cluster T becomes t1-active, then it may merge with other clusters, but these can only be heavy clusters (in which case T also becomes part of that heavy cluster) or t1-active clusters (which by Lemma 3.6 also have no external requests to active clusters). In either case, cluster T will never be part of another cluster that gets processed in Step 10 again. It now follows that the total removed demand (over all iterations) is at most 3 times the total preserved demand, which completes the proof. ∎

Completing proof of Theorem 3.4. We are now ready to prove the multicommodity clustering theorem. Let $\widehat{\mathcal{T}}$ denote the final collection of clusters and $K \subseteq [k]$ the final set of requests, at the end of algorithm MCcluster.

Property (i). *Each cluster in $\widehat{\mathcal{T}}$ is internal or heavy.* By the termination condition, there is no active cluster remaining at the end, i.e., $\mathcal{A}_1 \cup \mathcal{A}_2 = \emptyset$. So, $\widehat{\mathcal{T}} = \mathcal{T}_h \cup \mathcal{T}_i$, which means all clusters are internal/heavy (see Lemma 3.5).

Property (ii). *Each terminal (i.e., node in $\{s_i, t_i\}_{i \in K}$) lies in exactly one cluster.* Initially, each terminal is in its own cluster, so this property is true. In algorithm MCcluster, we only merge clusters together by adding some subtrees (based on SSNC instances). Note that these subtrees do not contain any terminal because each terminal is a leaf-node, and the sources/sink in our SSNC instances are new nodes (different from the original terminals). So, each terminal lies in exactly one cluster throughout the algorithm.

Property (iii). *The total demand of request-pairs K is $\sum_{i \in K} d_i \geq D/4$.* This follows directly from Lemma 3.10.

Property (iv). *Each node appears in at most $O(\log k)$ different clusters.* We first claim that the number of iterations in MCcluster is $O(\log k)$. Indeed, by Lemma 3.9(5), the number of active clusters drops by a factor of 4/3 in each iteration. As there are $2k$ active clusters initially, we will have no active clusters left after $O(\log k)$ iterations. By Lemma 3.9(1), in each iteration, each node gets added to at most two clusters. Combined with the number of iterations, property (iv) follows.

Property (v). *The demand of each cluster is at most $O(\gamma^2 \log k) \cdot q$.* The demand of any internal/heavy cluster when it is formed is $O(\gamma) \cdot q$ by Lemma 3.9(3). Note that internal clusters do not change after they are formed (and added to $\mathcal{T}_i$). For any heavy cluster, by Lemma 3.9(4), the increase in demand is $O(\gamma^2) \cdot q$ in each iteration. As the number of iterations is $O(\log k)$, the final demand of any heavy cluster is $O(\gamma^2 \log k) \cdot q$.

Property (vi). *The total cost $\sum_{T \in \widehat{\mathcal{T}}} \sum_{v \in T} c_v \leq O(\beta \cdot \log k) \cdot \mathsf{Opt}$.* The cost of nodes added in any iteration is $O(\beta) \cdot \mathsf{Opt}$ by Lemma 3.9(2). Combined with the $O(\log k)$ number of iterations, we obtain property (vi).

This completes the proof of Theorem 3.4.

3.2. Routing across clusters. From Theorem 3.4, we have a collection $\widehat{\mathcal{T}}$ of low-cost, nearly node-disjoint clusters containing requests $K \subseteq [k]$. Here, we show how to route a constant fraction of the requests in K. Routing within a cluster can be done easily using the corresponding subtree. So we focus on routing each request across clusters, from their "source cluster" to their "sink cluster." In order to achieve this, we will add some nodes/edges to our solution.

Algorithm overview. Recall that $\widehat{\mathcal{T}}$ contains two types of clusters: internal and heavy. Let $\mathcal{T}_i$ denote all internal clusters in $\widehat{\mathcal{T}}$, and $\mathcal{T}_h = \widehat{\mathcal{T}} \setminus \mathcal{T}_i$ denote the heavy clusters. If a cluster is both internal and heavy, then it is treated as an internal cluster. If $\mathcal{T}_i$ contains a constant fraction of the demand in K, then we don't need to route across clusters; we simply use the subtrees in $\mathcal{T}_i$ to route all these internal requests. The hard case is when most of the demand in K is contained in $\mathcal{T}_h$. In this case, we use an idea from [7] for the *edge*-capacitated routing problem. Specifically, each request i "hallucinates" that its demand equals q with probability $\approx \log(k) \frac{d_i}{q}$ (and zero otherwise), and we find a subgraph $\mathcal{H}$ that supports all the hallucinated demands. We can find a good approximation to this "hallucinated instance" by rounding a natural linear program (see section 3.2.2). We then use the union of $\mathcal{H}$ and $\widehat{\mathcal{T}}$ as our solution. However (unlike the edge-capacitated case), this solution may not suffice to route *all* the remaining demands. Nevertheless, we show that a constant fraction of the remaining demand can still be routed in $\mathcal{H} \cup \widehat{\mathcal{T}}$. To this end, we partition the heavy clusters into $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_p$ such that the minimum (edge) cut of the demand

Algorithm 3.4. MCNC Routing Algorithm.

1: if the internal requests of clusters in $\mathcal{T}_i$ have demand at least $\frac{1}{6}\sum_{i\in K} d_i$, return $\mathcal{E} = \mathcal{T}_i$ as the solution and all internal requests in $\mathcal{T}_i$ as the routable pairs K'.
2: apply Theorem 3.13 to the heavy clusters $\mathcal{T}_h$ and obtain partition $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_p$.
3: let $K' \subseteq K$ denote the union of requests induced in $\mathcal{T}_j$ (for $j = 1, 2, \ldots, p$).
4: let $r = \frac{\alpha_1 \log k}{q}$, where $\alpha_1 > 1$ is some constant.
5: let $\mathcal{M}$ denote the random instance with each request $i \in K$ having demand $B_i \cdot q$, where $B_i \sim \mathsf{Binomial}(d_i, r)$ independently.
6: apply Theorem 3.15 to obtain solution $\mathcal{H} \subseteq V$ for the hallucinated instance $\mathcal{M}$.
7: return $\mathcal{E} = \mathcal{H} \cup \mathcal{T}_h$ as the solution and K' as the routable pairs.

graph induced on each $\mathcal{T}_j$ is at least $\Omega(q)$. This partitioning algorithm is based on iteratively removing *minimal* min-cuts (see section 3.2.1). We also delete all requests in K crossing from one part to another and prove that the remaining demand is still a constant fraction of that in K. Then, we show that when min-cuts are large, the hallucinated request-pairs behave like a *cut-sparsifier* [34] of the demand graph (after contracting the clusters). So the hallucinated solution $\mathcal{H}$ has enough capacity to support an *edge-capacitated* routing across clusters. Finally, we "un-contract" these clusters by using the trees in $\mathcal{T}_h$ to route within each cluster (see section 3.2.3). We bound the *node congestion* of the routing using the fact that each cluster has load $O(\gamma^2 \log k) \cdot q$. The formal algorithm is given as Algorithm 3.4: it takes as input the clusters $\widehat{\mathcal{T}}$ (and requests K) from Theorem 3.4 and outputs a subgraph $\mathcal{E}$ of G along with a subset $K' \subseteq K$ of requests that can be routed in $\mathcal{E}$ at low node congestion.

The rest of this subsection proves the following main result.

THEOREM 3.11. *Given any* MCNC *instance with optimal cost* Opt*, after running Algorithms* 3.1 *and* 3.4*, we have the following with probability at least* $1 - O(\frac{1}{k^2})$*:*

- *The cost of solution* $\mathcal{E}$ *is* $O(\beta \log k) \cdot \mathsf{Opt}$.
- *Solution* $\mathcal{E}$ *supports an unsplittable routing for all requests in* K' *with node congestion* $O(\gamma^2 \log^3 k)$.
- *The total demand of requests in* K' *is at least* $D/24$.

Throughout, we assume that $r = \frac{\alpha_1 \log k}{q} < 1$, so r is a valid probability value in step 5. If $r \geq 1$, then we have $q = O(\log k)$, in which case there is an easy $(O(\log n \log k), O(\log n \log k))$ bicriteria approximation algorithm; see Appendix B.1.

3.2.1. Identifying routable request-pairs in heavy clusters. We now show how to identify a subset $K' \subseteq K$ of request-pairs by partitioning the demand graph into components of high min-cut values. Having a high min-cut in the demand graph is necessary for the cut-sparsification argument that is used (in section 3.2.3) to route requests K' with low congestion.

We first define a cluster multigraph as follows.

DEFINITION 3.12 (cluster multigraph $\mathcal{C}(\mathcal{T})$). *Given a collection* $\mathcal{T}$ *of clusters, the multigraph* $\mathcal{C}(\mathcal{T})$ *has a node for every cluster* $T \in \mathcal{T}$*, and for each request-pair* $i \in K$*, an edge of weight* d_i *between clusters* $T_a, T_b \in \mathcal{T}$*, where* $s_i \in T_a$ *and* $t_i \in T_b$.

We now show how to partition $\mathcal{C}(\mathcal{T})$ into several *parts* so that the minimum (edge) cut of each part is $\Omega(q)$ and the demand of "crossing" requests is small. For any graph J and subset X of nodes, we use the notation $\partial_J(X)$ to denote the set of edges in J with exactly one endpoint in X.

THEOREM 3.13. *There is a polynomial-time algorithm that, given any collection $\mathcal{T}$ of clusters, computes a partition $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_p$ of $\mathcal{T}$ such that the following hold:*

i. *For each $j \in [p]$, the induced cluster graph $\mathcal{C}(\mathcal{T}_j)$ has min-cut at least $q/8$.*

ii. *The total weight of edges in $K' = \bigcup_{j=1}^{p} \mathcal{C}(\mathcal{T}_j)$ is at least $W - \frac{Nq}{4}$.*

Here, $N = |\mathcal{T}|$ and W is the total weight of edges in $\mathcal{C}(\mathcal{T})$.

Proof. Consider the following procedure to obtain the partition. Initially, $\mathcal{T}' = \mathcal{T}$ and K' consists of all edges in $\mathcal{C}(\mathcal{T})$. For $j = 1, 2, \ldots$ do the following:

1. If the min-cut value in $\mathcal{C}(\mathcal{T}')$ is at least $\frac{q}{4}$, then $\mathcal{T}_j \leftarrow \mathcal{T}'$ and stop.
2. Let $S \subseteq \mathcal{T}'$ denote a *minimal* min-cut in graph $\mathcal{C}(\mathcal{T}')$.
3. Set $\mathcal{T}_j \leftarrow S$, $\mathcal{T}' \leftarrow \mathcal{T}' \setminus S$, and $K' \leftarrow K' \setminus \partial_{\mathcal{C}(\mathcal{T}')}(S)$.

Note that this procedure creates one part in each iteration. At any point, $\mathcal{T}'$ denotes the remaining set of clusters/nodes (that are still unassigned to parts), and K' denotes the current set of noncrossing edges. (An edge is said to be crossing if its endpoints lie in different parts.) Let $G' = \mathcal{C}(\mathcal{T}')$ denote the current graph. Let p denote the number of iterations, which is also the number of parts in the partition of $\mathcal{T}$. At the end of this procedure, K' equals the set of edges in $\bigcup_{j=1}^{p} \mathcal{C}(\mathcal{T}_j)$, which are all the noncrossing edges.

We first prove condition (ii). Clearly, the number of iterations is at most N, the number of nodes in $\mathcal{C}$. As S is a min-cut, the total weight of the edges $\partial_{G'}(S)$ removed in any iteration is at most $q/4$. So the total weight of all edges removed is at most $(Nq)/4$. So, the total weight of K' (at the end) is at least $W - (Nq)/4$.

We now prove condition (i). Note that each part $\mathcal{T}_j$ is either (i) the set S in some iteration above, or (ii) the final set $\mathcal{T}'$. Clearly, in the latter case, graph $\mathcal{C}(\mathcal{T}_j)$ has min-cut at least $q/4 \geq \frac{q}{8}$. In the former case, consider the graph $G' = \mathcal{C}(\mathcal{T}')$ and set $S \subseteq \mathcal{T}'$ in the iteration when $\mathcal{T}_j = S$ was created. If $|S| = 1$, then there is nothing to prove as part $\mathcal{T}_j = S$ would have infinite min-cut value. Let $A \subset S$ be any strict subset. By minimality of S, the weights of $\partial_{G'}(A)$ and $\partial_{G'}(S \setminus A)$ are both at least $\frac{q}{4}$. Let a (resp., b) denote the total weight of edges having one endpoint in A (resp., $S \setminus A$) and the other endpoint in $\mathcal{T}' \setminus S$. Also, let x denote the total weight of edges having one endpoint in A and the other in $S \setminus A$. Note that the weight of $\partial_{G'}(A)$ is $x + a$, the weight of $\partial_{G'}(S \setminus A)$ is $x + b$, and the weight of $\partial_{G'}(S)$ is $a + b$. Combined with the observation above (by minimality of cut S),

$$x + a \geq \frac{q}{4}, \quad x + b \geq \frac{q}{4}, \quad a + b < \frac{q}{4}.$$

It follows that $x \geq \frac{q}{8}$, i.e., the weight of edges between A and $S \setminus A$, is at least $\frac{q}{8}$. As this holds for all strict subsets $A \subset S$, the min-cut of $G'[S] = \mathcal{C}(\mathcal{T}_j)$ is at least $\frac{q}{8}$. ∎

In our algorithm, we apply this result to the collection of heavy clusters $\mathcal{T}_h$. Next, we show how to add some nodes $\mathcal{H}$ to the heavy clusters (see section 3.2.2) so that all requests in K' can be routed in the resulting solution $\mathcal{E} = \mathcal{T}_h \cup \mathcal{H}$ with low node congestion (see section 3.2.3).

3.2.2. Hallucinating to connect heavy clusters. Here, we show how to find a low-cost solution $\mathcal{H}$ for the "hallucinated instance" $\mathcal{M}$. Recall that instance $\mathcal{M}$ has demand $B_i \cdot q$ for each request-pair $i \in K$, where $B_i \sim \mathsf{Binomial}(d_i, r)$ independently. Note that the expected demand of request i is $\mathbb{E}[B_i] \cdot q = r d_i q = O(\log k) \cdot d_i$. We treat the $B_i \cdot q$ demand of each request i as B_i many copies (each with demand q); so these demands can be sent along B_i different $s_i - t_i$ paths (each carrying q units). Note that all nodes in graph G have capacity q. By scaling down all capacities/demands by q, we obtain an equivalent instance with unit node-capacities and B_i many (unit

demand) requests between s_i and t_i, for all $i \in K$. For simplicity, we work with this scaled instance as $\mathcal{M}$. We first show that (with high probability) there exists a solution to $\mathcal{M}$ of low cost and bounded node congestion, and then we provide an algorithm to find such a solution.

LEMMA 3.14. *With probability at least* $1 - O(\frac{1}{n^2})$*, there is an unsplittable routing* $\{P_i^*\}_{i \in \mathcal{M}}$ *of the hallucinated requests where* $\sum_{i \in \mathcal{M}} \sum_{v \in P_i^*} c_v \leq (\alpha_2 \log n) \cdot \mathsf{Opt}$ *and the node-congestion is at most* $\alpha_2 \cdot \log n$*, where* $\alpha_2 = O(1)$*.*

Proof. Consider the optimal solution for the original MCNC instance. Let P_i^* denote the path used for sending d_i units of flow between s_i and t_i, for each request $i \in [k]$. We now consider the solution X that sends B_i units of flow on the optimal path P_i^* for each request $i \in K$. (Equivalently, each of the B_i copies of request i uses the same path P_i^* to route its unit flow.) We now show that this solution has $O(\log n)$ congestion with high probability. To see this, consider any node $v \in V$. By feasibility of the solution $\{P_i^* : i \in [k]\}$, we have $\sum_{i \in [k]: v \in P_i^*} d_i \leq q$. The load on node v in solution X is $L_v := \sum_{i \in K: v \in P_i^*} B_i$. As each B_i is a binomial random variable, L_v is the sum of independent $[0,1]$ random variables. The mean

$$\mathbb{E}[L_v] = \sum_{i \in K: v \in P_i^*} \mathbb{E}[B_i] = \frac{\alpha_1 \log k}{q} \sum_{i \in K: v \in P_i^*} d_i \leq \alpha_1 \log k \leq \alpha_1 \log n.$$

By a Chernoff bound, there is a constant α_2 such that $\Pr[L_v > \alpha_2 \cdot \log n] \leq \frac{1}{n^3}$. Taking a union bound over all n nodes, we have $\Pr[\exists v : L_v > \alpha_2 \cdot \log n] \leq \frac{1}{n^2}$. We condition on the event that $L_v \leq \alpha_2 \cdot \log n$ for all $v \in V$. Then, the node-congestion of solution X is as claimed. We now bound the cost:

$$\sum_{i \in \mathcal{M}} \sum_{v \in P_i^*} c_v = \sum_{v \in V} c_v \cdot L_v = \sum_{v \in \mathsf{Opt}} c_v \cdot L_v \leq (\alpha_2 \log n) \cdot \mathsf{Opt}.$$

Hence solution X satisfies both properties in the lemma. ∎

THEOREM 3.15. *There is a polynomial algorithm that finds a solution* $\mathcal{H}$ *for the hallucinated instance* $\mathcal{M}$ *satisfying the following with probability at least* $1 - O(\frac{1}{n^2})$*:*

- *The total cost of nodes in* $\mathcal{H}$ *is* $O(\log n) \cdot \mathsf{Opt}$*.*
- *The node congestion of* $\mathcal{H}$ *is* $O(\log n)$*.*

Proof. We use the following linear program relaxation:

$$(LP_h) \qquad \min \sum_{i \in \mathcal{M}} \sum_{p \in \mathcal{P}_i} \left(\sum_{v \in p} c_v \right) \cdot f(p)$$

$$\text{(3.1)} \qquad \text{s.t.} \sum_{p \in \mathcal{P}_i} f(p) = 1 \qquad \forall i \in \mathcal{M},$$

$$\text{(3.2)} \qquad \sum_{p | v \in p} f(p) \leq \alpha_2 \cdot \log n \qquad \forall v \in V,$$

$$\text{(3.3)} \qquad f(p) \geq 0 \qquad \forall i \in \mathcal{M},\ \forall p \in \mathcal{P}_i.$$

Here, $\mathcal{P}_i$ is the set of all s_i-t_i flow paths in G. Constraint (3.1) requires that exactly one path be selected for each request $i \in \mathcal{M}$ (recall that $\mathcal{M}$ contains B_i copies of each request $i \in K$). Constraint (3.2) bounds the node-congestion by $O(\log n)$. Note that the LP objective corresponds to the sum of costs over all paths in the solution (so each node gets counted multiple times). By Lemma 3.14, the optimal value of

(LP_h) is at most $(\alpha_2 \log n) \cdot \mathsf{Opt}$ with probability at least $1 - O(\frac{1}{n^2})$. Although this LP has an exponential number of variables, it can be reformulated using a polynomial number of (flow-based) variables. Hence, we can solve this LP exactly in polynomial time.

We now perform simple randomized rounding of the LP solution. For each request $i \in \mathcal{M}$, select a random path $U_i \in \mathcal{P}_i$ with probability $\{f(p)\}_{p \in \mathcal{P}_i}$ independently. The solution $\mathcal{H} = \cup_{i \in \mathcal{M}} U_i$. We now prove that $\mathcal{H}$ satisfies the claimed properties with probability at least $\frac{1}{2}$.

For any path p, let $c(p) = \sum_{v \in p} c_v$ denote its total node cost. The cost of $\mathcal{H}$ is at most $C(\mathcal{H}) := \sum_{i \in \mathcal{M}} c(U_i)$; note that $\mathbb{E}[C(\mathcal{H})]$ equals the LP optimal value, which is at most $(\alpha_2 \log n) \cdot \mathsf{Opt}$. So, by Markov's inequality, with probability at least $\frac{2}{3}$, the cost of $\mathcal{H}$ is at most $(3\alpha_2 \log n) \cdot \mathsf{Opt}$.

We now bound the node congestion. For any node $v \in V$, let L_v denote the number of paths in $\{U_i\}_{i \in \mathcal{M}}$ containing v. Note that L_v is the sum of independent 0/1 random variables, with mean $\mathbb{E}[L_v] \leq \alpha_2 \log n$ by (3.2). By a Chernoff bound, there is a constant α_3 such that $\Pr[L_v > \alpha_3 \cdot \log n] \leq \frac{1}{n^3}$. Taking a union bound over all nodes v, we have $\Pr[\exists v : L_v > \alpha_3 \cdot \log n] \leq \frac{1}{n^2}$.

Hence, with probability at least $\frac{1}{2}$, we obtain both the claimed properties of $\mathcal{H}$. We can boost the success probability by repeating this algorithm independently $O(\log n)$ times and returning the best solution found as $\mathcal{H}$. This proves that $\mathcal{H}$ satisfies the claimed properties with probability at least $1 - O(\frac{1}{n^2})$. ∎

3.2.3. Routing flow in hallucinated graph. Here, we show that all requests in K' can be routed in our solution $\mathcal{E} = \mathcal{T}_h \cup \mathcal{H}$ with low congestion. Recall that K' is the set of "routable requests" identified in section 3.2.1 and $\mathcal{H}$ is the hallucinated solution found in section 3.2.2. Also, recall the partition $\mathcal{T}_1, \ldots, \mathcal{T}_p$ of heavy clusters obtained by applying Theorem 3.13. For each part $\mathcal{T}_j$, define a random edge-capacitated graph $H_C(j)$ as follows. Nodes of $H_C(j)$ correspond to clusters of $\mathcal{T}_j$. For each request i with both s_i and t_i in clusters of $\mathcal{T}_j$, there are B_i *edges of capacity* q in $H_C(j)$ between the clusters containing s_i and t_i; these edges correspond to the B_i many $s_i - t_i$ paths found in the hallucinated instance $\mathcal{M}$ (see Theorem 3.15).

Henceforth, we shall slightly abuse notation and refer to the cluster graph $\mathcal{C}(\mathcal{T}_j)$ as just $\mathcal{C}(j)$. Recall that each edge in $\mathcal{C}(j)$ corresponds to some request $i \in K$ with both s_i and t_i in $\mathcal{T}_j$ and has weight d_i (the demand of request i).

We will make use of the following cut-sparsification result.

THEOREM 3.16 (Theorem 2.1 [34]). *Let G be an N-node multigraph with min-cut κ, and $r \in [0,1]$. Let H be a multigraph containing each edge of G independently with probability r. If $r \cdot \kappa \geq \frac{3(d+2)\ln N}{\epsilon^2}$ for some d, ϵ, then with probability $1 - O(1/N^d)$, every cut in H has value within $r(1 \pm \epsilon)$ of the cut value in G.*

LEMMA 3.17. *For any $j \in [p]$, with probability at least $1 - O(\frac{1}{k^3})$, all request-pairs in $\mathcal{C}(j)$ can be routed fractionally in $H_C(j)$ without exceeding edge capacities.*

Proof. By Theorem 3.13(i), the minimum cut in $\mathcal{C}(j)$ has value $\kappa \geq \frac{q}{8}$. Let $\widetilde{\mathcal{C}}(j)$ be an *unweighted* multigraph obtained from $\mathcal{C}(j)$ by replacing each edge i in $\mathcal{C}(j)$ with d_i parallel edges (d_i is the demand of request i). Note that for any subset $S \subseteq \mathcal{T}_j$, its cut values in $\mathcal{C}(j)$ and $\widetilde{\mathcal{C}}(j)$ are the same. So the min-cut in $\widetilde{\mathcal{C}}(j)$ is also $\kappa \geq \frac{q}{8}$. Note that $H_C(j)$ can be viewed equivalently as a random subgraph of $\widetilde{\mathcal{C}}(j)$ obtained by selecting each edge independently with probability $r = \frac{\alpha_1 \ln k}{q}$ and assigning capacity q to each selected edge.

We now apply Theorem 3.16 on graph $\widetilde{\mathcal{C}}(j)$ with r and κ as above. Note that $r \cdot \kappa \geq \frac{\alpha_1}{8} \ln k \geq 60 \ln k$, assuming that $\alpha_1 \geq 480$. So, we can set $d = 3$ and $\epsilon = \frac{1}{2}$ in this

theorem, which implies that with probability at least $1 - O(\frac{1}{k^3})$, for every subset $S \subseteq \mathcal{T}_j$, its cut value in $H_C(j)$ is at least $\frac{\alpha_1 \ln k}{2}$ times its cut value in $\mathcal{C}(j)$.

Note that each edge in $H_C(j)$ has capacity q, so the cut value of S in $H_C(j)$ is $cut_H(S) := q \cdot |\partial_{H_C(j)}(S)|$. Let $cut_G(S) := \sum_{i \in \partial_{\mathcal{C}(j)}(S)} d_i$ denote the cut value of S in $\mathcal{C}(j)$. In other words, the *nonuniform sparsest cut* of the multicommodity routing instance with demand-graph $\mathcal{C}(j)$ and capacity-graph $H_C(j)$ is

$$\min_{S \subseteq \mathcal{T}_j} \frac{\text{capacity across } S}{\text{demand across } S} = \min_{S \subseteq \mathcal{T}_j} \frac{cut_H(S)}{cut_G(S)} \geq \frac{\alpha_1 \ln k}{2}.$$

Choosing α_1 to be a large enough constant, we can ensure that the above sparsest cut value is at least the multicommodity flow-cut gap $\Theta(\log k)$ [40]. This proves the existence of a fractional routing for request-pairs in $\mathcal{C}(j)$. ∎

This lemma enables us to find an *edge-capacitated* multicommodity flow for the request pairs K' in $\bigcup_{j=1}^{p} \mathcal{C}(j)$. For each request i in $\mathcal{C}(j)$, let f_i denote the fractional flow sending d_i units from the source to sink cluster in graph $H_C(j)$. Note that the flows $\{f_i\}_{i \in K'}$ can be routed concurrently, while respecting all edge capacities, so the total flow through each edge of $\bigcup_{j=1}^{p} H_C(j)$ is at most q. Further, the next lemma shows that the total flow through any node is also bounded.

LEMMA 3.18. *The total capacity of edges in* $\bigcup_{j=1}^{p} H_C(j)$ *incident to any node is* $O(\gamma^2 \log^2 k) \cdot q$, *with probability at least* $1 - O(\frac{1}{k^3})$.

Proof. Consider any node (cluster) $T \in H_C(j)$ for any $j \in [p]$. From Theorem 3.4(iv), we know that the total demand of requests R_T incident to T (i.e., having a terminal in T) is $O(\gamma^2 \log k)q$. Every edge in $H_C(j)$ incident to T corresponds to some request $i \in R_T$, and the edge has capacity $B_i \cdot q$, where $B_i \sim \mathsf{Bernoulli}(d_i, r)$. So the total capacity incident to node T is $X := \sum_{i \in R_T} B_i \cdot q$. Note that $\mathbb{E}[X] = qr \sum_{i \in R_T} d_i = \alpha_1 \log k \sum_{i \in R_T} d_i = O(\gamma^2 \log^2 k) \cdot q$. As X is the sum of independent $\{0, q\}$ random variables, we obtain by a Chernoff bound that $X = O(\gamma^2 \log^2 k) \cdot q$ with probability at least $1 - \frac{1}{k^4}$. Finally, a union bound over all nodes/clusters completes the proof. ∎

Obtaining node-capacitated routing in G. Now, we show that the edge capacitated routing $\mathcal{F} = \{f_i : i \in K'\}$ in the hallucinated graph $H_C := \bigcup_{j=1}^{p} H_C(j)$ can also be implemented as a node-capacitated routing in the real graph G. This involves un-contracting nodes of H_C into clusters $\mathcal{T}_h$ and the edges of H_C into flow-paths of the hallucinated solution $\mathcal{H}$. See Figure 4 for an example.

LEMMA 3.19. *With probability at least* $1 - O(\frac{1}{k^2})$, *solution* $\mathcal{E} = \mathcal{H} \cup \mathcal{T}_h$ *in step 7 of Algorithm* 3.4 *supports an unsplittable routing of requests* K' *with node-congestion* $O(\gamma^2 \log^3 k) \cdot q$.

Proof. We start with the fractional routing $\mathcal{F}$ in the hallucinated graph H_C, which corresponds to routing flow across clusters. By Lemma 3.17, the total flow on each edge of H_C is at most q, with probability at least $1 - O(\frac{1}{k^3})$. We replace the flow in $\mathcal{F}$ on each edge i of H_C with the flow-path U_i used for request i in the hallucinated solution $\mathcal{H}$ (recall that i corresponds to some request in $\mathcal{M}$). By Theorem 3.15, each node in graph G appears in $O(\log n)$ many flow-paths $\{U_i\}$. So, this results in a flow of $O(\log n) \cdot q$ through each node of graph G. However, we do not yet have a valid routing as we still need to route flow within each cluster (i.e., node of H_C).

In order to route flow in $\mathcal{F}$ through any node/cluster $T \in H_C$, we do the following. Consider any pair of consecutive edges i_1, i_2 (both incident to cluster T) used in the

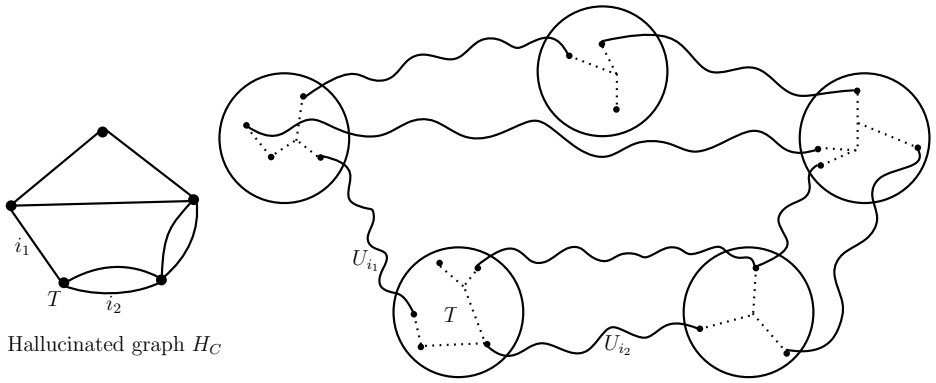

FIG. 4. *Un-contracting hallucinated graph H_C using clusters $\mathcal{T}_h$ and flow $\mathcal{H}$.*

routing $\mathcal{F}$. Although the endpoints of paths U_{i_1} and U_{i_2} may be different, they must both be terminals in cluster T, so we can use the subtree corresponding to T to route the flow through this cluster. See Figure 4. By Lemma 3.18, the total flow in $\mathcal{F}$ through any node of H_C is $O(\log^2 k)\gamma^2 \cdot q$, with probability at least $1 - O(\frac{1}{k^3})$. Moreover, by Theorem 3.4(iii), each node in graph G appears in $O(\log k)$ many clusters. Hence, the flow within clusters can be implemented so that the flow through each node is $O(\log^3 k)\gamma^2 \cdot q$.

Combining the flow routing across clusters and within each cluster, we obtain a valid fractional flow in graph G with node congestion $O(\log n) \cdot q + O(\log^3 k)\gamma^2 \cdot q = O(\log^3 k)\gamma^2 \cdot q$; here we used the fact that $\gamma \geq \log n$.

Finally, we perform simple randomized rounding to obtain an unsplittable routing. Using the fact that each demand is at most q and a Chernoff bound, the node congestion in G remains $O(\gamma^2 \log^3 k)q$ with probability at least $1 - \frac{1}{n^3}$. ∎

3.2.4. Completing proof of multicommodity routing. We now combine the results from sections 3.2.1, 3.2.2, and 3.2.3 to complete the proof of the routing algorithm (Theorem 3.11). Let $D' = \sum_{i \in K} d_i$ denote the total demand of the requests in K that are obtained after Algorithm 3.1. By Theorem 3.4(i), we have $D' \geq D/4$, where D is the total demand of all requests in the MCNC instance. Let $K_1 \subseteq K$ denote all requests $i \in K$ that have both s_i and t_i in the *same* internal cluster $T \in \mathcal{T}_i$. Let $K_2 \subseteq K$ denote all requests $i \in K$ with both s_i and t_i in some heavy cluster (the source/sink can be in different clusters of $\mathcal{T}_h$). Let $K_0 = K \setminus K_1 \setminus K_2$ be all other requests. We use D_1, D_2, and D_0 to denote the total demand of the respective sets. Let $N = |\mathcal{T}_h|$ be the number of heavy clusters. Then, we make the following claim.

CLAIM 3.20. *If $D_1 < \frac{D'}{6}$, then $(D_2 - \frac{Nq}{4}) \geq \frac{D'}{6}$.*

Proof. For every request $i \in K_0$, either s_i or t_i appears as an external terminal in some cluster $T \in \mathcal{T}_i$. This implies that D_0 is at most the total demand of external terminals of $\mathcal{T}_i$. By definition of internal clusters (Definition 3.1), the external demand of any $T \in \mathcal{T}_i$ is less than $\mathsf{load}(T)/2$ and the internal demand of T is at least $\mathsf{load}(T)/2$. Adding over all internal clusters $T \in \mathcal{T}_i$, we obtain $D_0 \leq \frac{1}{2}\sum_{T \in \mathcal{T}_i} \mathsf{load}(T)$ and $D_1 \geq \frac{1}{2}\sum_{T \in \mathcal{T}_i} \frac{\mathsf{load}(T)}{2}$ (the factor 2 reduction is because each internal request contributes twice to the cluster's load). Hence, $D_0 \leq 2D_1$.

By definition of heavy clusters, the total demand in each $T \in \mathcal{T}_h$ is at least q. Adding over all $T \in \mathcal{T}_h$, we get $D_0 + 2D_2 \geq qN$, where we used that the sum of demands over $\mathcal{T}_h$ is at most $2D_2 + D_0$.

Now, using $D' = D_0 + D_1 + D_2$ and $D_0 \leq 2D_1$, we obtain $D_2 \geq D' - 3D_1 > \frac{1}{2}D'$ as $D_1 < \frac{D'}{6}$. Also, $D_0 \leq 2D_1 < \frac{1}{3}D' < \frac{2}{3}D_2$. Now we have $Nq \leq D_0 + 2D_2 < \frac{8}{3}D_2$. So, $D_2 - \frac{Nq}{4} > D_2 - \frac{2}{3}D_2 > \frac{D'}{6}$, which proves the claim. ∎

Large internal demand. We first consider the easier case that $D_1 \geq D'/6$. So, step 1 applies in Algorithm 3.4. Here, our solution $\mathcal{E} = \mathcal{T}_i$ and requests $K' = K_1$. Note that each cluster $T \in \mathcal{T}_i$ can support all its internal demands on the tree corresponding to T with node-congestion $\mathsf{load}(T)$. The cost of $\mathcal{E}$ is $O(\beta \log k) \cdot \mathsf{Opt}$ by Theorem 3.4(v). Moreover, each node of G appears in $O(\log k)$ many clusters (Theorem 3.4(iii)) and $\mathsf{load}(T) = O(\gamma^2 \log k)q$ for each cluster T (Theorem 3.4(iv)). So the node-congestion of this routing is $O(\gamma^2 \log^2 k)q$.

Large external demand. We now consider the case that $D_1 < D'/6$. Here, our solution $\mathcal{E} = \mathcal{H} \cup \mathcal{T}_h$, where $\mathcal{H}$ is the solution to the hallucinated instance in step 6. The requests K' are those obtained in Theorem 3.13, which implies $K' \subseteq K_2$ and its total demand is at least $D_2 - \frac{Nq}{4} \geq D'/6$ (the last inequality is by Claim 3.20). By Theorem 3.15, the cost of $\mathcal{H}$ is $O(\log n) \cdot \mathsf{Opt}$ with probability $1 - \frac{1}{n^2}$. And, the cost of $\mathcal{T}_h$ is $O(\beta \log k) \cdot \mathsf{Opt}$ by Theorem 3.4(v). So the total cost of $\mathcal{E}$ is $O(\beta \log k) \cdot \mathsf{Opt}$, where we use $\beta \geq \log n$. By Lemma 3.19, with probability $1 - \frac{1}{k^2}$, requests K' can be routed with node-congestion $O(\gamma^2 \log^3 k)q$.

Thus, in either case, we have with probability at least $1 - \frac{1}{k}$ that

- the cost of solution $\mathcal{E}$ is $O(\beta \log k) \cdot \mathsf{Opt}$;
- requests K' can be routed in $\mathcal{E}$ with node-congestion $O(\gamma^2 \log^3 k)q$;
- the total demand in K' is at least $D'/6 \geq D/24$.

3.3. Wrapping up. Our algorithm for MCNC invokes Algorithms 3.1 and 3.4 iteratively $O(\log k)$ many times. By Theorem 3.11, each iteration results in cost $O(\beta \log k) \cdot \mathsf{Opt}$ and node-congestion $O(\gamma^2 \log^3 k)q$ and routes a constant fraction of the total remaining demand. Hence, after $O(\log D)$ iterations, we would have routed all the demands. As described in Appendix B.2, we can ensure that D is polynomial in $k \leq n$. So, the number of iterations is $O(\log k)$. The final cost is $O(\beta \log^2 k) \cdot \mathsf{Opt}$, and node-congestion is $O(\gamma^2 \log^4 k)q$. By Theorem 1.2, we have $\beta = O(\log^2 n)$ and $\gamma = O(\log^3 n)$, which implies that our cost is $O(\log^2 n \log^2 k) \cdot \mathsf{Opt}$ and node-congestion is $O(\log^6 n \log^4 k) \cdot q$. This completes the proof of Theorem 1.3.

4. Conclusions. In this paper, we obtained the first polylogarithmic bicriteria approximation algorithm for the uniform node-capacitated network design problem. There is still a large gap between our approximation bounds and the $\Omega(\log k)$ hardness of approximation that follows from node-weighted Steiner tree. Closing this gap is an interesting open question. Another question concerns non-uniform capacities, which is also open in the edge-capacitated case. A key challenge in dealing with nonuniform capacities is that there is no clear notion of how much demand should be aggregated in one cluster.

Appendix A. Simple reductions for MCNC and NEERP.

A.1. Reducing NEERP to MCNC. Here, we show that the energy-efficient routing problem can be reduced to the capacitated network design problem. This essentially follows from [3], but we provide the details for completeness.

Consider any instance of NEERP with energy function (1.1), graph $G = (V, E)$ having node-costs $\{c_v\}_{v \in V}$, and requests $\{(s_i, t_i, d_i)\}_{i=1}^{k}$. Let $q := \sigma^{1/\alpha}$. Let $D_1 = \{i \in [k] : d_i \leq q\}$ denote all requests with demand at most q, and let $D_2 = [k] \setminus D_1$ denote the "large" demands. We will handle these two demand types separately. Let Opt denote the optimal value.

For small demands (D_1) we define an MCNC instance on graph (V', E') where V' contains k copies of each node in V. For each $(u, v) \in E$, there is an edge in E' between every copy of u and every copy of v. For any node $v \in V$ and $i \in [k]$, the ith copy of v in V' has cost $c_v\sigma(i^\alpha - (i-1)^\alpha)$. The requests remain the same (we can use any copy of the source/sink nodes). We claim that the optimal value of the MCNC instance is at most $2^\alpha \cdot$ Opt. To see this, consider the same routing as for the optimal NEERP solution. If the flow through any node v is x, then we include the *first* $\lceil x/q \rceil$ copies of v into the MCNC solution. Note that the cost of these copies of node v is

$$c_v\sigma\lceil x/q \rceil^\alpha \leq c_v\sigma\,(1 + x/q)^\alpha = c_v(q + x)^\alpha \leq c_v 2^\alpha(q^\alpha + x^\alpha) = 2^\alpha \cdot c_v \cdot f(x),$$

which is 2^α times the cost of node v in the NEERP solution. Adding over all nodes, the total cost of this MCNC solution is at most $2^\alpha \cdot$ Opt.

Let $U \subseteq V'$ denote a (β, γ) bicriteria approximate solution for MCNC. Then, there is a feasible multicommodity flow that uses capacity at most $\gamma \cdot q$ on each node of U. By solving the natural LP and random rounding, we can also find (in polynomial time) a flow with load at most $L := O(\gamma + \log n) \cdot q$ on each node of U. We now bound the NEERP cost of this flow. Consider any node $v \in V$ that has i copies selected in U, so the total cost of these nodes is $c_v\sigma i^\alpha$. The energy cost on v is at most

$$c_v(\sigma + (iL)^\alpha) = O(\gamma^\alpha + \log^\alpha n)\sigma i^\alpha \cdot c_v.$$

Adding over all $v \in V$, the total energy cost is $O(\gamma^\alpha + \log^\alpha n)$ times the MCNC cost, which is $O(\gamma^\alpha \cdot \beta) \cdot$ Opt. Here, we assumed that $\gamma \geq \log n$.

For large demands (D_2) we just find an unsplittable routing that minimizes the αth power of loads. Let $\mathcal{I}'$ denote this problem instance. Note that this problem differs from NEERP only in the definition of the energy function, which is now $f'(x) = x^\alpha$ instead of $f(x) = \sigma + x^\alpha$. There is an α^α-approximation algorithm for this problem [41]. Clearly, the optimal value of $\mathcal{I}'$ is at most Opt. Let τ denote an approximate solution to $\mathcal{I}'$ and $U \subseteq V$ denote the nodes carrying positive flow. As every request in D_2 has demand at least q (and we have unsplittable flows), every node in U has flow *at least* q. Using the fact that $f(x) \leq 2f'(x)$ for all $x \geq q$, it now follows that the NEERP cost of τ is at most *twice* the f'-cost of τ, i.e., at most $2\alpha^\alpha \cdot$ Opt.

Combining the routing for D_1 and D_2 completes the proof of Theorem 1.1.

A.2. Approximation ratio relative to splittable routing. In our node-capacitated network design problem (MCNC and SSNC), our goal is to find an *unsplittable* routing of each demand. We now observe that our approximation guarantees in Theorems 1.2 and 1.3 are stronger, and hold relative to an optimal solution that only supports a *splittable* (i.e., fractional) flow of the demands.

For SSNC, we only use the optimal solution Opt in Lemma 2.3. Observe that this relies on applying the confluent flow result (Theorem 2.2) to the optimal solution, which only requires a splittable flow.

For MCNC, we use the optimal solution Opt in the following steps:

- Bounding the optimal cost of the SSNC instances $\mathcal{I}_1$ and $\mathcal{I}_2$ (Lemmas 3.7 and 3.8). Here, we only use Opt to demonstrate a feasible fractional routing for $\mathcal{I}_1$ and $\mathcal{I}_2$, so we can also use a splittable-routing solution.

- Bounding the optimal cost of the hallucinated instance (Lemma 3.14). Here, we used the optimal paths P_i^* to route a random quantity B_i between s_i and t_i, for each request $i \in [k]$. Then, we used a Chernoff bound to prove that the node congestion is $O(\log n)$ with high probability. Instead of an integral path P_i^*, we can use a fractional flow as a distribution $\mathcal{F}_i$ over $s_i - t_i$ paths for each $i \in [k]$. Then, we can first sample a random path P_i from $\mathcal{F}_i$ and route the B_i units on this path P_i. Again, a Chernoff bound can be used to prove that the node congestion is $O(\log n)$ with high probability. So, this step can also be carried out relative to an optimal splittable routing.

A.3. Reducing edge-costs to node-costs. Here, we observe that the node version NEERP is more general than the edge-version studied previously [3, 10, 7]. Consider an instance of energy-efficient routing with edge energy costs (1.1), graph $G = (V, E)$ having edge-costs $\{c_e\}_{e \in E}$, and requests $\{(s_i, t_i, d_i)\}_{i=1}^k$. We define an NEERP instance on the graph G' obtained by subdividing each edge $e \in E$ with a node v_e. The node costs are zero for all nodes of V and c_e for each node v_e (for $e \in E$). This NEERP instance is clearly equivalent to the original edge version.

A similar reduction shows that edge-capacitated MCNC is a special case of the node-capacitated problem studied in this paper.

Appendix B. Missing details from section 3.

B.1. Approximation algorithm for small q. Here, we provide a bicriteria approximation algorithm for MCNC when q is small. The idea is essentially the same as that used in Theorem 3.15. Given an MCNC instance $\mathcal{I}$, consider a new instance $\mathcal{I}'$ where every demand equals 1 and the goal is to select $s_i - t_i$ paths P_i with node-congestion at most q that minimizes the *sum* of all path costs, i.e., $\sum_{i=1}^k \sum_{v \in P_i} c_v$. Note that each node may be counted multiple times in this objective. Using an optimal solution to $\mathcal{I}$ as a feasible solution to $\mathcal{I}'$, we have $\mathsf{Opt}(\mathcal{I}') \leq q \cdot \mathsf{Opt}(\mathcal{I})$. We now write an LP relaxation for $\mathcal{I}'$ that is just (LP_h) used in Theorem 3.15, where $\mathcal{M} = [k]$ and the right-hand side in constraint (3.2) is q. It is clear that this is a valid relaxation. The rounding algorithm is the same as that described in Theorem 3.15. The same analysis implies that we obtain a solution to $\mathcal{I}'$ of cost $O(\log n) \cdot \mathsf{Opt}(\mathcal{I}') \leq O(q \log n) \cdot \mathsf{Opt}(\mathcal{I})$ and node-congestion $O(q \log n)$. Using this as a solution to $\mathcal{I}$, the cost remains $O(q \log n) \cdot \mathsf{Opt}(\mathcal{I})$ and the node-congestion increases by at most a factor q. So we obtain an $(O(q \log n), O(q \log n))$ bicriteria approximation algorithm for MCNC.

We remark that the case when every demand is large, i.e., $\min_i d_i = \Omega(q)$, can also be solved by this approach. We just uniformly scale all demands and capacity by $\min_i d_i$ so that the new capacity $q' = O(1)$. Then, we obtain an $(O(\log n), O(\log n))$ bicriteria approximation algorithm for MCNC with large demands.

B.2. Ensuring polynomially bounded demands. Here, we show that the total demand $D = \sum_{i=1}^k d_i$ can be ensured to be polynomial in the number of requests k. Let $D_1 \subseteq [k]$ denote all requests with demand at least q/k, and $D_2 = [k] \setminus D_1$. We will handle the requests in D_1 and D_2 separately. We round up the demand d_i of each $i \in D_1$ to an integer multiple of q/k, which increases each demand by at most a factor two. Then, scaling all demands down by q/k, we obtain an equivalent MCNC instance with capacity $q' = O(k)$, which implies that total demand in this instance is $D' = O(k^2)$. For requests in D_2, we just use the minimum node-weighted Steiner forest, which admits an $O(\log k)$-approximation algorithm [36].

REFERENCES

[1] A. Agrawal, P. N. Klein, and R. Ravi, *When trees collide: An approximation algorithm for the generalized Steiner problem on networks*, SIAM J. Comput., 24 (1995), pp. 440–456, https://doi.org/10.1137/S0097539792236237.

[2] M. Andrews, *Hardness of buy-at-bulk network design*, in FOCS, 2004, pp. 115–124.

[3] M. Andrews, S. Antonakopoulos, and L. Zhang, *Minimum-cost network design with (dis)economies of scale*, SIAM J. Comput., 45 (2016), pp. 49–66, https://doi.org/10.1137/110825959.

[4] M. Andrews, A. Fernández, L. Zhang, and W. Zhao, *Routing for energy minimization in the speed scaling model*, in INFOCOM, 2010, pp. 2435–2443.

[5] M. Andrews and L. Zhang, *Hardness of the undirected congestion minimization problem*, SIAM J. Comput., 37 (2007), pp. 112–131, https://doi.org/10.1137/050636899.

[6] S. Antonakopoulos, C. Chekuri, F. B. Shepherd, and L. Zhang, *Buy-at-bulk network design with protection*, Math. Oper. Res., 36 (2011), pp. 71–87.

[7] A. Antoniadis, S. Im, R. Krishnaswamy, B. Moseley, V. Nagarajan, K. Pruhs, and C. Stein, *Hallucination helps: Energy efficient virtual circuit routing*, SIAM J. Comput., 49 (2020), pp. 37–66.

[8] B. Awerbuch and Y. Azar, *Buy-at-bulk network design*, in FOCS, 1997, pp. 542–547.

[9] M. A. Babenko, A. V. Goldberg, A. Gupta, and V. Nagarajan, *Algorithms for hub label optimization*, ACM Trans. Algorithms, 13 (2016), 16.

[10] N. Bansal, A. Gupta, R. Krishnaswamy, V. Nagarajan, K. Pruhs, and C. Stein, *Multicast routing for energy minimization using speed scaling*, in MedAlg, 2012, pp. 37–51.

[11] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan, *Detecting high log-densities: An $O(n^{1/4})$ approximation for densest k-subgraph*, in STOC, 2010, pp. 201–210.

[12] D. Brooks, P. Bose, S. Schuster, H. M. Jacobson, P. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. V. Zyuban, M. Gupta, and P. W. Cook, *Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors*, IEEE Micro, 20 (2000), pp. 26–44.

[13] J. Byrka, F. Grandoni, T. Rothvoss, and L. Sanità, *Steiner tree approximation via iterative randomized rounding*, J. ACM, 60 (2013), 6.

[14] R. D. Carr, L. Fleischer, V. J. Leung, and C. A. Phillips, *Strengthening integrality gaps for capacitated network design and covering problems*, in SODA, 2000, pp. 106–115.

[15] D. Chakrabarty, C. Chekuri, S. Khanna, and N. Korula, *Approximability of capacitated network design*, Algorithmica, 72 (2015), pp. 493–514.

[16] D. Chakrabarty, R. Krishnaswamy, S. Li, and S. Narayanan, *Capacitated network design on undirected graphs*, in APPROX, 2013.

[17] T. Chakraborty, J. Chuzhoy, and S. Khanna, *Network design for vertex connectivity*, in STOC, 2008, pp. 167–176.

[18] C. Chekuri, M. T. Hajiaghayi, G. Kortsarz, and M. R. Salavatipour, *Approximation algorithms for non-uniform buy-at-bulk network design*, in FOCS, 2006, pp. 677–686.

[19] C. Chekuri, M. T. Hajiaghayi, G. Kortsarz, and M. R. Salavatipour, *Approximation algorithms for node-weighted buy-at-bulk network design*, in SODA, 2007, pp. 1265–1274.

[20] J. Chen, R. D. Kleinberg, L. Lovász, R. Rajaraman, R. Sundaram, and A. Vetta, *(Almost) tight bounds and existence theorems for single-commodity confluent flows*, J. ACM, 54 (2007), 16.

[21] J. Chuzhoy and S. Khanna, *An $o(k^3 \log n)$-approximation algorithm for vertex-connectivity survivable network design*, Theory Comput., 8 (2012), pp. 401–413.

[22] A. M. Costa, *A survey on benders decomposition applied to fixed-charge network design problems*, Comput. Oper. Res., 32 (2005), pp. 1429–1450.

[23] Y. Dinitz, N. Garg, and M. X. Goemans, *On the single-source unsplittable flow problem*, Combinatorica, 19 (1999), pp. 17–41.

[24] Y. Emek, S. Kutten, R. Lavi, and Y. Shi, *Approximating generalized network design under (dis)economies of scale with applications to energy efficiency*, J. ACM, 67 (2020), 7.

[25] U. Feige, G. Kortsarz, and D. Peleg, *The dense k-subgraph problem*, Algorithmica, 29 (2001), pp. 410–421.

[26] W. S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi, *A general framework for graph sparsification*, in STOC, ACM, 2011, pp. 71–80.

[27] M. X. Goemans and D. P. Williamson, *A general approximation technique for constrained forest problems*, SIAM J. Comput., 24 (1995), pp. 296–317, https://doi.org/10.1137/S0097539793242618.

[28] S. Guha and S. Khuller, *Improved methods for approximating node weighted Steiner trees and connected dominating sets*, Inform. and Comput., 150 (1999), pp. 57–74.
[29] S. Guha, A. Meyerson, and K. Munagala, *A constant factor approximation for the single sink edge installation problem*, SIAM J. Comput., 38 (2009), pp. 2426–2442, https://doi.org/10.1137/050643635.
[30] A. Gupta, A. Kumar, M. Pál, and T. Roughgarden, *Approximation via cost sharing: Simpler and better approximation algorithms for network design*, J. ACM, 54 (2007), 11.
[31] M. Hajiaghayi, R. Khandekar, G. Kortsarz, and Z. Nutov, *On fixed cost k-flow problems*, Theory Comput. Syst., 58 (2016), pp. 4–18.
[32] M. Hewitt, G. L. Nemhauser, and M. W. P. Savelsbergh, *Branch-and-price guided search for integer programs with an application to the multicommodity fixed-charge network flow problem*, INFORMS J. Comput., 25 (2013), pp. 302–316.
[33] K. Jain, *A factor 2 approximation algorithm for the generalized Steiner network problem*, Combinatorica, 21 (2001), pp. 39–60.
[34] D. R. Karger, *Random sampling in cut, flow, and network design problems*, Math. Oper. Res., 24 (1999), pp. 383–413.
[35] D. Kim and P. M. Pardalos, *A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure*, Oper. Res. Lett., 24 (1999), pp. 195–203.
[36] P. Klein and R. Ravi, *A nearly best-possible approximation algorithm for node-weighted Steiner trees*, J. Algorithms, 19 (1995), pp. 104–115.
[37] J. Könemann, S. S. Sadeghabad, and L. Sanità, *An LMP $O(\log n)$-approximation algorithm for node weighted prize collecting Steiner tree*, in FOCS, 2013.
[38] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, Addison-Wesley, Boston, 2009.
[39] R. Lee, D. Pinner, K. Somers, and S. Tunuguntla, *The Case for Committing to Greener Telecom Networks*, McKinsey Report, McKinsey & Company, 2020.
[40] N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15 (1995), pp. 215–245.
[41] K. Makarychev and M. Sviridenko, *Solving optimization problems with diseconomies of scale via decoupling*, J. ACM, 65 (2018), 42.
[42] P. Manurangsi, *Almost-polynomial ratio ETH-hardness of approximating densest k-subgraph*, in Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, 2017, pp. 954–961.
[43] A. Moss and Y. Rabani, *Approximation algorithms for constrained node weighted Steiner tree problems*, SIAM J. Comput., 37 (2007), pp. 460–481, https://doi.org/10.1137/S0097539702420474.
[44] V. Nagarajan and L. Wang, *Online generalized network design under (dis)economies of scale*, Math. Oper. Res., 49 (2024), pp. 107–124.
[45] Z. Nutov, *Approximating minimum-cost connectivity problems via uncrossable bifamilies*, ACM Trans. Algorithms, 9 (2012), 1.
[46] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.
[47] D. A. Spielman and S.-H. Teng, *Spectral sparsification of graphs*, SIAM J. Comput., 40 (2011), pp. 981–1025, https://doi.org/10.1137/08074489X.
[48] A. Wierman, L. L. H. Andrew, and A. Tang, *Power-aware speed scaling in processor sharing systems*, in INFOCOM, 2009, pp. 2007–2015.