Cluster Before You Hallucinate: Approximating Node-Capacitated Network Design and Energy Efficient Routing

Ravishankar Krishnaswamy* Viswanath Nagarajan† Kirk Pruhs‡ Cliff Stein§

November 12, 2013

Abstract

We consider circuit routing with an objective of minimizing energy, in a network of routers that are speed scalable and that may be shutdown when idle. We consider both multicast routing and unicast routing. It is known that this energy minimization problem can be reduced to a capacitated flow network design problem, where vertices have a common capacity but arbitrary costs, and the goal is to choose a minimum cost collection of vertices whose induced subgraph will support the specified flow requirements. For the multicast (single-sink) capacitated design problem we give a polynomial-time algorithm that is $O(\log^3 n)$-approximate with $O(\log^4 n)$ congestion. This translates back to a $O(\log^{4\alpha+3} n)$-approximation for the multicast energy-minimization routing problem, where α is the polynomial exponent in the dynamic power used by a router. For the unicast (multicommodity) capacitated design problem we give a polynomial-time algorithm that is $O(\log^5 n)$-approximate with $O(\log^{12} n)$ congestion, which translates back to a $O(\log^{12\alpha+5} n)$-approximation for the unicast energy-minimization routing problem.

*Computer Science, Princeton University. Supported by the Simons Postdoctoral Fellowship.

†IBM T.J. Watson Research Center

‡University of Pittsburgh. Supported in part by NSF grants CCF-1115575, CNS-1253218 and an IBM Faculty Award.

§Columbia University. Supported in part by NSF grants CCF-0915681 and CCF-1349602.

1 Introduction

Data networks consume large amounts of energy, and reducing this energy usage is an important problem. According to the US Department of Energy [1], data networks consume more than 50 billion kWH of energy per year, and a 40% reduction in wide-area network energy is possibly achievable if network components were energy proportional. Circuit routing, in which each connection is assigned a fixed route in the network, is used by several network protocols to achieve reliable communication [31].

Motivated by this, we consider circuit routing protocols with an objective of minimizing energy, in a network of routers that (i) are speed scalable, and (ii) may be shutdown when idle. We use the standard models for circuit routing and component energy, in particular these are the same as used in [3,4,6,10]. In the **Energy Efficient Vertex Routing Problem** (EEVRP), the input consists of an undirected multi-graph $G = (V, E)$, with $|V| = n$, $|E| = m$, and a collection of k request-pairs $\{(s_i, t_i) \mid s_i, t_i \in V \text{ and } i \in [k]\}$. The output is set of paths P_i, each representing the circuit for a unit bandwidth demand, between vertices s_i and t_i, for request-pair $i \in [k]$. We make the standard assumption that the power used by a router/vertex with load f (the number of request-pairs which route their unit flow through the vertex) is $\sigma + f^\alpha$ if $f > 0$, and that the router is shutdown and consumes no power if its load is zero. The objective is to minimize the aggregate power used over all the routers. We also consider the single sink version MEEVRP, which corresponds to multicast routing where there is a common sink t s.t for every i, $t_i = t$. As in prior works, the term f^α is the dynamic power of the component as it varies with the load, or equivalently the speed with which the router must be run. Here $\alpha > 1$ is a parameter specifying the energy inefficiency of the components, as speeding up by a factor of s increases the energy used per unit computation/communication by a factor of $s^{\alpha-1}$. The value of α is in the range $[1.1, 3]$ for essentially all technologies [12,35]. The parameter σ is the static power, which is the base power required to keep the router turned on, and that can only be saved by turning the router off.

The problems EEVRP and MEEVRP have been previously studied in the case that speed scaling occurred on the edges instead of the vertices [3,4,6,10]. Although speed scalable edges are plausible, it is more likely that speed scaling occurs at the routers/vertices. Presumably, the assumption in [3,4,6,10] 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 vertex costs. Indeed, the edge problems studied earlier are all special cases of the EEVRP problem we study.

To understand the difficulty in handling an energy function that is neither concave or convex, and to survey prior results, let us assume for the moment that speed scaling occurs on the edges. First consider the case that the static power σ is zero. In this case, the objective function becomes convex, and one could simply write a convex program and perform a randomized rounding. Intuitively, the convexity of the power function implies that the optimal solution would spread the flows out as much as possible over the whole network. In fact, [7] shows that the natural greedy algorithm, which routes each new request in the cheapest possible way, is an $O(1)$-competitive online algorithm with respect to the dynamic energy cost. Subsequently, [24] showed how to use convex duality to obtain the same result. On the other hand, if the static power is very large ($\sigma \gg k^\alpha$), then the optimal solution simply routes all flow using a minimum cardinality Steiner forest connecting corresponding request-pairs (since this minimizes static power); that is, the flow should be as concentrated as possible. The difficulty in the standard energy function comes from these competing goals of minimizing static power, where it's best that flows are concentrated, and dynamic power, where it's best that the flows are spread out. In fact, [4] showed that there is a limit to how well these competing objectives can be balanced by showing an $\Omega(\log^{1/4} n)$ inapproximability result for even the edge version, under standard complexity theoretic assumptions. On the positive side, [3] showed that these competing forces can be "poly-log-balanced" by giving an efficient poly-log approximation algorithm for the edge version of EEVRP. Subsequently, [10] gave a polynomial-time $O(1)$-approximation algorithm and an $O(\log^{2\alpha+1} n)$-competitive randomized online algorithm for the edge version of MEEVRP. Recently, [6] gave a simple combinatorial algorithm for the edge version of EEVRP that is $O(\log^\alpha n)$-approximate, and that naturally extends to an $\tilde{O}(\log^{3\alpha+1} n)$-competitive online algorithm.

To the best of our knowledge, previous algorithms do not handle the setting when the nodes are the speed-scalable elements. In this paper, we obtain the first poly-log approximation algorithms for this class of problems.

Theorem 1 *There is a polynomial-time $O(\log^{4\alpha+3} n)$-approximation algorithm for the multicast energy routing problem MEEVRP.*

Theorem 2 *There is a polynomial-time $O(\log^{12\alpha+5} n)$-approximation algorithm for the unicast energy routing problem EEVRP.*

It is known that one can reduce EEVRP to the following network design problem for flows (the reduction can be found in [3], but for completeness we give the reduction in Appendix A): In the **Multicommodity Node-Capacitated Network Design Problem** (MCNC), the input consists of an undirected graph $G = (V, E)$, with $|V| = n$, $|E| = m$, and a collection of k request-pairs $\{(s_i, t_i) \mid s_i, t_i \in V \text{ and } i \in [k]\}$. Each vertex $v \in V$ has a cost c_v and capacity q. The output is a subset of nodes $V' \subseteq V$ such that the graph $G[V']$ induced by the vertices V' can support one unit of flow between vertices s_i and t_i concurrently for each request-pair $i \in [k]$. The objective is to minimize the total cost $c(V') = \sum_{v \in V'} c(v)$ of the output graph. Our algorithms will find bi-criteria approximations, ones in which we allow the algorithm to violate the capacity constraints by some amount. We also consider the corresponding single sink version of the problem called SSNC. When combined with the reduction in [3], Theorems 1 and 2 follow from the following theorems.

Theorem 3 *There is a polynomial-time for the single sink problem SSNC that is $O(\log^3 n)$-approximate with respect to cost with $O(\log^4 n)$ congestion.*

Theorem 4 *There is a polynomial-time for the multicommodity problem MCNC that is $O(\log^5 n)$-approximate with respect to cost with $O(\log^{12} n)$ congestion.*

To understand the difficulty of extending the algorithms for the edge version of these problems to the node version, let us consider the single sink problem SSNC where there are k sources with each source having unit demand. Roughly, the algorithms in [3,6,10] all choose an *approximate Steiner tree* T connecting all the sources and the sink. They then choose a set of roughly k/q "leaders", and find a minimum cost subgraph H which can send q flow from the leaders to the sink (which is basically the standard min-cost flow problem). Finally they route every demand from its source to a leader using T (without congesting any edge too much), and then use H to route from the leaders to the sink. The main difficulty in emulating this approach for the node problems is that a low node congestion routing from the sources to the leaders may not even exist on the Steiner tree T, e.g., T is a star with the sources and sink as leaves. To surmount this difficultly we will show how to efficiently find a low-cost collection of *nearly node-disjoint clusters* that span all terminals; this can then be used to obtain an aggregation of flows with low vertex congestion. A priori, it is not even immediate that such a clustering should exist. We give an overview of these ideas and our techniques in Section 1.2.

1.1 Additional Related Results

Beyond the prior literature on which we explicitly build (which we surveyed in the introduction), there are several other results in the network design literature related to our work.

There is a significant literature on node weighted problems in Steiner tree and network design, beginning with [29], who gave a logarithmic factor approximation algorithm for node-weighted Steiner Tree. A crucial building block in our algorithm will be an $O(\log n)$-approximation algorithm for the *partial node weighted Steiner tree* problem (PNWST). In this problem, we are given a node-weighted graph, a subset of the nodes labeled as terminals, and a target k, and want to find the minimum node cost Steiner tree that contains at least k terminals. [30] give a polynomial-time, Lagrangian multiplier preserving, $O(\log n)$-approximation algorithm for the Node Weighted Prize Collecting Steiner Tree Problem (here Lagrangian multiplier preserving means that the approximation is only in the cost, not in the penalty term for excluding terminals). An $O(\log n)$-approximation algorithm for PNWST problem then follows from a reduction given in [33].

Another related framework is that of *buy-at-bulk* network design, where the cost on a network element is a concave function of the load through it. Here again, there were several works focusing on the edge-case [8,17,23] before the node-cases were understood [5,18]. [2] showed poly-logarithmic hardness of approximation for the edge-versions of uniform and non-uniform buy-at-bulk network design. Lower bounds on these edge-weighted problems apply easily to the node-weighted versions. From a technical standpoint, the hallucination idea used in [6] and also in our algorithm, is rather similar to the Sample-Augment framework [25] for solving Buy-at-Bulk problems. However, our algorithm analysis is quite different than those for Buy-at-Bulk, and is more similar in spirit of the analysis of cut-sparsification algorithms [22,28,34].

A generalization of Steiner tree is *survivable network design*, where the goal is to design a minimum-cost network that can route a set of demands. This problem differs crucially from ours in that the goal in survivable network design is to install enough capacity so that each demand can be routed in isolation, whereas our objective is to have enough capacity to route all demands simultaneously. The best result for this problem for edge-connectivity is a 2-approximation algorithm [27]. There are also many algorithms for the vertex-connectivity variant, with the best known bound being an $O(k^3 \log n)$-approximation [19]; here k is the largest demand. Unlike the edge survivable network design, the vertex version is $\Omega(k^\epsilon)$-hard to approximate [16]. There has also been recent focus on the capacitated versions of these problems [13–15,26].

1.2 Overview of Technical Results

We first develop an algorithm for the single-sink node-capacitated network design problem (SSNC). This will serve as a warm-up in understanding the difficulties posed by node capacities, and it will also be used as a sub-routine for the multicommodity case. As mentioned in the introduction, our approach is to find a nearly node-disjoint and low-cost collection of clusters (i.e., trees) where each cluster contains approximately q sources. Once the flow is aggregated within the clusters, we route the demand from the roots of these clusters to the sink, and this is a simple min-cost flow problem since demands are equal to node capacities.

Our first step towards the clustering is to show that there *exists* one with $O(\log^2 n)$ congestion and cost at most the optimal SSNC cost. Our existence proof starts with the optimal unsplittable flow $\mathcal{F}^*$, and repeatedly finds a *first-merge* node v, which is the deepest node where there are two incoming flow arcs into v. If at least q units of flow aggregate at v, then these flows give us a cluster. Otherwise, if $d < q$ units of flow aggregate at v, then we form a partial cluster (of demand d), and remove the flows from the original sources in this cluster to v. But this leaves us with a splittable flow from v to t carrying d units of flow. In order to proceed with the clustering, we now make v a source (with demand d) for which $\mathcal{F}^*$ is a feasible splittable flow. We can then *convert this into an unsplittable flow* using the unsplittable-flow algorithm [20], which additively increases the load on any vertex by at most q. We keep repeating this process until we are left with clusters of total demand of roughly q. Moreover, we do this in a way such that we invoke the [20] algorithm only $O(\log q)$ times, so the overall increase in the load on any node is bounded. Once we know the existence of such a clustering, we can efficiently compute one which is $O(\log^3 n)$-approximate on the cost with $O(\log^4 n)$ congestion, using the *low load set cover* framework [9] combined with the $O(\log n)$-approximation algorithm for the *partial node weighted Steiner tree problem* [30,33].

We now turn to our algorithm for the multi-commodity problem MCNC. Again the first step is to find clusters that aggregate $\Omega(q)$ demands that are both low-cost and low-congestion. However, since the optimal flow is not directed (earlier, it was directed from the sources to the sink), we don't have a meaningful starting point to merge these flows into clusters. Our algorithm surmounts this issue by *repeatedly generating and solving instances of the single sink problem!* Indeed, given a set of clusters, we connect artificial sources to some of these clusters, and connect an artificial sink to some other clusters, and ask for a solution to the resulting SSNC instance. The crux is to choose these clusters and connections carefully to ensure that the SSNC instance (a) has a low cost routing, and (b) helps us make progress in our clustering. While this is easy initially (set all s_i vertices as sources, and connect t to each t_i vertex), this is the main challenge in a general iteration. We show that our SSNC instances have low cost by producing a witness using OPT for the original multicommodity instance. To make progress, we use the directed nature of our SSNC routing to merge clusters à la the single-sink setting. Finally, this entire process is repeated $O(\log n)$ times to get big enough clusters. We remark that it appears difficult to bypass the use of SSNC instances in the multicommodity clustering, since optimizing directly for the "best" cluster turns out to be as hard as the *dense-k-subgraph* problem [11,21]. The complete clustering algorithm becomes much more complicated than the single-sink setting, and we will only be able to cluster a constant fraction of the demands.

After this step, we run the "hallucination" algorithm like in the edge version [6]. Each request-pair hallucinates its demand to be q instead of 1 with probability $\log n/q$, and we find the minimum cost subgraph H which can route the hallucinated demand with low node congestion. Since all demands and capacities are equal, we can simply solve an LP and do a randomized rounding to find such paths for the hallucinated demands. Our contribution here is to show that the *union of the clusters and subgraph H is sufficient* to route a constant fraction of the demands. This proof uses several new ideas on top of those used in the edge-case [6].

2 Single-Sink Node-Capacitated Network Design

The input to the single-sink node-capacitated network design problem (SSNC) consists of an undirected graph $G = (V, E)$, with $|V| = n$, and a collection of k sources $\mathcal{D} = \{s_i \mid i \in [k]\}$ with respective demands $\{d_i \geq 1 \mid i \in [k]\}$. There is a specified sink $t \in V$ to which each source s_i wants to send d_i units of flow unsplittably. Each vertex $v \in V \setminus \{t\}$ has a cost $c(v)$ and uniform capacity q; the sink t is assumed to have zero cost and infinite capacity [1]. We assume that each demand is at most q (otherwise the instance is infeasible). The output is a subset of nodes $V' \subseteq V$ such that the graph $G[V']$ induced by the vertices 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 $c(V') = \sum_{v \in V'} c(v)$ of the output graph. We will also refer to the vertices $\{s_i \mid i \in [k]\}$ as *terminals*.

2.1 Roadmap

Our algorithm for the single-sink special case serves as an important subroutine for the multicommodity problem. It also brings out some crucial issues that need to be dealt with in the node-capacitated setting (as opposed to edge-capacitated). Resolving these details for the single-sink case serves as a warm-up for the more complicated multicommodity algorithm. The algorithm works in two phases which are described below:

Clustering Phase. A *cluster* is a subtree T_i of G together with a set of assigned sources D_i that lie within T_i. The *total demand assigned to cluster* T_i is then $\sum_{s_i \in T_i} d_i$, and the cost of the cluster is $c(T_i) = \sum_{v \in T_i} c_v$ We want to find a collection of nearly node-disjoint clusters, each with roughly q assigned sources; using this we aggregate all demands of a cluster at one of its assigned sources. An important step along the way is to show the existence of such clusters, which we do in Section 2.2. The existence argument is based on an iterative application of the *single-sink unsplittable flow* algorithm DGG [20]. We then give an algorithm for finding such clusters in Subsection 2.3. This algorithm relies (in a black-box fashion) on two other results: an $O(\log n)$-approximation algorithm for *partial node-weighted Steiner tree* [30,33], 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 each set of terminals connected by a tree T is a set (of cost $\sum_{v \in T} c_v$), and the goal is to find a minimum cost set cover of all terminals such that no vertex is in too many sets. The algorithm of [9] requires a *max-density* oracle, for which we use the partial node-weighted Steiner tree algorithm as a subroutine.

Routing Phase. Once we find such a clustering, we can route all the demand from the sources to the roots of each cluster. The final step is to then route $\Theta(q)$ flow from each cluster root to the sink t. We reduce this problem to a standard minimum cost network flow problem (using the fact that demands and capacities are equal).

DGG Algorithm. We will repeatedly use the following algorithm for unsplittable flows in this paper. Given a directed node-capacitated graph, a set X of sources with demands $\{\tilde{d}(s) : s \in X\}$, a sink vertex t, node capacities $\{f_v\}$, and a splittable routing $\mathcal{F}$ for all demands, the DGG algorithm efficiently constructs an *unsplittable* flow $\mathcal{F}'$ that routes $d(s)$ units of flow from each source s to t along one of the paths used by the splittable flow $\mathcal{F}$, and $\mathcal{F}'$ respects node capacities $\{f_v + \max_{s \in X} \tilde{d}(s)\}$.

2.2 Existence of Good Clustering

The main result in this section is the following lemma.

Lemma 5 *There exists a collection of clusters $\{T_i\}$ such that*

(i) Each cluster T_i contains $O(q \cdot \log q)$ assigned sources.

(ii) Each cluster T_i with $t \notin T_i$ contains at least q assigned sources.

(iii) Every source lies in some cluster T_i.

(iv) Every node in $V \setminus \{t\}$ is contained in $O(\log q)$ clusters.

(v) The total cost $\sum_i \sum_{v \in T_i} c_v = O(\log q) \cdot c(\mathsf{Opt})$.

[1] Assigning the sink zero cost only makes approximation harder. Also if the sink had capacity of q, we would be limited to solving only problems with total demand at most q.

(vi) The total cost $\sum_{v \in \bigcup T_i} c_v \leq c(\mathsf{Opt})$.

Proof: Let V^* denote the set of nodes in an optimal solution, $\mathcal{F}^*$ denote an optimal flow for the sources $\mathcal{D}$, and $\mathcal{F}^*_v \leq q$ denote the flow through each node $v \in V^*$ in this solution. Clearly, the node capacities $\{\mathcal{F}^*_v\}$ suffice to send d_i units from each source s_i to the sink t. Since this is an instance of single-commodity flow[2], we may assume that $\mathcal{F}^*$ is a directed acyclic flow that (possibly splittably) routes d_i units from each source s_i to the sink t under node capacities $\{\mathcal{F}^*_v\}$. Moreover, we can use the DGG algorithm on this flow $\mathcal{F}^*$ to make it *an unsplittable flow* sending d_i units from s_i to t. Now the total flow through each node in V^* might be $2q$. Let D^* denote the directed acyclic graph on vertices V^* containing the arcs used in this unsplittable acyclic flow $\mathcal{F}^*$. Henceforth, we shall slightly abuse notation and let $\mathcal{F}_v$ denote the total flow through node v in a flow $\mathcal{F}$, and $\mathcal{F}(s)$ denote the flow-paths from a source s to the sink t. If the flow is unsplittable, then $\mathcal{F}(s)$ denotes a single s-t path. We now give a recursive procedure to construct the desired clusters $\{T_i\}$. In this process, we solve many flow subproblems, all of which will be defined and supported on D^*. The procedure to construct clusters $\{T_i\}$ is given as Algorithm 2 below. Algorithm Cluster $(X, d, \mathcal{F}, \{T(s) : s \in X\})$ takes as input:

1. Set X of "source vertices" (not necessarily the original terminals).
2. Demands $\{\tilde{d}(s) : s \in X\}$, which may again be different from the original demands of the terminals.
3. Unsplittable flow $\mathcal{F}$ that for each $s \in X$, sends $\tilde{d}(s)$ units of flow on path $\mathcal{F}(s)$ from s to t. Flow $\mathcal{F}$ will always be supported on the directed acyclic graph D^*.
4. For each $s \in X$, a tree $T(s)$ containing s.

The initial call is with the original sources and demands: $X = \{s_i : i \in [k]\}$ and $\tilde{d}(s_i) = d_i$, unsplittable flow $\mathcal{F}^*$ from above, and singleton trees $T(s_i) = \{s_i\}$ for $i \in [k]$. The high-level idea of Cluster is the following: Given $X, \tilde{d}$ and $\mathcal{F}$, we use the acyclic nature of $\mathcal{F}$ to find a node-disjoint collection of trees $\{\tau_v\}$ that collectively span all vertices in X. Some of these trees have total demand at least q (or contain the sink t); these trees are added to the output set since they satisfy condition (ii). The other trees (corresponding to $\mathcal{C}$) have less than q total demand; Cluster recurses on these trees, using the roots of each tree as a new source, with demand equal to the total demand within the tree. To aid in the recursive clustering, we recompute an unsplittable flow $\mathcal{F}'$ from the new sources using the original flow $\mathcal{F}$.

Algorithm 1 Cluster $(X, \tilde{d}, \mathcal{F}, \{T(s) : s \in X\})$

set $Y \leftarrow X, \mathcal{C} \leftarrow \emptyset, \mathcal{O} \leftarrow \emptyset, X' \leftarrow \emptyset$.
while $Y \neq \emptyset$ **do**
  let v be the deepest node in D^* with at least two incoming edges carrying non-zero flow in $\mathcal{F}$. If there is no such node, $v \leftarrow t$.
  let $S_v \subseteq Y$ be the sources whose flow-paths meet at v, and let τ_v be the tree containing the prefix of paths $\{\mathcal{F}(s) : s \in S_v\}$ until node v.
  set tree $T(v) := \cup_{s \in S_v} T(s) \cup \{\tau_v\}$.
  if $(\sum_{s \in S_v} \tilde{d}(s) \geq q$ **or** $t \in T(v))$ **then add** $T(v)$ to $\mathcal{O}$, the *output* set of trees[3].
  else add $T(v)$ to $\mathcal{C}$, the set of small clusters; **add** v to X' and **set** demand $\tilde{d}'(v) \leftarrow \sum_{s \in S_v} \tilde{d}(s)$.
  remove v and the sources in S_v. Set $Y \leftarrow Y \setminus S_v$ and $\mathcal{F} \leftarrow \mathcal{F} \setminus \{\mathcal{F}(s) : s \in S_v\}$.
end while
find unsplittable flow $\mathcal{F}'$ with sources X' and demands $\{\tilde{d}'(v)\}_{v \in X'}$ using the DGG algorithm on $\mathcal{F}$.
return $\mathcal{O} \cup$ Cluster $(X', \tilde{d}', \mathcal{F}', \mathcal{C})$ if $\mathcal{C} \neq \emptyset$.
if $(\mathcal{C} \neq \emptyset)$ **then return** $\mathcal{O} \cup$ Cluster $(X', \tilde{d}', \mathcal{F}', \mathcal{C})$ **else return** $\mathcal{O}$.

We now prove that the output trees satisfy all five conditions in Lemma 5. By definition of Cluster, each terminal initially lies in a cluster, and clusters only get merged over time. Therefore, it is easy to see that each

[2] We can add a super-source and attach it to the real sources with capacity d_i and require $\sum_i d_i$ flow from s to t.
[3] If $t \in T(v)$, then we break the cluster up into ones of size $O(q \cdot \log q)$ and add those to $\mathcal{O}$.

terminal lies in some output tree, i.e. condition (iii) holds. Moreover, the total demand of every output tree is at least q (or it contains the sink t), since these are the only times we include a tree in the output clustering; so condition (ii) also holds. To prove the remaining properties, we state some useful claims.

Claim 6 *In any call of Cluster, the trees $\{\tau_v\}$ are node-disjoint. Hence, the nodes added to trees $\{T(s)\}$ are disjoint, and have total cost at most $c(\mathsf{Opt})$.*

Proof: Consider any iteration of the while loop. Since the unsplittable flow $\mathcal{F}$ is acyclic (it is supported on D^*), the notion of "deepest node" v is well-defined. Clearly, the flow-paths $\{\mathcal{F}(s) : s \in S_v\}$ until the deepest merging point v are disjoint from all remaining flow-paths $\{\mathcal{F}(s) : s \in Y \setminus S_v\}$. That is, the new tree τ_v found in any iteration is disjoint from the remaining flow $\mathcal{F}$ (and hence from all other trees in this recursive call).

The nodes added to the trees $\{T(s)\}$ in any call of Cluster are precisely those of $\{\tau_v\}$. Since these are node disjoint, the increase in total cost is at most $\sum_{v \in V^*} c_v = c(\mathsf{Opt})$. □

Claim 7 *The number of calls to Cluster is at most $\log_2 q$.*

Proof: We will show that in each call to Cluster, the minimum new demand $\min_{a \in X'} \tilde{d}'(a)$ is at least twice the minimum old demand $\min_{s \in X} \tilde{d}(s)$. This would imply that the minimum demand after j recursive calls is at least 2^j. Note that any tree with at least q demand is output immediately and can not be part of the residual set $\mathcal{C}$. So after $\log_2 q$ recursive calls, the set $\mathcal{C}$ would be empty. This would prove the claim. Indeed, consider any "deepest node" v that is found in some iteration of the while-loop. If $v = t$ then the corresponding tree is immediately output (it satisfies condition (ii)), and $t \notin X'$. If $v \neq t$, then by definition, $|S_v| \geq 2$ flow-paths merge at v; so $\tilde{d}'(v) = \sum_{s \in S_v} \tilde{d}(s) \geq 2 \cdot \min_{s \in X} \tilde{d}(s)$. Thus $\min_{a \in X'} \tilde{d}'(a) \geq 2 \cdot \min_{s \in X} \tilde{d}(s)$. □

Claim 8 *The unsplittable flow $\mathcal{F}$ in the j^{th} recursive call to Cluster uses node capacities at most $\{\mathcal{F}^*_v + j \cdot q\}$.*

Proof: We prove this by induction on j. This is true for $j = 1$, since the initial unsplittable flow uses capacities $\mathcal{F}^*_v + \max_i d_i \leq \mathcal{F}^*_v + q$. For the inductive step, suppose the input flow $\mathcal{F}$ to the j^{th} call of Cluster uses capacities $\mathcal{F}^*_v + jq$. We will show that the flow $\mathcal{F}'$ uses capacities at most $\mathcal{F}_v + q$. Indeed, observe that there is a *splittable* flow routing the demands $\{\tilde{d}'(v) : v \in X'\}$ under capacities $\mathcal{F}_v$: for each $v \in X'$, take the suffix of each flow-path $\{\mathcal{F}(s) : s \in S_v\}$ from v until the sink t. Cluster then runs the DGG algorithm to obtain an *unsplittable* flow $\mathcal{F}'$ for demands $\{\tilde{d}'(v) : v \in X'\}$; the capacities are at most $\mathcal{F}_v + \max_{v \in X'} \tilde{d}'(v) \leq \mathcal{F}_v + q$. □

We now complete the proof of Lemma 5. To see condition (i), using Claims 7 and 8 note that the maximum node capacity used by any flow $\mathcal{F}$ is $q + q \cdot \log_2 q = O(q \cdot \log q)$. Therefore, any "deepest node" v chosen in Cluster, has only $O(q \cdot \log q)$ flow through it. In other words, any output tree has $O(q \cdot \log q)$ total demand. Now, using Claims 6 and 7, it is clear that each node appears in at most $\log_2 q$ output trees, i.e. condition (iv) holds. Condition (vi) holds since every node used in the clusters was in the support of $\mathcal{F}^*$. And condition (v) then directly follows from condition (iv) and (vi). □

2.3 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. We use our knowledge of the existence of a good clustering to find the clustering itself. Our algorithm will use an approximation algorithm [9] for the *low load set cover* (LLSC) problem using a max-density oracle, for which we use an approximation algorithm [30,33] for the *partial node weighted Steiner tree* problem (PNWST), both of which are defined below:

Low load set cover problem (LLSC). In this problem, we are given a set system $(U, \mathcal{C})$, costs $\{c_v : v \in U\}$, and bound p. 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 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 the groundset: $W \subseteq U$ of *required* elements that need to be covered, and $L \subseteq U$ of *capacitated* elements. The goal is to find a minimum cost set cover $\mathcal{C}' \subseteq \mathcal{C}$ for the required elements W such that each capacitated element $e \in L$ appears in at most p sets of $\mathcal{C}'$. The algorithm of [9] uses a *max-density oracle*, which takes as input: costs $\{\beta_v : v \in U\}$ and subset $X \subseteq W$ (of already covered required elements), and outputs a set $S \in \mathcal{C}$ that minimizes $\frac{\sum_{e \in S} \beta_e}{|S \cap (W \setminus X)|}$.

Partial node weighted Steiner tree problem (PNWST). The input is a graph $G = (U, E)$ with node-weights $\{\beta_v : v \in U\}$, root $r \in V$, subset $W \subseteq U$ of terminals, and a target ℓ. The objective is to find a minimum node cost Steiner tree containing r and at least ℓ terminals.

Theorem 9 ([30,33]) *There is an $O(\log n)$-approximation algorithm for the PNWST problem.*

Theorem 10 ([9]) *Assuming a poly-time ρ-approximate max-density oracle, there is a poly-time $O(\rho \log |U|)$-approximation algorithm for the LLSC problem, that violates any capacity by an $O(\rho \log |U|)$ factor.*

The SSNC problem as LLSC. We now cast the clustering problem (see the properties from Lemma 5) as an instance of LLSC. The groundset $U := V \cup W$ where V is the vertex-set of the original SSNC problem, and $W = \{s_i(j) : 1 \leq j \leq d_i, i \in [k]\}$ consists of d_i (the demand of s_i) new elements for each source s_i in SSNC. For any source s_i, we refer to the d_i elements $\{s_i(j) : 1 \leq j \leq d_i\}$ as the W-elements of s_i. Note that the size $|U| \leq nq$. The costs c_v are the node-costs in SSNC; elements in W have zero cost. The bound p is set to $O(\log q)$. The required elements are all elements of W, and the capacitated elements are $L := V \setminus \{t\}$, all nodes of V except the sink. The sets in $\mathcal{C}$ are defined as follows. For each tree T in the original graph G, containing $O(q \log q)$ total demand and satisfying one the following:

- T contains at least q total demand, or
- T contains the sink t,

there is a set $T' \in \mathcal{C}$ consisting of all nodes of V in T along with the W-elements of all the sources in T. Note that Lemma 5 implies that the optimal value of this LLSC instance is $O(\log q) \cdot c(\mathsf{Opt})$.

Lemma 11 *There is an $O(\log n)$-approximate max-density oracle for the SSNC clustering problem.*

Proof: This is a straightforward reduction to the PNWST problem. In the max-density oracle of the clustering problem, we are given costs (node weights) $\{\beta_v : v \in U\}$ and subset $X \subseteq W$. The goal is to find a set $T' \in \mathcal{C}$ (corresponding to tree T in G) minimizing

$$\frac{\sum_{e \in T'} \beta_e}{|T' \cap (W \setminus X)|}.$$

Define a new graph $\widehat{G}$ on vertex set $V \cup W$, which consists of the original graph G (on vertices V) along with "pendant" edges $\{(s_i, s_i(j)) : 1 \leq j \leq d_i, i \in [k]\}$ between each source s_i and its W-elements. We consider separately the max-density problem for the two types of trees in $\mathcal{C}$.

- *T contains at least q total demands.* For each demand $q \leq \ell \leq O(q \log q)$ and root $r \in V \setminus \{t\}$, solve the PNWST on $\widehat{G}$ with node-weights $\{\beta_v\}$, root r, terminals $W \setminus X$ and target ℓ.
- *T contains the sink t.* For each demand $1 \leq \ell \leq O(q \log q)$, solve the PNWST instance on $\widehat{G}$ with node-weights $\{\beta_v\}$, root t, terminals $W \setminus X$ and target ℓ.

For all the above instances, we use the $O(\log n)$-approximation algorithm for PWNST [30,33]. Finally, we output the tree T' that minimizes the ratio of its cost to the number of terminals. □

We note that without loss of generality, any solution T' to the max-density problem contains all the W-elements of its sources: this is because the β-weight of all W elements remain zero throughout our algorithm for this LLSC instance. Combining this lemma with Theorem 10 we obtain:

Lemma 12 *For any SSNC instance, there is an efficient algorithm that finds a collection of clusters $\{T_i\}$ s.t*

(i) Each T_i contains $O(q \cdot \log q)$ total demand.

(ii) Each T_i with $t \notin T_i$ contains at least q total demand.

(iii) Every terminal lies in some tree T_i.

(iv) Every node in $V \setminus \{t\}$ appears in $O(\log q \cdot \log^2 n)$ trees.

(v) The total cost $\sum_i \sum_{v \in T_i} c_v \leq O(\log q \cdot \log^2 n) \cdot c(\mathsf{Opt})$.

Note that in the clustering above, a terminal may belong to upto $O(\log q \cdot \log^2 n)$ clusters and may be counted by each of these clusters towards their demands (to satisfy property (ii)). This is a subtle difference between the existence result (where each terminal was assigned to a unique cluster), but we cannot enforce that in our LSSC algorithm. However, this will not be an issue as we handle it in the next part.

2.4 Routing from Clusters to the Sink

We now have a clustering of sources into trees, each of which has total demand more than q (unless the tree contains the sink t). The final routing consists of two parts: aggregating all the demand in a tree at one "root" node in the tree, and routing from all roots to the sink t.

Aggregation at roots. For each tree T_i in our clustering from Lemma 12, choose an arbitrary node as its root $r(T_i)$; if $t \in T_i$ then we ensure $r(T_i) = t$. We route demands from all sources in T_i to $r(T_i)$. Note that the flow through any node of T_i is $O(q \log q)$ which is an upper bound on the total demand in T_i. Since each node of the graph appears in at most $O(\log q \cdot \log^2 n)$ trees, the total flow through any node of G is $O(\log^2 q \cdot \log^2 n) \cdot q$. Moreover, the total cost of nodes $\cup_i T_i$ used in this step is $O(\log q \cdot \log^2 n) \cdot c(\mathsf{Opt})$.

Routing from roots to sink. Note that all demands in trees s.t $t \in T_i$ are already routed to the sink t, since we ensured $r(T_i) = t$ for such trees. So the only trees that remain to be routed to the sink are those not containing t: by Lemma 12 each such tree has demand at least q. After scaling the capacity and demand by q, consider an instance $\mathcal{I}$ of *min-cost flow* on graph G with sink t with infinite capacity, sources $r(T_i)$ (for all remaining trees T_i) with unit demand, uniform node capacity $u := \Theta(\log q \cdot \log^2 n)$ for $u \in V \setminus \{t\}$, and node cost per unit flow $\{c_v : v \in V\}$. From the integrality of such instances, we get paths P_i from $r(T_i)$ to t in an optimal solution to $\mathcal{I}$. Then, we route the entire demand in tree T_i along path P_i to t. This routes all demands from roots $r(T_i)$ to the sink t. Since each tree has $O(q \log q)$ demand, and each node has capacity u in $\mathcal{I}$, the flow through any node of G is $O(q \log q) \cdot u = O(\log^2 q \cdot \log^2 n) \cdot q$. We now show that the total cost of nodes in $\cup_i P_i$ is also small.

Lemma 13 *The minimum cost flow in instance $\mathcal{I}$ is $O(\log q \cdot \log^2 n) \cdot c(\mathsf{Opt})$.*

Proof: We will show that there is a feasible fractional flow of the claimed cost; this suffices due to the integrality of the flow polytope. To this end, consider the following flow:

1. For each tree T_i, send $\frac{d(s)}{d(T_i)}$ flow from $r(T_i)$ to each source $s \in T_i$, along the tree T_i. Note that the total flow out of $r(T_i)$ is exactly one. So the flow through any node of T_i is at most one. Since each node of G appears in $O(\log q \cdot \log^2 n)$ trees, the net flow through any node is $O(\log q \cdot \log^2 n)$. The cost (in instance $\mathcal{I}$) of nodes used in this step is at most $\sum_i \sum_{v \in T_i} c_v \leq O(\log q \cdot \log^2 n) \cdot c(\mathsf{Opt})$.
2. Next, we use the optimal routing $\mathcal{F}^*$ scaled by $O(\log q \log^2 n)$ to send upto $\sum_{T:\, s \in T} \frac{d(s)}{d(T)} \leq O(\log q \log^2 n) \frac{d(s)}{q}$ flow from each source s to the sink t; this uses the fact that $d(T) \geq q$, and the fact that any vertex appears in at most $O(\log q \log^2 n)$ clusters T. This step sends flow of at most $O(\log q \log^2 n)$ through each node, and the cost of nodes (w.r.t instance $\mathcal{I}$) is at most $O(\log q \log^2 n) \cdot c(\mathsf{Opt})$.

Combining the flow from the above two steps, we obtain feasible fractional solution to $\mathcal{I}$, since the net flow through any node is $O(\log q \cdot \log^2 n) \leq u$. Moreover, the cost is $O(\log q \cdot \log^2 n) \cdot c(\mathsf{Opt})$. □

Combining the aggregation and routing steps, we obtain a solution to SSNC with cost $O(\log q \cdot \log^2 n) \cdot c(\mathsf{Opt})$, where each node supports $O(\log^2 q \cdot \log^2 n) \cdot q$ flow. This proves Theorem 3.

3 Multicommodity Node-Capacitated Network Design

We now discuss the general multicommodity case of the problem. The input consists of an undirected graph $G = (V, E)$, with $|V| = n$, and a collection of k request-pairs $\{(s_i, t_i) \mid s_i, t_i \in V \text{ and } i \in [k]\}$. Each vertex $v \in V$ has a cost c_v and capacity q. The output is a subset of nodes $V' \subseteq V$ such that the graph $G[V']$ induced by V' can simultaneously support unit of flow between vertices s_i and t_i, for each $i \in [k]$. The objective is to minimize the total cost $c(V') = \sum_{v \in V'} c_v$ of the output graph. We refer to the set $\{s_i\}_{i=1}^k \cup \{t_i\}_{i=1}^k$ of all sources and sinks as *terminals*. For any terminal s, we define its *mate* to be the unique t such that (s, t) is a request-pair.

3.1 Roadmap

Our algorithm first clusters the terminals into nearly node-disjoint trees $\mathcal{C}$ of low total cost like in the SSNC algorithm. Then we buy a subgraph H using the hallucination algorithm [6] to connect the clusters. Finally, we show that $\left(\bigcup_{T\in\mathcal{C}} T\right) \cup H$ is a subgraph of low cost which can route a constant fraction of all demands with low congestion. Since our overall algorithm is fairly involved, we first give a pseudocode and explain it informally.

In the clustering portion of the pseudocode below, we will maintain three types of clusters: (i) *frozen cluster*, which either has a size $\Omega(q)$, or more than half of the terminals it contains also have their mates inside the same cluster, (ii) *safe cluster*, which does not satisfy the criteria to be frozen, and also has most of its crossing demand to other active clusters, and (iii) *unsafe cluster*, which does not satisfy the criteria to be frozen, but has most of its crossing demand to frozen clusters. Safe clusters and unsafe clusters are together called active clusters. Our goal is to merge the active clusters with each other (or with frozen clusters) until they all become frozen.

Algorithm 2 Algorithm MCNC Pseudocode

repeat
  Clustering phase begins.
  define each remaining terminal to be its own safe cluster.
  repeat
    if there are unsafe clusters **then**
      solve a single-sink instance $\mathcal{I}_1$ with a unique source connected to each unsafe cluster and the sink connected to every frozen cluster.
      find lowest merge-points in this flow.
      if the lowest merge-point is not in a frozen cluster **then**
        merge the unsafe clusters meeting at this point.
      else
        merge the unsafe clusters to the frozen cluster and **delete** demands between the merged unsafe clusters and other active clusters.
      end if
    end if
    partition the safe clusters into $(\mathcal{T}^+, \mathcal{T}^-)$ such that every cluster in $\mathcal{T}^+$ has more demands to clusters in $\mathcal{T}^-$.
    solve a single-sink instance $\mathcal{I}_2$ with a unique source connected to each cluster in $\mathcal{T}^+$ and the sink connected to every cluster in $\mathcal{T}^-$.
    merge clusters using lowest merge-meeting points.
    if there are clusters where most demands are internal or that contain $\Omega(q)$ terminals **then**
      freeze these clusters.
    end if
  until all clusters are frozen
  Clustering phase ends, and routing phase begins.
  hallucinate a demand of q for each request-pair with probability $\Theta(\log n)/q$.
  solve min-cost subgraph H to route hallucinated demands by randomized rounding of a natural LP.
  if a constant fraction of the terminals in the frozen clusters are internal **then**
    route internal demands within each cluster.
  else
    delete at most half the request-pairs so that every component of the resulting demand graph has mincut $\geq q$.
    route all demands not deleted using the union of H and all the clusters.
  end if
  Routing phase ends.
until all demands are satisfied

Clustering. The algorithm works in $O(\log n)$ phases, in each phase we will merge a constant fraction of the clusters. Initially each remaining unsatisfied terminal is its own safe cluster, which is active by definition. Over time, clusters that reach size $\Omega(q)$ are said to be *frozen* and don't look to grow in size. In addition, we also freeze clusters when at least half of the terminals in the cluster have their mate also in same the cluster, since we can use the Steiner tree of this cluster to route its induced demands. In the single-sink setting, we merged these active clusters using the directed acyclic nature of an optimal unsplittable flow. However, we don't have such a witness here. We handle this by in fact repeatedly solving low-cost instances of the single sink problem! Given a set of clusters, we merge the active clusters using the following two instances of the single-sink SSNC problem.

The first SSNC instance $\mathcal{I}_1$ is defined as follows. we create a source for each unsafe cluster (with demand equal to the number of terminals it contains), and create a fake root vertex for each frozen cluster, and connect it to every terminal in the frozen cluster. The fake roots have 0 cost and capacity q. Finally, we create a global sink t and connect it to each fake root vertex. We first show that there *exists a low-cost solution to this instance* by deriving a witness solution using OPT for the original multicommodity instance, and then use our SSNC algorithm to find such a routing. Since this is an acyclic routing, we can merge these flows based on the deepest merge-points, and in turn merge the corresponding clusters. Some of these flows may merge at the fake root vertices, in which case we merge these clusters with the corresponding frozen cluster.

To define the the second instance $\mathcal{I}_2$, we partition the safe clusters into two groups $\mathcal{T}^+$ and $\mathcal{T}^-$ such that every cluster in $\mathcal{T}^+$ has more demand crossing over to clusters in $\mathcal{T}^-$ than to other clusters in $\mathcal{T}^+$. We create a source for each cluster in $\mathcal{T}^+$ (with demand equal to the number of terminals it contains), and create a global sink t and connect it to each terminal in $\mathcal{T}^-$. Like in the first part, we first show the existence of a low-cost solution, and find one such subgraph and associated routing. Again, we can merge these flows (and associated clusters) based on the deepest merge-points to get bigger clusters. We repeat this iterative merging until all clusters are frozen.

Hallucination. We connect these clusters using the hallucination algorithm: each request-pair hallucinates its demand to be q with probability $\log n/q$, and we find the minimum cost subgraph H to route the hallucinated demand concurrently (This problem is easy to solve by a simple LP rounding since all demands and capacities are equal). While this algorithm is the same as that in [6], our analysis differs significantly. Indeed, one main contribution here is in showing that the *union of H and the clusters is sufficient* to route a constant fraction of the demands. To this end, we partition the clusters into groups s.t the min-cut of the demands induced by any group is at least q; we then show that the hallucinated request-pairs behave like a *good cut-sparsifier* [28] after contracting the clusters, and hence conclude that H can route the original demands. We finally "un-contract" these clusters by using the Steiner trees to route within them, while ensuring that the load on these trees is bounded.

3.2 Clustering

Let X denote the set of all terminals from the k request pairs. We now cluster the terminals of X into (nearly) disjoint groups having useful properties for the subsequent routing step. We assume an α_{ss}-approximation β_{ss}-congestion algorithm for the single-sink node-capacitated network design problem. First, we define the types of clusters that we deal with, and then present our main clustering result.

Definition 14 *A cluster is a tree T containing a subset of terminals that are assigned to it. The number of terminals assigned to cluster T is denoted by $\mathsf{load}(T)$. A cluster is said to be one of the following three types:*

(i) **Internal Cluster** *is assigned $O(\beta_{\mathsf{ss}}^2 \log n) \cdot q$ terminals, and more than half its terminals have their mates also inside the cluster.*

(ii) **External Cluster** *is assigned $O(\beta_{\mathsf{ss}}^2 \log n) \cdot q$ terminals, and at least $q/8$ terminals have their mates outside this cluster.*

(iii) **Active Cluster** *is assigned at most $q/4$ terminals, and is neither internal nor external.*

Theorem 15 *There is an efficient algorithm to find a collection $\widehat{\mathcal{T}}$ of internal and external clusters such that:*

(i) The total cost of all the clusters $\sum_{T \in \widehat{\mathcal{T}}} \sum_{v \in T} c_v \leq \alpha_{\mathsf{mc}} \cdot c(\mathsf{Opt}) = O(\alpha_{\mathsf{ss}} \cdot \log n) c(\mathsf{Opt})$.

(ii) Each vertex appears in $O(\log n)$ different clusters.

(iii) At least a $1/4$ fraction of the request-pairs have both end-points in nodes of $\widehat{\mathcal{T}}$.

(iv) Each terminal is assigned to at most one cluster.

We now complete our algorithm assuming this theorem, and then prove it in Subsection 3.5.

3.3 Hallucinating to Connect Clusters

From Theorem 15, we have a collection $\widehat{\mathcal{T}}$ of (nearly) node-disjoint clusters, where each node lies in at most $O(\log n)$ clusters, and the total cost of all clusters is at most $\alpha_{\mathsf{mc}} \cdot c(\mathsf{Opt})$. We now perform a hallucination step to connect these clusters in a node-disjoint manner à la [6]: *each request-pair imagines its demand to be q units independently with probability $p = O(\log n)/q$, and zero otherwise*. Let $\mathcal{M}$ denote the set of hallucinated request-pairs. We now show that w.h.p, there exists a solution to $\mathcal{M}$ of low cost and bounded node congestion.

Lemma 16 *With high probability, there is a feasible routing of $\mathcal{M}$ using nodes of cost $c(\mathsf{Opt})$ where each node has congestion at most $O(\log n)$.*

Proof: Consider the optimal solution for the original MCNC instance. Let P_i^* denote the path used for sending unit flow between s_i and t_i for each pair $i \in [k]$. We now consider the solution X that sends q units of flow on the optimal path P_i^* for each hallucinated pair $i \in \mathcal{M}$. Clearly, the solution has cost at most $c(\mathsf{Opt})$. We will now show that this solution also has low congestion whp. To see this, consider any node $v \in V$. By feasibility of the solution $\{P_i^* : i \in [k]\}$, we have $|\{i \in [k] : v \in P_i^*\}| \leq q$. The load on node v in solution X is $L_v := \sum_{i \in [k]: v \in P_i^*} q \cdot I_i$ where I_i is a $0/1$ random variable denoting whether/not pair i hallucinates. Note that L_v is the sum of independent $[0, q]$ random variables, with mean $\mathbb{E}[L_v] \leq q$. By a standard Chernoff bound, there is a constant d_1 such that $\Pr[L_v > d_1 \cdot \log n \cdot q] \leq \frac{1}{n^3}$. Taking a union bound over all v, we have $\Pr[\exists v \text{ s.t } L_v > d_1 \cdot \log n \cdot q] \leq \frac{1}{n^2}$. So X satisfies the condition in the lemma whp. □

For such instances, notice that all demands and capacities are q. We show a simple LP-based bi-criteria approximation algorithm when demands and capacities are equal. Indeed, let $\{(s_i, t_i) : i \in \mathcal{M}\}$ denote the hallucinated pairs (with demand q each). We use the following natural LP relaxation:

$$\min \sum_v c_v x_v \qquad (LP_h)$$

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

$$\sum_{p | v \in p} f(p) \leq q \cdot x_v \qquad \forall v \in V \qquad (2)$$

$$f(p) \geq 0 \qquad \forall i \in \mathcal{M}, \forall p \in \mathcal{P}_i \qquad (3)$$

$$0 \leq x_v \leq O(\log n) \qquad \forall v \in V \qquad (4)$$

Here $\mathcal{P}_i$ is the set of all s_i-t_i flow paths in G. The x-variables represent which nodes are used. Note that we allow these variables to take values up to $O(\log n)$: this capacity relaxation ensures that the cost of the hallucinated instance is small (we can use Lemma 16 to bound the cost of this LP). After solving LP_h, we do a simple randomized rounding on the flow paths of each hallucinated pair $i \in \mathcal{M}$: this yields an unsplittable routing U_i of q units between s_i and t_i. Let $H = \{U_i\}_{i \in \mathcal{M}}$ denote the resulting solution.

Lemma 17 *The hallucinated flow graph H consists of a path U_i for each $i \in \mathcal{M}$ such that:*

- *The expected cost of nodes in $H \equiv \cup_{i \in \mathcal{M}} U_i$ is $O(\log n) \cdot c(\mathsf{Opt})$.*
- *With high probability, $|\{i \in \mathcal{M} : U_i \ni v\}| \leq O(\log n)$ for all $v \in V$.*

Proof: For every vertex v in $V(\mathsf{Opt})$, set $x_v = \Theta(\log n)$, and set $x_v = 0$ for every other vertex. From Lemma 16, we know that w.h.p, this is a feasible solution for LP_h with cost $O(\log n) \cdot c(\mathsf{Opt})$. Since we perform randomized rounding on the optimal LP_h solution, to get our paths U_i and subgraph H, the expected cost of H equals the LP cost which is $O(\log n) \cdot c(\mathsf{Opt})$. Moreover, the expected number of U_i paths through any node is at most $O(\log n)$. Using standard Chernoff bounds, we get that w.h.p., the number of paths U_i's through any node is $O(\log n)$. □

3.4 Our Final Solution, and Finding Routable Request-Pairs

Our final solution for this iteration is $\widehat{F} = \left(\bigcup_{T \in \widehat{\mathcal{T}}} T\right) \cup H$. We now constructively compute the set of request-pairs whose demands can be routed with low congestion, and then recurse on the remaining demands.

Theorem 18 *The cost of $\widehat{F}$ is $O(\alpha_{\mathsf{ss}} \log n) \cdot c(\mathsf{Opt})$, and we can efficiently find a set of at least $\Omega(k)$ request-pairs which can be routed in $\widehat{F}$ with congestion at most $O(\beta_{\mathsf{ss}}^2 \log^3 n)$.*

Proof: Let us partition $\widehat{\mathcal{T}} = \mathcal{T}_{ex} \cup \mathcal{T}_{in}$ as the disjoint union of the external clusters and internal clusters. Our constructive proof will use the notion of a cluster graph, defined below:

Definition 19 (Cluster Graph $G_C(\mathcal{T})$) *Given a collection $\mathcal{T}$ of clusters, the (multi)graph $G_C(\mathcal{T})$ consists of a vertex for every cluster $T \in \mathcal{T}$, and an edge between clusters $T_a, T_b \in \mathcal{T}$ for each request-pair (s,t) with $s \in T_a$ and $t \in T_b$.*

Our high-level idea is to consider two cases: if most of the demands are contained in $\mathcal{T}_{in}$, then we simply use the node-disjoint trees we bought in the clusters to route all the demands without congesting each node too much. On the other hand, if most of the demands are contained in $\mathcal{T}_{ex}$, we would like hallucination to come to our help. Here we use the expansion property of the clusters in $\mathcal{T}_{ex}$ to argue that we can partition the clusters in $\mathcal{T}_{ex}$ into $\mathcal{T}_1, \mathcal{T}_2, \ldots$ such that the min-cut of each $G_C(\mathcal{T}_j)$ is at least $\Omega(q)$, and we throw out at most a constant fraction of the request-pairs. We next create a graph $H_C(j)$ where we place an edge of capacity q between the clusters of the end-points of each hallucinated request-pair. But when the min-cuts are all large, *hallucination does exactly what Karger's cut sparsification algorithm does — sample each request-pair with probability $\Theta(\log n/q)$ and set its value to be q!* Therefore, each original request-pair in $G_C(\mathcal{T}_j)$ can be routed (in an edge-disjoint manner) in the graph $H_C(j)$. We know that for each edge in $H_C(j)$, we have a corresponding path of capacity q in the graph H (Lemma 17), so we can route the flow using this path. Finally, we need to route *within the clusters*, to jump from one edge to another in the cluster graph. We use the Steiner trees within each of the (nearly) node-disjoint clusters for this purpose. We now formally present these ideas. Indeed, by property (iii) of Theorem 15, we know that at least $k/4$ request-pairs have their terminals in $\bigcup_{T \in \mathcal{T}_{ex} \cup \mathcal{T}_{in}} T$. We now consider two cases:

More request-pairs incident in $\mathcal{T}_{in}$ We first handle the easy case where at least $k/8$ request-pairs have at least one of their terminals incident on $\bigcup_{T \in \mathcal{T}_{in}} T$. In this case, because each cluster T in $\mathcal{T}_{in}$ has at least half as many terminals paired up internally as its total number of terminals, we get that there are at least $k/32$ request-pairs (s,t) such that both s and t are contained in the same cluster of $\mathcal{T}_{in}$. So, all these demands can be concurrently routed using just the nodes of $\cup_{T \in \mathcal{T}_{in}} T$. Therefore, we would route at least $1/32$ fraction of all demands at cost $O(\alpha_{\mathsf{ss}} \log n) \cdot c(\mathsf{Opt})$ and congestion $O(\beta_{\mathsf{ss}}^2 \log^2 n)$.

More request-pairs incident in $\mathcal{T}_{ex}$ We may now assume that at least $k/8$ request-pairs have *both* their terminals in $\bigcup_{T \in \mathcal{T}_{ex}} T$. So, we are in the case where the cluster graph $G_C(\mathcal{T}_{ex})$ has at least $k/8$ edges. We now partition $\mathcal{T}_{ex}$ into $\mathcal{T}_1, \mathcal{T}_2, \ldots$ such that the min-cut of each $G_C(\mathcal{T}_j)$ is $\Omega(q)$. For the following, we shall refer to $G_C(\mathcal{T}_{ex})$ as G_C. Note that each vertex in G_C has degree between $q/8$ and $O(\beta_{\mathsf{ss}}^2 \log n)q$. For the remaining part of the proof, let δ denote the constant $1/8$ and Δ denote $O(\beta_{\mathsf{ss}}^2 \log n)$. Let N be the number of vertices in G_C; so the number of edges in G_C is between $Nq\delta/2$ and $Nq\Delta$. We first show that a small number of edges can be removed from G_C so that each connected component has large min-cut (at least $\delta q/4$).

Lemma 20 *There is a poly-time computable subgraph G'_C of G_C containing at least $Nq\delta/4$ edges such that every connected component of G'_C has min-cut at least $\frac{\delta q}{8}$.*

Proof: Consider the following procedure to construct subgraph G'_C. Initially $K = G_C$ and $G'_C = \emptyset$. As long as $\mathsf{min\text{-}cut}(K) < (\delta q)/4$ do:

1. Let $S \subseteq V(K)$ be a minimal min-cut in K.
2. Remove the edges in K crossing S.
3. Add $K[S]$ to G'_C if S is not a singleton vertex.

Add the final graph K to G'_C, if K is not a singleton vertex.

Clearly, the number of iterations above is at most N, the number of vertices in G_C. Since at most $(\delta q)/4$ edges are removed in each iteration, the total number of edges removed is at most $(\delta N q)/4$. So G'_C has at least $\delta N q/2 - \delta N q/4 = \delta N q/4$ edges. We now show that the min-cut of each component in G'_C is at least $\frac{\delta q}{8}$. Note that each component of G'_C is either (i) the set S in some iteration above, or (ii) the final graph K. Clearly, in the latter case, the component has min-cut at least $(\delta q)/4 \geq \frac{\delta q}{8}$. In the former case, consider the graph K and set S in the iteration when this component was created. Let $A \subseteq S$ be any strict subset. Note that $|\partial_K(A)| \geq \frac{\delta q}{4}$ and $|\partial_K(S \setminus A)| \geq \frac{\delta q}{4}$ by the minimality of set S.[4] Let a (resp. b) denote the number of edges having one end-point in A (resp. $S \setminus A$) and the other end-point in $V(K) \setminus S$. Also let x denote the number of edges having one end-point in A and the other in $S \setminus A$. Note that:

$$|\partial_K(A)| = x + a, \quad |\partial_K(S \setminus A)| = x + b, \quad |\partial_K(S)| = a + b.$$

Combined with the observation above (by minimality of cut S),

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

It follows that $x \geq \frac{\delta q}{8}$, i.e. the number of edges between A and $S \setminus A$ is at least $\frac{\delta q}{8}$. Since this holds for all strict subsets $A \subseteq S$, the min-cut of $K[S]$ is at least $\frac{\delta q}{8}$. □

Let the clusters in each connected component in G'_C be denoted as $\mathcal{T}_1, \mathcal{T}_2, \ldots$. Here $\mathcal{T}_j$ is the set of clusters corresponding to the j^{th} connected component in G'_C.

Routing flow in hallucinated graph. For each component $\mathcal{T}_j$ of G'_C, we define a "hallucinated graph" $H_C(j)$ as follows. Vertices of $H_C(j)$ correspond to clusters of $\mathcal{T}_j$. For each hallucinated pair $(s_i, t_i) \in \mathcal{M} \cap G_C(\mathcal{T}_j)$ that is induced on $\mathcal{T}_j$, there is *an edge of capacity* q in $H_C(j)$ between the clusters containing s_i and t_i; this edge corresponds to the path U_i between s_i and t_i in the hallucination step (Lemma 17). Henceforth, we shall slightly abuse notation and refer to $G_C(\mathcal{T}_j)$ as $G_C(j)$.

Lemma 21 *All request-pairs in $G_C(j)$ can be routed in $H_C(j)$ without exceeding edge capacities, w.h.p.*

Proof: Note that, by Lemma 20, the minimum cut in $G_C(j)$ is at least $\frac{\delta q}{8}$. Moreover, because each request-pair hallucinated independently, each edge of $G_C(j)$ lies in $H_C(j)$ independently with probability $p = \Theta(\log n)/q$. We now use the following cut-sparsification theorem due to Karger.

Theorem 22 ([28]) *Let G be an undirected graph with min-cut c, and $p \in [0, 1]$. Let H be a graph containing each edge of G independently with probability p. If $p \cdot c \geq \frac{3(d+2)\ln n}{\epsilon^2}$ for some d, ϵ, then with probability $1 - n^{-d}$, every cut in H has value within $p(1 \pm \epsilon)$ the cut value in G.*

Setting a large enough constant in our sampling probability $p = \Theta(\log n)/q$, and using $\textsf{min-cut}(G_C(j)) \geq \frac{\delta q}{8}$, we obtain that w.h.p. $\partial_{H_C(j)}(T) \geq \frac{1}{2} \cdot p \cdot \partial_{G_C(j)}(T)$ for all $T \subseteq V(G_C(j))$. Since each edge of $H_C(j)$ actually has capacity q, the capacity across any cut $T \subseteq V(G_C(j))$ is at least $\frac{q}{2} \cdot p \cdot \delta_{G_C(j)}(T)$. In other words, the *non-uniform sparsest cut* of this multicommodity routing instance is:

$$\min_{T \subseteq V(G_C(j))} \frac{\text{capacity across } T}{\text{demand across } T} \quad = \quad \min_{T \subseteq V(G_C(j))} \frac{q \cdot \delta_{H_C(j)}(T)}{\delta_{G_C(j)}(T)} \quad \geq \quad \frac{pq}{2}.$$

Since we chose $p = \Theta(\log n)/q$, we can ensure that the sparsest cut is at least the multicommodity flow-cut gap $\lambda = \Theta(\log n)$ [32]. This proves the existence of a routing for request-pairs in $G_C(j)$. □

Now we find such an edge-capacitated multicommodity routing for every request-pair in $G_C(j)$ using the edges of $H_C(j)$. For each request-pair $(s_i, t_i) \in G_C(j)$ let f_i denote its unit-flow in graph $H_C(j)$; note that for each edge of $H_C(j)$, the total flow through it is at most q. Moreover, the total flow *through any vertex* is also bounded since:

[4] For any undirected graph J and vertex subset X, the set of edges with exactly one end-point in X is denote $\partial_J(X)$.

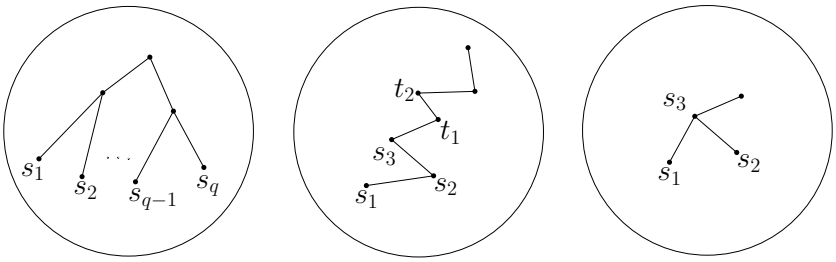


Figure 1: Examples of external, internal clusters, and active clusters

Claim 23 *With high probability, the total capacity of edges in $H_C(j)$ incident to any vertex is $O(\log n) \cdot q$.*

Proof: Consider any vertex (cluster) $T \in G_C(j)$. Recall that the number of edges incident to T (i.e. the terminals in T) is $O(\beta_{\mathsf{ss}}^2 \log n)q$. Since $H_C(j)$ contains each edge independently with probability $p = \Theta(\log n)/q$, the expected number of edges incident to T is $O(\beta_{\mathsf{ss}}^2 \log^2 n)$. By a simple Chernoff bound, the number of edges incident to T is also $O(\beta_{\mathsf{ss}}^2 \log^2 n)$ with high probability. A simple union bound completes the proof. □

Converting edge-capacitated routing in $H_C(j)$ to real routing in G. Now we show that this edge-capacitated routing $\mathcal{F} = \{f_i : i \in G_C(j)\}$ in the hallucinated graph can also be implemented as a node-capacitated routing in the real graph G. This combines the hallucinated routing H (from Lemma 17) with nodes of the clusters $\mathcal{T}_j$. Indeed, recall that each edge e in $H_C(j)$ corresponds to a flow-path U_e in H carrying q units between source s (say in cluster T_a) and sink t (say in cluster T_b): then we simply route all the flow through e in $\mathcal{F}$, along the path U_e in H. Note that the total flow through any edge $e \in E(H_C(j))$ (and hence through path U_e) is at most q. In order to route flow in $\mathcal{F}$ through any vertex (say cluster T), we use the Steiner tree for cluster T obtained from Theorem 15. By Claim 23, the total flow in $\mathcal{F}$ through any vertex $T \in G_C(j)$ (and hence through any vertex in T's Steiner tree) is $O(\log^2 n)\beta_{\mathsf{ss}}^2 \cdot q$. Finally, observe that each node of the underlying graph lies in (i) $O(\log n)$ clusters of $\widehat{\mathcal{T}}$, and (ii) $O(\log n)$ hallucinated paths $\{U_i : i \in \mathcal{M}\}$ in H. Thus the total flow through any node of the original graph is at most $O(\beta_{\mathsf{ss}}^2 \log^3 n)q$. This completes the proof of Theorem 18. □

3.5 Proof of Theorem 15

We now complete our discussion by proving the clustering theorem. Recall the different types of clusters from Definition 14 (Figure 1). Also, a terminal assigned to cluster T is said to be *crossing* (resp. *internal*) if its mate lies outside (resp. inside) T. The number of terminals in cluster T is denoted as $\mathsf{load}(T)$.

Our algorithm will proceed in many iterations. Initially each terminal is assigned to its own cluster, and is trivially an active cluster. At any stage of our algorithm, we refer to the current collection of all external and internal clusters as the **frozen clusters**; these clusters don't look to expand further. In each iteration, we will merge some active clusters (either within themselves or to existing frozen clusters) until all clusters are frozen. We will ensure that in every iteration, the number of active clusters decreases by a constant factor. This ensures that the number of iterations is $O(\log n)$. Along the way, we will also have to discard some request-pairs, so we also need to bound the fraction of discarded pairs (to establish property (iii)) of the theorem. In each iteration, we will use two new copies of the graph, and add a node-disjoint collection of trees in each copy. Since there are only $O(\log n)$ iterations, we can easily bound the total number of times a vertex appears by $O(\log n)$.

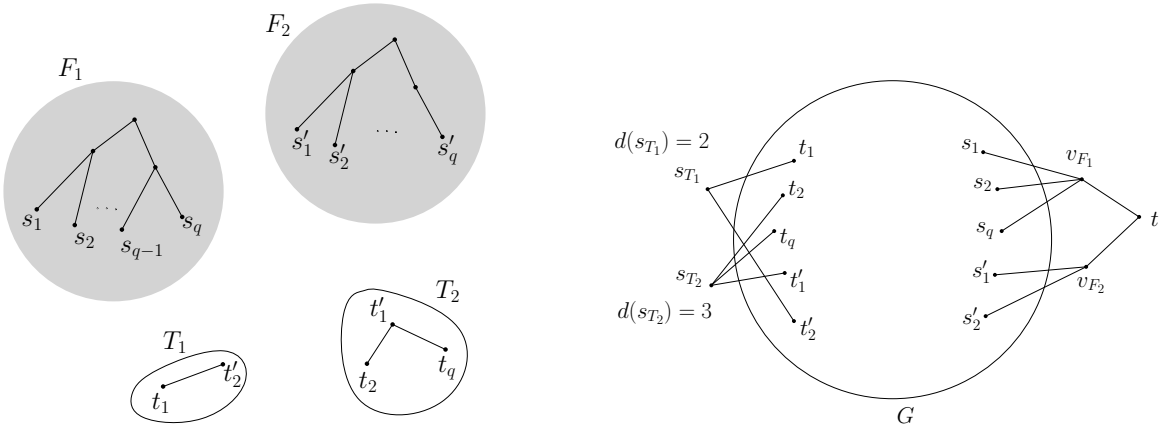
We now describe a single iteration of the clustering algorithm. The current clusters at the start of any iteration consist of frozen and active clusters. We will maintain the following important invariant in all iterations:

Invariant 1 *Each frozen cluster has at most $9\beta_{\mathsf{ss}} \cdot q$ demands crossing to active clusters.*

This is trivially true at the start of the clustering, since there are no frozen clusters.

Defining unsafe active clusters. Repeat the following as long as there is some active cluster T with more than a quarter of its terminals having mates in frozen clusters:

1. Delete all demands crossing from T to other active clusters.



(a) Unsafe clusters and frozen clusters

(b) Corresponding single-sink instance $\mathcal{I}_1$

Figure 2: Illustration of SSNC Instance $\mathcal{I}_1$

2. If T or any other active cluster now becomes *internal*, add them to the frozen set.
3. If T is not an internal cluster, declare T to be an *unsafe cluster*.

All remaining active clusters are called *safe clusters*: these have less than a quarter of their terminals crossing over to frozen clusters.

Claim 24 *The number of deleted demands is at most thrice the number of demands between frozen clusters and newly formed unsafe clusters.*

Proof: Consider the demands deleted when an active cluster T is considered above. By the condition on T, it has at least $\mathsf{load}(T)/4$ demands crossing to frozen clusters. Thus, the number of remaining terminals in T is at most $\frac{3}{4} \cdot \mathsf{load}(T)$, which is also an upper bound on the number of demands deleted from T. Adding over all clusters T considered above, the claim follows. □

Let $\mathcal{T}_f$, $\mathcal{T}_u$ and $\mathcal{T}_s$ denote the collections of frozen, unsafe and safe clusters at this point. We will merge the unsafe and safe clusters separately, using two different instances of the single-sink problem (SSNC).

3.5.1 Single-sink instance $\mathcal{I}_1$ for unsafe clusters.

Notice that each unsafe cluster only has internal request-pairs and those crossing over to the frozen clusters. We merge all unsafe clusters (either with one another, or with some frozen cluster) by solving the following SSNC instance $\mathcal{I}_1$. The node capacities in this instance are $\tilde{q} = 5q$.

- For each unsafe cluster $T \in \mathcal{T}_u$, there is a new source node s_T of zero cost, with unsplittable demand $d(s_T) = \mathsf{load}(T)$, connected to each terminal of T.
- For each frozen cluster $F \in \mathcal{T}_f$, there is a new "root" vertex v_F of zero cost, connected to every terminal of F. The capacity of this node is $8\beta_{\mathsf{ss}} \cdot \tilde{q}$; equivalently, there are $8\beta_{\mathsf{ss}}$ identical copies of v_F having capacity $\tilde{q}$ each.
- There is a new global sink t that is connected to all the roots $\{v_F : F \in \mathcal{T}_f\}$.
- Every vertex $v \in V$ (in the original graph) has cost $c(v)$, and node capacity $\tilde{q}$.

Note that each demand is at most $q/4$ since all clusters in $\mathcal{T}_u$ are active. So we can use the SSNC algorithm from Section 2. A simple example is shown in Figure 2. Below we show that the optimal cost of this instance is small.

Claim 25 *For every unsafe cluster $T \in \mathcal{T}_u$, the number of crossing terminals is at least* $\mathsf{load}(T)/4$.

Proof: This is immediate, by definition of unsafe clusters. □

Lemma 26 *The optimal cost of SSNC instance $\mathcal{I}_1$ is at most $c(\mathsf{Opt})$.*

Proof: We first exhibit a feasible fractional flow solution using the optimal multicommodity flow of the MCNC instance. Let $V^* \subseteq V$ be the nodes used in Opt. For each demand pair $i \in [k]$, 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 $|\{i \in [k] : P_i^* \ni v\}| \leq q$ for all nodes v.

Consider any unsafe cluster T. Note that all crossing terminals of T have mates in frozen clusters. We route its $\mathsf{load}(T)$ demand from s_T to t as follows: for each crossing terminal s_i in T (with its mate t_i in $F \in \mathcal{T}_f$), send 4 units from s_T to s_i, then along path P_i^*, then from t_i to the root v_F, and finally from v_F to sink t. Note that these are valid paths in the graph of instance $\mathcal{I}_1$. Moreover, by Claim 25, the net flow out of s_T is at least $\mathsf{load}(T)$.

Routing as above for all $T \in \mathcal{T}_u$, we get a fractional routing to $\mathcal{I}_1$ using nodes V^*, where the flow through each node $v \in V^*$ is at most $4 \cdot |\{i \in [k] : P_i^* \ni v\}| \leq 4q$. Moreover, the flow through each root v_F is at most 4 times the number of crossing terminals in F, i.e. at most $36\beta_{\mathsf{ss}}q$ using Invariant 1. We can now apply the DGG algorithm to this fractional routing and obtain *an unsplittable flow* for $\mathcal{I}_1$, where (i) the flow through each node of V^* is at most $4q + \max_{T \in \mathcal{T}_u} \mathsf{load}(T) \leq 5q = \tilde{q}$, and (ii) the flow through each root v_F is at most $36\beta_{\mathsf{ss}}q + q \leq 8\beta_{\mathsf{ss}}\tilde{q}$. □

Following Lemma 26, and applying our single-sink algorithm (Theorem 3) on $\mathcal{I}_1$, we obtain an unsplittable flow $\mathcal{F}_1$ from the sources $\{s_T : T \in \mathcal{T}_u\}$ to the global sink t. Moreover, we know that (a) the cost of nodes used in $\mathcal{F}_1$ is at most $\alpha_{\mathsf{ss}} \cdot c(\mathsf{Opt})$, (b) the capacity used at any node of V is at most $\beta_{\mathsf{ss}} \cdot \tilde{q} = 5\beta_{\mathsf{ss}}q$, and (c) the capacity used at any root v_F is at most $\beta_{\mathsf{ss}} \cdot 8\beta_{\mathsf{ss}}\tilde{q} = 40\beta_{\mathsf{ss}}^2 q$. We may assume, wlog, that $\mathcal{F}_1$ is acyclic since it corresponds to a single-sink flow. Next, we run (one call of) the Cluster algorithm from Subsection 2.2 with $X = \{s_T : T \in \mathcal{T}_u\}$, demands $d(s_T) = \mathsf{load}(T)$ for all $T \in \mathcal{T}_u$, and $\mathcal{F} = \mathcal{F}_1$. Recall that a single call of Cluster merges nodes of X using node-disjoint trees $\{\tau_v\}$ (from the support of $\mathcal{F}_1$) such that (a) each tree (except possibly one) contains at least two nodes of X, and (b) the total demand in each tree τ_v is at most the capacity at its center vertex v. We merge the unsafe clusters $\mathcal{T}_u$ using this collection $\{\tau_v\}$. There are two cases:

1. The center $v \notin \{v_F : F \in \mathcal{T}_f\}$. In this case, we merge the unsafe clusters $\{T \in \mathcal{T}_u : s_T \in \tau_v\}$ using tree τ_v. Note that the total demand in this new cluster is at most $5\beta_{\mathsf{ss}}q$.
2. The center $v \in \{v_F : F \in \mathcal{T}_f\}$. In this case, we will merge the clusters $\mathcal{T}_v = \{T \in \mathcal{T}_u : s_T \in \tau_v\}$ with the frozen cluster F. Observe that the path from each s_T ($T \in \mathcal{T}_v$) to root v_F must go through some terminal in cluster F (recall that these are the only vertices connected to v_F). So we simply merge the clusters $\mathcal{T}_v$ with the frozen cluster F by adding the nodes on their path to F in τ_v. Note that the increase in demand at the frozen cluster F is at most $40\beta_{\mathsf{ss}}^2 q$.

If any newly formed cluster (after merging) becomes an internal or external cluster then we freeze it. Based on the above discussion, we have:

Claim 27 *The following properties hold after the of unsafe clusters:*

(i) The size/load of any newly frozen cluster is at most $5\beta_{\mathsf{ss}}q$.

(ii) The load of each existing frozen cluster increases by at most $40\beta_{\mathsf{ss}}^2 q$.

(iii) The number of new active clusters is at most $\frac{1}{2}|\mathcal{T}_u|$.

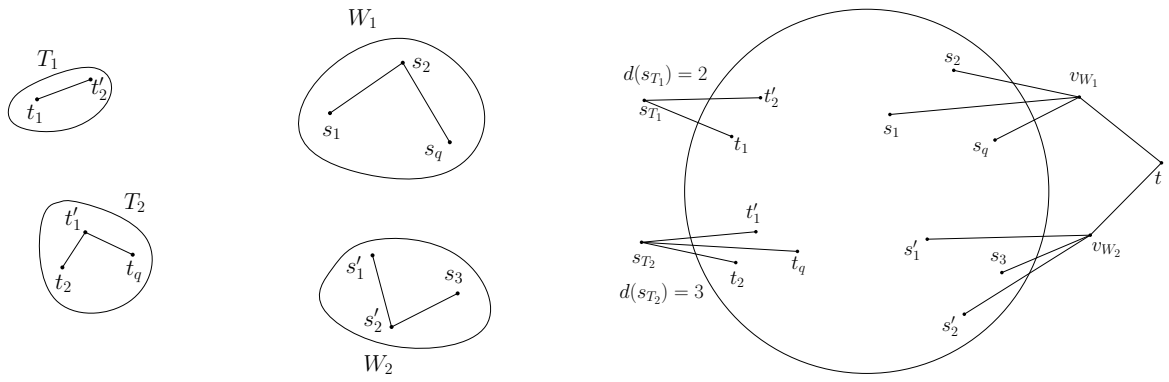
(iv) The new vertices added to clusters form disjoint sets. The total cost of new vertices is at most $\alpha_{\mathsf{ss}} \cdot c(\mathsf{Opt})$.

(v) For any frozen cluster T', the number of demands crossing from T' to active clusters does not increase.

(vi) Invariant 1 continues to hold.

Proof: Conditions (i)-(iv) clearly hold, by the properties of instance $\mathcal{I}_1$ and our merging step using Cluster. For condition (v), notice that only unsafe clusters get merged with any frozen cluster T'; since we deleted all demands from unsafe clusters to other active clusters, there is no increase in the number of demands between T' and active clusters. By conditions (i) and (v), Invariant 1 also holds. □

Remark: Notice how Invariant 1 is useful in bounding the load increase of existing frozen clusters, over the iterations. Ensuring this property is the reason we deleted demands out of unsafe clusters.



(a) Safe clusters and the bipartition $(\mathcal{T}_s^+, \mathcal{T}_s^-)$ (b) Corresponding single-sink instance $\mathcal{I}_2$

Figure 3: Illustration of SSNC Instance $\mathcal{I}_2$

3.5.2 Single-sink instance $\mathcal{I}_2$ for safe clusters.

We now consider the safe active clusters and merge them amongst themselves, again by solving an appropriate single-sink instance. Observe that:

Claim 28 *Each safe cluster $S \in \mathcal{T}_s$ has $\mathsf{load}(S) \leq q/4$ terminals, and has at least $\mathsf{load}(S)/4$ terminals cross to other safe clusters.*

Proof: Since safe clusters are active, each $S \in \mathcal{T}_s$ has $\mathsf{load}(S) \leq q/4$ terminals, and at least $\mathsf{load}(S)/2$ are crossing. By construction of the safe clusters, at most $\mathsf{load}(S)/4$ terminals of S cross to frozen clusters. Moreover, there are no demands between safe and unsafe clusters (recall that each unsafe cluster deletes all its demands crossing to active clusters). Thus S has at least $\mathsf{load}(S)/4$ terminals crossing to other safe clusters. □

We first compute a bipartition $(\mathcal{T}_s^+, \mathcal{T}_s^-)$ of the safe clusters, where each cluster $S \in \mathcal{T}_s^+$ (resp. $S \in \mathcal{T}_s^-$) has more demand crossing from S to $\mathcal{T}_s^-$ (resp. $\mathcal{T}_s^+$) than from S to $\mathcal{T}_s^+$ (resp. $\mathcal{T}_s^-$). This can be computed by a simple iterative max-cut procedure: if there is any cluster having more demands within its part, then it is moved to the other part. The total number of demands crossing the bipartition increases by at least one in each step above: so the procedure terminates. Wlog, we assume that $|\mathcal{T}_s^+| \geq |\mathcal{T}_s^-|$. We refer to clusters in $\mathcal{T}_s^+$ and $\mathcal{T}_s^-$ as *source* and *sink* clusters respectively. So there are at least $|\mathcal{T}_s|/2$ source clusters. By Claim 28 and the construction of this bipartition, it follows that:

Claim 29 *Each source cluster $S \in \mathcal{T}_s^+$ has at least $\mathsf{load}(S)/8$ demands crossing to sink clusters.*

We are now ready to define the SSNC instance $\mathcal{I}_2$. A small illustration appears in Figure 3. The node capacities in this instance are $q' = 9q$.

- For each source cluster $T \in \mathcal{T}_s^+$, there is a new source node s_T of zero cost, with unsplittable demand $d(s_T) = \mathsf{load}(T)$, connected to each terminal of T.
- For each sink cluster $W \in \mathcal{T}_s^-$, there is a new "root" vertex v_W of zero cost and capacity q', connected to every terminal of W.
- There is a new global sink t that is connected to all the roots $\{v_W : W \in \mathcal{T}_s^-\}$.
- Every vertex $v \in V$ (in the original graph) has cost c_v, and node capacity q'.

Lemma 30 *The optimal cost of SSNC instance $\mathcal{I}_2$ is at most $c\mathsf{Opt}$.*

Proof: The proof is similar to Lemma 26 for SSNC instance $\mathcal{I}_1$. We first exhibit a feasible fractional flow solution using the optimal multicommodity flow of the MCNC instance. Let $V^* \subseteq V$ be the nodes used in Opt. For each demand pair $i \in [k]$, 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 $|\{i \in [k] : P_i^* \ni v\}| \leq q$ for all nodes v.

Consider any source cluster $T \in \mathcal{T}_s^+$. By Claim 29 it has at least $\mathsf{load}(T)/8$ demands crossing to sink clusters: let $C_T \subseteq [k]$ denote this set of demands. We route s_T's demand of $\mathsf{load}(T)$ units from s_T to t as follows: for each $i \in C_T$, send 8 units from s_T to s_i, then along path P_i^*, then from t_i to the root v_W (where t_i lies in sink cluster $W \in \mathcal{T}_s^-$), and finally from v_W to sink t. Note that these are valid paths in the graph of instance $\mathcal{I}_2$. Moreover, the net flow out of s_T is at least $\mathsf{load}(T)$.

Performing the above routing for all $T \in \mathcal{T}_s^+$, we obtain a fractional-routing solution to $\mathcal{I}_2$ using nodes V^*, where the flow through each node $v \in V^*$ is at most $8 \cdot |\{i \in [k] : P_i^* \ni v\}| \leq 8q$. Moreover, the flow through each root v_W is at most 8 times the number of crossing terminals in $W \in \mathcal{T}_s^-$, i.e. at most $8 \cdot \frac{q}{4} = 2q$.

Now, applying the DGG algorithm to this fractional routing, we obtain an unsplittable flow for $\mathcal{I}_2$, where the flow through each node is at most $8q + \max_{T \in \mathcal{T}_s^+} \mathsf{load}(T) \leq 9q = q'$. This completes the proof of the lemma. □

Again, applying our single-sink algorithm (Theorem 3) on $\mathcal{I}_2$, we obtain an unsplittable flow $\mathcal{F}_2$ from the sources $\{s_T : T \in \mathcal{T}_s^+\}$ to the global sink t. Moreover, by Lemma 26 we know that (a) the cost of nodes used in $\mathcal{F}_2$ is at most $\alpha_{\mathsf{ss}} \cdot c(\mathsf{Opt})$, and (b) the capacity used at any node is at most $\beta_{\mathsf{ss}} \cdot q' = 9\beta_{\mathsf{ss}} q$. We assume, wlog, that $\mathcal{F}_2$ is acyclic since it corresponds to a single-sink flow. Next, (as for $\mathcal{I}_1$) we run the Cluster algorithm (Subsection 2.2) with $X = \{s_T : T \in \mathcal{T}_s^+\}$, demands $d(s_T) = \mathsf{load}(T)$ for all $T \in \mathcal{T}_s^+$, and $\mathcal{F} = \mathcal{F}_2$. This merges nodes of X using node-disjoint trees $\{\tau_v\}$ (from the support of $\mathcal{F}_2$) such that (a) each tree (except possibly one) contains at least two nodes of X, and (b) the total demand in each tree τ_v is at most the capacity $\beta_{\mathsf{ss}} q'$ at its center vertex v. We merge the source clusters $\mathcal{T}_s^+$ using this collection $\{\tau_v\}$. There are two cases: a subset of source clusters merge with each other, or a subset of source clusters merge with a sink cluster. In either case, the total demand in the new cluster is at most $\beta_{\mathsf{ss}} q' = 9\beta_{\mathsf{ss}} q$.

Again, if any newly formed cluster (after merging) becomes an internal or external cluster then we freeze it. Based on the above discussion, we have

Claim 31 *The following properties hold after the merging of safe clusters:*

(i) The size of any newly frozen cluster is at most $9\beta_{\mathsf{ss}} q$.

(ii) The number of new active clusters is at most $\frac{3}{4}|\mathcal{T}_s|$.

(iii) The new vertices added to clusters form disjoint sets. The total cost of new vertices is at most $\alpha_{\mathsf{ss}} \cdot c(\mathsf{Opt})$.

(iv) Invariant 1 continues to hold.

3.5.3 Assembling the pieces: Completing Proof of Theorem 15

Summarizing the two steps in subsections 3.5.1 and 3.5.2 we have (by combining Claims 27 and 31):

Lemma 32 *In each iteration, the following properties hold:*

1. *Each newly frozen cluster has load at most $9\beta_{\mathsf{ss}} q$.*
2. *The load increase of any existing frozen cluster is at most $40\beta_{\mathsf{ss}}^2 q$.*
3. *The number of demands crossing from any frozen cluster to active clusters is at most $9\beta_{\mathsf{ss}} q$.*
4. *The total cost of new vertices added to clusters is at most $2\alpha_{\mathsf{ss}} c(\mathsf{Opt})$.*
5. *The number of clusters containing any vertex increases by at most two.*
6. *The number of active clusters at the end of the iteration is at most $\frac{3}{4}|\mathcal{T}_s| + \frac{1}{2}|\mathcal{T}_u|$.*
7. *The number of deleted demands is at most thrice the number of demands between frozen clusters and unsafe clusters formed in this iteration.*
8. *No demand with an end-point in a frozen cluster is deleted.*

Finally, to prove Theorem 15, we now show each of the stated properties. Note that by property 6 above, the number of active clusters decreases by a factor $4/3$ in each iteration. So there are only $O(\log n)$ iterations.

To bound the load of any final cluster, note that the load of a frozen cluster when it is formed is at most $9\beta_{\mathsf{ss}} q$ (property 1), and it may increase by at most $40\beta_{\mathsf{ss}}^2 q$ in each subsequent iteration (property 2). So the final load of any cluster is $O(\beta_{\mathsf{ss}}^2 \log n) \cdot q$, which satisfies the definition of internal/external clusters.

Since there are only $O(\log n)$ iterations, property 4 implies that the total cost of nodes is $O(\alpha_{\mathsf{ss}} \log n) \cdot c(\mathsf{Opt})$. Similarly, using property 5, the total number of clusters containing any vertex is $O(\log n)$.

We now bound the number of deleted demands. By property 7 the number deleted in each iteration is at most thrice the number of demands having one end-point in a frozen cluster and the other in a *new* unsafe cluster (formed in the current iteration). Moreover, by property 8, demands incident to a frozen cluster are never deleted. So the total number of deleted demands is at most thrice the number of demands incident to frozen clusters. So there are at least $k/4$ demands incident to frozen clusters, at the end of the clustering. Since no demand from frozen clusters is deleted (property 8), it follows that at least $k/4$ demands are contained in the final set of clusters.

References

[1] Vision and roadmap: Routing telecom and data centers toward efficient energy use, May 2009. Proceedings of Vision and Roadmap Workshop on Routing Telecom and Data Centers Toward Efficient Energy Use.

[2] Matthew Andrews. Hardness of buy-at-bulk network design. In *FOCS*, pages 115–124, 2004.

[3] Matthew Andrews, Spyridon Antonakopoulos, and Lisa Zhang. Minimum-cost network design with (dis)economies of scale. In *FOCS*, pages 585–592, 2010.

[4] Matthew Andrews, Antonio Fernández, Lisa Zhang, and Wenbo Zhao. Routing for energy minimization in the speed scaling model. In *INFOCOM*, pages 2435–2443, 2010.

[5] Spyridon Antonakopoulos, Chandra Chekuri, F. Bruce Shepherd, and Lisa Zhang. Buy-at-bulk network design with protection. In *FOCS*, pages 634–644, 2007.

[6] Antonios Antoniadis, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, Viswanath Nagarajan, Kirk Pruhs, and Cliff Stein. Hallucination helps: Energy efficient virtual circuit routing. In *SODA*, 2014.

[7] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, 44(3):486–504, 1997.

[8] Baruch Awerbuch and Yossi Azar. Buy-at-bulk network design. In *FOCS*, pages 542–547, 1997.

[9] Maxim A. Babenko, Andrew V. Goldberg, Anupam Gupta, and Viswanath Nagarajan. Algorithms for hub label optimization. In *ICALP (1)*, pages 69–80, 2013.

[10] Nikhil Bansal, Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, Kirk Pruhs, and Cliff Stein. Multicast routing for energy minimization using speed scaling. In *MedAlg*, pages 37–51, 2012.

[11] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC*, pages 201–210, 2010.

[12] David Brooks, Pradip Bose, Stanley Schuster, Hans M. Jacobson, Prabhakar Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor V. Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[13] Robert D. Carr, Lisa Fleischer, Vitus J. Leung, and Cynthia A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *SODA*, pages 106–115, 2000.

[14] Deeparnab Chakrabarty, Chandra Chekuri, Sanjeev Khanna, and Nitish Korula. Approximability of capacitated network design. In *IPCO*, pages 78–91, 2011.

[15] Deeparnab Chakrabarty, Ravishankar Krishnaswamy, Shi Li, and Srivatsan Narayanan. Capacitated network design on undirected graphs. In *APPROX*, 2013.

[16] Tanmoy Chakraborty, Julia Chuzhoy, and Sanjeev Khanna. Network design for vertex connectivity. In *STOC*, pages 167–176, 2008.

[17] Chandra Chekuri, Mohammad Taghi Hajiaghayi, Guy Kortsarz, and Mohammad R. Salavatipour. Approximation algorithms for non-uniform buy-at-bulk network design. In *FOCS*, pages 677–686, 2006.

[18] Chandra Chekuri, Mohammad Taghi Hajiaghayi, Guy Kortsarz, and Mohammad R. Salavatipour. Approximation algorithms for node-weighted buy-at-bulk network design. In *SODA*, pages 1265–1274, 2007.

[19] Julia Chuzhoy and Sanjeev Khanna. An $O(k^3 \log n)$-Approximation Algorithm for Vertex-Connectivity Survivable Network Design. In *FOCS*, pages 437–441, 2009.

[20] Yefim Dinitz, Naveen Garg, and Michel X. Goemans. On the single-source unsplittable flow problem. *Combinatorica*, 19(1):17–41, 1999.

[21] Uriel Feige, Guy Kortsarz, and David Peleg. The Dense *k*-Subgraph Problem. *Algorithmica*, 29(3):410–421, 2001.

[22] Wai Shing Fung, Ramesh Hariharan, Nicholas JA Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *STOC*, pages 71–80. ACM, 2011.

[23] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. A constant factor approximation for the single sink edge installation problems. In *STOC*, pages 383–388, 2001.

[24] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In *WAOA*, 2012.

[25] Anupam Gupta, Amit Kumar, Martin Pál, and Tim Roughgarden. Approximation via cost sharing: Simpler and better approximation algorithms for network design. *J. ACM*, 54(3):11, 2007.

[26] M. Hajiaghayi, R. Khandekar, G. Kortsarz, and Z. Nutov. Capacitated network design problems: hardness, approximation algorithms, and connections to group steiner tree. In *WAOA*, 2013.

[27] Kamal Jain. A factor 2 approximation algorithm for the generalized steiner network problem. *Combinatorica*, 21(1):39–60, 2001.

[28] David R Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, 24(2):383–413, 1999.

[29] Philip Klein and R Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *Journal of Algorithms*, 19(1):104–115, 1995.

[30] Jochen Könemann, Sina Sadeghian Sadeghabad, and Laura Sanità. An LMP $O(\log n)$-Approximation Algorithm for Node Weighted Prize Collecting Steiner Tree. In *FOCS*, 2013.

[31] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach*. Addison-Wesley Publishing Company, USA, 2009.

[32] Frank Thomson Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[33] Anna Moss and Yuval Rabani. Approximation algorithms for constrained node weighted steiner tree problems. *SIAM J. Comput.*, 37(2):460–481, 2007.

[34] Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM J. Comput.*, 40(4):981–1025, 2011.

[35] Adam Wierman, Lachlan L. H. Andrew, and Ao Tang. Power-aware speed scaling in processor sharing systems. In *INFOCOM*, pages 2007–2015, 2009.

A Reduction from Energy Routing to Capacitated Network Design

We can reduce the EEVRP problem to the MCNC problem as follows. Let $q' = \sigma^{1/\alpha}$ be a parameter chosen to equalize the fixed cost σ and the routing cost f^α. From the EEVRP problem with graph $G' = (V', E')$, let $c'_1 = 2\sigma$ be the cost of routing the first q' units of flow (which includes the fixed cost), and let $c'_i = (iq)^\alpha - ((i-1)q+1)^\alpha$ for $i > 1$ be the cost of routing units $(i-1)q' + 1$ through iq'. We form an instance of the MCNC problem, $G = (V, E)$ by replacing each vertex $v \in V'$ by k/q' vertices, each of capacity $q = q'$, where the ith copy has cost c'_i. If there is an edge $(u, v) \in G'$, we replace it by the complete bipartite graph between all the copies of u and all the copies of v. Each s_i or t_i is placed as a pendant vertex off of its original vertex. We greedily attach the pendant vertices to the cheapest possible copy of the original vertex. The idea of the reduction is that the ith copy of a vertex in V corresponds to routing the ith block of q units through the corresponding vertex in V' . By the choice of parameters, it is easy to show that if we take a flow solution to EEVRP and round each flow

value up to next multiple of q', we increase the objective by at most 2^α. It is then straightforward to convert the solution to this rounded up EEVRP problem to one for the MCNC problem, with the same cost. Similarly, any (ρ_1, ρ_2)-bicriteria approximate solution to the MCNC instance (i.e. of cost at most ρ_1 times optimum, and node congestion at most $\rho_2 \cdot q$) corresponds to a $(\rho_1 \cdot \rho_2^\alpha)$-approximate EEVRP solution.