The All-or-Nothing Multicommodity Flow Problem*

Chandra Chekuri† Sanjeev Khanna‡ F. Bruce Shepherd§

November 30, 2009

Abstract

We consider the *all-or-nothing multicommodity flow* problem in general graphs. We are given a capacitated undirected graph $G = (V, E, u)$ and a set of k node pairs $s_1t_1, s_2t_2, \ldots, s_kt_k$. Each pair has a unit demand. A subset S of $\{1, 2, \ldots, k\}$ is *routable* if there is a multicommodity flow in G that simultaneously sends one unit of flow between s_i and t_i for each i in S. Note that this differs from the *edge-disjoint path* problem (EDP) in that we do not insist on integral flows for the pairs. The objective is to find a *maximum* routable subset S. When G is a capacitated tree, the problem already generalizes b-matchings, and even in this case it is NP-hard and APX-hard to approximate. For trees, a 2-approximation is known for the cardinality case and a 4-approximation for the weighted case. In this paper we show that the natural linear programming relaxation for the all-or-nothing flow problem has a poly-logarithmic integrality gap in general undirected graphs. This is in sharp contrast with EDP where the gap is known to be $\Theta(\sqrt{n})$; this ratio is also the best approximation ratio currently known for EDP. Our algorithm extends to the case where each pair s_it_i has a demand d_i associated with it and we need to completely route d_i to get credit for pair i; we assume that the maximum demand of the pairs is at most the minimum capacity of the edges. We also consider the *online admission control* version where pairs arrive online and the algorithm has to decide immediately on its arrival whether to accept it or not and the accepted pairs have to be routed. We obtain a randomized algorithm which has a poly-logarithmic competitive ratio for maximizing throughput of the accepted requests if it is allowed to violate edge-capacities by a $(2 + \epsilon)$ factor.

1 Introduction

1.1 Background

A pervasive problem in communication networks is that of allocating bandwidth to satisfy a given collection of service requests. In situations where there is limited network capacity but an abundance of requests, one must optimize over the choice of which requests to satisfy. Such *maximization* problems arise for instance in the area of bandwidth trading, or when operators carve out subnets (so-called VPNs) within their network, for sale to interested enterprise customers. Constraints on how bandwidth may be allocated vary according to the type of requesting service, as well as the technology in the underlying network. In SONET networks for instance, each request must reserve a path between its origin and destination, each link of the path

*A preliminary version of this paper appeared in *Proc. of ACM STOC*, 2004.

†Department of Computer Science, 201 N. Goodwin Ave., University of Illinois, Urbana, IL 61801. Email: chekuri@cs.illinois.edu.

‡Dept. of CIS, University of Pennsylvania, Philadelphia PA. Email: sanjeev@cis.upenn.edu.

§Mathematics and Statistics, McGill University, 805 Sherbrooke West, Montreal, QC, Canada. Email: bruce.shepherd@mcgill.ca.

supporting the traffic rate specified by the request. (In this paper, we ignore any restrictions imposed by requirements that traffic be protected against network failures.) In many cases, including in data networks, the scale of demands is large relative to the fine granularity at which it can be managed and routed. In such settings, it makes sense to design the networks in terms of (fractional) flows, rather than (unsplittable) paths.

In this paper, we discuss and compare several fundamental models related to this class of optimization problems. In each model we are given an n-node capacitated graph $G = (V, E, u)$ (undirected or directed); here u denotes a non-negative integer edge (we use edge to also refer to a directed arc) capacity vector. In addition, we are supplied with a set of k (unit) demand node-pairs $s_1t_1, s_2t_2, \ldots, s_kt_k$, each possibly with its own weight w_i. We call the s_i's and t_i's *terminals*; note that they need not be distinct. A subset S of the demands $\{1, 2, \ldots, k\}$ is *routable* if the demands in S can be simultaneously "satisfied" while obeying the capacity of the graph. The objective is to find a largest (or maximum w-weight) routable subset of demands. Depending on what restrictions we place on how demands may be routed, we obtain several distinct models.

In this setting, a fundamental problem is the *edge-disjoint paths problem*, referred to as EDP from here on. Here, a set S is routable if G contains edge-disjoint paths P_i, for each $i \in S$, such that P_i is a path from s_i to t_i. For this problem, the best approximation ratio known is $O(\sqrt{n}))$ for undirected graphs [15], and $O(\min((n \log n)^{2/3}, \sqrt{m}))$ for directed graphs [45]. The directed version of this problem is provably hard to approximate. In [25], it was shown that for any fixed $\epsilon > 0$, there is no $O(n^{1/2-\epsilon})$-approximation algorithm unless $P = NP$. The story for undirected EDP is incomplete however. Even though the approximation ratio is polynomial in n, the best known inapproximability bound states that the problem is hard to approximate to within a factor of $\Omega(\log^{1/2-\epsilon})$ unless $NP \not\subset ZPTIME(n^{O(\text{polylog}(n))})$ [1]. Closing this gap is a fundamental open problems in approximation algorithms. Such divergence in our understanding of the approximability of undirected versus directed versions of a problem is not rare, and some other examples include generalized Steiner network and the multicut problem. Partly motivated by this gap in our understanding for undirected EDP, we focus on a class of maximization problems where demands only request a *fractional* unit flow in the network. This forms a relaxation of EDP and our positive bounds in this case improve on the best known for EDP (we establish a poly-logarithmic approximation). Some of these ideas have also subsequently led to improved understanding for certain classes of undirected EDP [13, 14, 15, 16].

1.2 All-or-Nothing Multicommodity Flows

For the remainder of the paper we study the *all-or-nothing multicommodity flow problem* which was introduced in [18]; we denote this problem by AN-MCF. For this version, a set S is *routable* if there is a multicommodity flow in G that satisfies every demand $i \in S$. In other words we want to find the largest weight subset of S such that the maximum concurrent multicommodity flow for the subset is at least 1. This differs from the *maximum edge-disjoint path* problem (EDP) in that we do not insist on integral flows for the pairs. We also observe that given S, the problem of deciding whether all of S can be routed, is polynomial-time solvable via linear programming. In contrast, for EDP this same decision problem is NP-hard; moreover, while for a fixed number of demands it is polynomial time solvable by the work of Robertson and Seymour, it is NP-hard for undirected graphs even when the terminals are restricted to lie amongst a given set of four nodes [22].

In trees, the AN-MCF problem coincides with the maximum integer multicommodity flow problem, and this generalizes the general b-matching problem. This problem on trees is APX-hard [24] and a 2-approximation is known for the cardinality case (each $w_i = 1$) [24] and a 4-approximation for the weighted case [18]. For general graphs, where fractional routings come into play, there is no previous work on AN-MCF. The best known approximation factor for the general problem so far is thus $O(\sqrt{n}))$, the same as that

for EDP [15] [1]. On the other hand, recently, it was shown that for general undirected graphs, AN-MCF is $\Omega(\log^{1/2-\epsilon})$-hard to approximate for any $\epsilon > 0$ under the assumption that $NP \not\subset ZPTIME(n^{O(\text{polylog}(n))})$ [1]. As noted earlier, this is also the strongest hardness result known for undirected EDP [1].

A natural linear programming relaxation (which we denote by MCF-LP) for the all-or-nothing problem is as follows. For each demand pair i, let $\mathcal{P}_i$ denote the paths joining s_i, t_i in G. We then have a non-negative variable $x(P)$ for each $P \in \mathcal{P}_i$ and each $i = 1, 2, \ldots, k$. The objective is to maximize $\sum_i w_i \sum_{P \in \mathcal{P}_i} x(P)$ subject to the constraints $\sum_{P \in \mathcal{P}_i} x(P) \leq 1$ for each demand i, and $\sum_{P: e \in P} x(P) \leq u_e$ for each edge e. Note that this same LP is also valid for EDP and in fact all known approximation algorithms for EDP obtain their ratios directly or indirectly via the lower bound provided by the LP. Throughout, we let OPT denote the optimal value of the LP relaxation for the instance under consideration. Our main result is the following.

Theorem 1.1 *There is a polynomial-time algorithm which given an undirected n-node instance of* AN-MCF *problem, returns a solution with weight $\Omega(\frac{\text{OPT}}{\log^3 n \log\log n})$.*

In contrast to the above, an $\Omega(\sqrt{n})$ integrality gap is known for this LP when applied to EDP [24]; we mention that this gap is tight [15]. For the AN-MCF problem, in [1] an integrality gap of $\log^{1/2-\epsilon} n$ is established for the natural flow based LP relaxation. Our proof of Theorem 1.1 uses the results of Räcke on hierarchically decomposing undirected graphs to construct *demand-oblivious* routings for multicommodity flow problems [38]. Specifically, for a given n-node capacitated graph G, Räcke constructs an oblivious routing by means of a *flow template* as follows. For each pair of nodes s, t, a template must specify a fractional unit flow between s and t. Räcke's template has the following property. Given *any* demand matrix on the nodes, routing the flow according to the flow template (that is, if a pair st has a demand d, then d units of flow are routed along the fixed flow paths for the unit flow from s to t with each path getting d times its share of the unit flow), the congestion on any edge is $O(\log^3 n)$ times the optimal congestion for routing the given demand matrix. In the present setting we combine these *oblivious* routings of Räcke with other routings to avoid bottlenecks in lightly capacitated regions of the network. Azar et al. [7] and Applegate and Cohen [2] show that the *optimal* (with respect to congestion) oblivious routing can be computed in polynomial time by solving a linear program. Bansal et al. [8] give an online algorithm to compute a near optimal oblivious routing. However we need the tree-based hierarchical scheme of Räcke in our algorithm. We note that some other applications of oblivious routing schemes also need the tree based hierarchical decomposition. See Maggs et al. [37] for an application to speed up iterative solvers of linear systems.

We have thus far assumed that each demand i asks for a "unit" flow between its endpoints, but on occasion we may be supplied more generally with a demand d_i other than 1. We mention that our techniques equally apply to such *multicommodity demand flow problems* (AN-DMCF). Here, a subset S of the demands is *routable* if there exists a multicommodity flow in G that satisfies each demand $i \in S$. Note that we maintain the all-or-nothing aspect that we receive the credit w_i for demand i only if we fulfill the whole demand d_i. For an instance of the demand flow problem, we let $d_{\max} = \max\{d_i : i \in S\}$ and $u_{\min} = \min\{u_e : e \in E\}$. When $d_{\max} \leq u_{\min}$ (no *bottleneck* case), our result for unit demand carries over with a loss of a constant factor in the approximation ratio by a result in [18], whose proof is based on the grouping and scaling technique of [31]. On the other hand, when $d_{\max}$ and $u_{\min}$ are allowed to be arbitrary, it was noted in [25], that the demand flow problem cannot in general be approximated better than $O(n^{1/2-\epsilon})$ if integer flows are required. However the proof from [25] extends directly to show the same hardness even if fractional flows are permitted. In [10], the integrality gap of the LP for demand flow is shown to be $\Omega(n)$ even when G is a path. But if we allow an additive congestion of $d_{\max}$, we can once again carry over

[1] At the time of the conference version of this paper, the best known ratio was $O(\min(n^{2/3}, \sqrt{m}))$ [12].

our results, to obtain the following extension of Theorem 1.1. Recall that OPT is the value of an optimal solution to MCF-LP to the given instance.

Theorem 1.2 *There is a polynomial-time algorithm which given an undirected n-node instance of the multicommodity demand flow problem (*AN-DMCF*), returns a solution with weight at least $\Omega(\frac{\text{OPT}}{\log^3 n \log\log n})$ such that the flow on each edge e is at most $u(e)$ if $d_{\max} \leq u_{\min}$.*

Admission Control in the Online Setting: We have defined the AN-MCF and AN-DMCF problems as offline optimization problems. We also consider the *online* versions of these problems. We are given a capacitated graph G upfront, however the pairs that need to be routed arrive online. When a pair $s_i t_i$ is presented to the online algorithm, it has to immediately decide if it should accept or reject the pair, and if it accepts, it has to route one unit of flow from s_i to t_i (d_i units in the demand case). For the unit demands case, the competitive ratio of the algorithm is the worst case ratio of the number of demands routed by the online algorithm to the optimal offline algorithm. For non-unit demands, we compare the sum total of demands routed (throughput) by the online algorithm to the flow routed by the optimal offline algorithm. We assume that once the pair is accepted it stays forever. In the routing literature this is referred to as the permanent connection model [40]. We have the following theorem.[2]

Theorem 1.3 *For the* AN-MCF *and* AN-DMCF *problems there are randomized online algorithms that route $\Omega(\frac{\text{OPT}}{\log^5 n \log\log n})$ flow such that the flow on each edge e is at most $(2+\epsilon)u(e)$ if $d_{\max} \leq u_{\min}$.*

The approximation ratios in this paper improve by a $\log n$ factor for graphs that exclude minors of fixed size, in particular for planar graphs.

1.3 Related Work

The all-or-nothing flow problem in general graphs was introduced in [18]. They obtain results in the context where the supply network is a tree. Of course, in the tree case, there is no advantage to allowing fractional routing: each commodity must route all of its demand on a single path. General multicommodity flow problems that require each demand to be routed on a single path are called *unsplittable flow problems* (UFP). EDP is a special case of UFP where all demands are 1. Multicommodity flow and disjoint path problems, along with several variants and special cases, have been extensively studied both for their fundamental importance to combinatorial optimization and their applications to a variety of areas such as network routing, VLSI layout, parallel computing and many others. We refer the reader to [41, 34, 23, 29, 40, 44, 25, 32, 38, 12] for some pointers. Here we confine ourselves to mentioning the directly relevant literature on EDP and UFP and their online variants.

We first consider the offline case. For both EDP and UFP the best known approximation ratio in general graphs is $O(\sqrt{n}))$ for undirected graphs [15] and $O(\min((n\log n)^{2/3}, \sqrt{m}))$ for directed graphs [45, 44]. EDP and UFP are $\log^{1/2-\epsilon} n$-hard in undirected graphs [1], and are $\Omega(n^{1/2-\epsilon})$-hard in directed graphs [25]. If all pairs share a source, then EDP reduces to the single commodity maximum flow problem and can be solved in polynomial time. Even for UFP, the single source case is tractable in that most variants have constant factor approximation algorithms [28, 31, 21]. For EDP and UFP, large capacities help. Randomized rounding [41] yields constant factor approximation algorithms if $d_{\max} \leq u_{\min}/(\Omega(\log n))$. More generally if $d_{\max} \leq u_{\min}/B$ for integer B, then an approximation ratio of $O(n^{1/B})$ is achievable [44, 6]. We mention

[2]The conference version of this paper incorrectly claimed slightly stronger results for the online problem.

that all of the offline bounds for EDP and UFP also apply to the integrality gap of MCF-LP. It is known however that MCF-LP has an integrality gap of $\Omega(\sqrt{n})$ [24] and this is tight [15].

Now we consider the online versions of EDP and UFP. These problems have applications in ATM networks and are usually referred to as admission control for virtual circuit routing. A variety of models exist based on whether the circuits (pairs) are permanent or temporary and in the temporary case whether their durations are known or unknown. We refer the reader to the survey [40] for more details. Here we confine ourselves to the case of permanent connections. In this case online EDP asks to maximize the number of pairs accepted compared to the offline optimal. For UFP we are interested in maximizing the *throughput*, that is, the sum of demands accepted. For both these problems the best competitive ratios are comparable to the offline approximation ratios. However, if capacities are large relative to the demands (a realistic assumption in practice) the algorithm of Awerbuch, Azar, and Plotkin [5] is $O(\log n)$-competitive provided $d_{\max} \leq u_{\min}/\Omega(\log n)$.

In the context of EDP and UFP we can also consider the problem of routing a given set of demands S so as to minimize the *congestion* of the routing. Congestion of a routing is defined as the maximum over all edges of the ratio of the flow on the edge to its capacity. If flows can be split, then the minimum congestion routing can be computed by solving a linear program. However for integral flow paths or for unsplittable flow, randomized rounding is the only effective algorithm known and yields an $O(\log n/\log\log n)$ approximation [41]. In directed graphs this gap is known to be tight [35]; that is there are instances where S can be fractionally routed with congestion 1 while any integral routing has $\Omega(\log n/\log\log n)$ congestion. Recently Chuzhoy et al. [19] have shown that, in directed graphs, the minimum congestion to route a given set of demands is hard to approximate within a factor of $\Omega(\log n/\log\log n)$, effectively closing the approximability of the problem. Despite this hardness, even in the case where demands to be routed arrive online, an $O(\log n)$-competitive ratio is possible [3]. More recently, Räcke [38] obtained a randomized $O(\log^3 n)$-competitive algorithm for minimizing congestion that is *oblivious*; this result of Räcke [38] is the starting point for our work. The ratio has been improved to $O(\log^2 n \log\log n)$ by Harrelson, Hildrum, and Rao [27], and very recently Räcke [39] has obtained an optimal $O(\log n)$ bound.

2 Preliminaries

We consider multicommodity flow problems in a given capacitated graph $G = (V, E, u)$ and we assume that each capacity $u(e)$ is an integer. We let $n = |V|$ and $m = |E|$. For any graph G and a node subset $S \subseteq V$, we denote by $\delta_G(S)$, or simply $\delta(S)$ if G is clear from the context, the set of edges of G with exactly one endpoint in S.

An *instance* of a *multicommodity flow* problem consists of a graph $G = (V, E, u)$ and a collection of undirected demand pairs $s_i t_i$ for $i = 1, 2 \ldots, k$ where each $s_i, t_i \in V$. Without loss of generality we assume that demand pairs are distinct. Otherwise, we can add dummy nodes to ensure this. In addition, we may have a weight w_i associated with each demand. Let $\mathcal{P}_i$ denote the paths joining s_i, t_i in G. For each path P in G, let $x(P)$ denote a non-negative variable. An assignment x is called a *multicommodity flow* and is said to *satisfy* demand i, if $\sum_{P \in \mathcal{P}_i} x(P) = 1$. (If in addition, a demand has an associated value d_i, then the right hand side changes from 1 to d_i.) The *load* of x on the edge e is $l(e) = \sum_i \sum_{P \in \mathcal{P}_i : e \in P} x(P)$ and the *congestion* on e is $l(e)/u(e)$. The flow x is *feasible* if the congestion on each each edge is at most 1. A set S of demands is *routable* if there is a feasible flow that satisfies each demand $i \in S$. We say that a demand i is *routed* on path P, if $x(P) > 0$. We also refer to a flow as being obtained by *routing* a demand on certain paths.

The objective of the all-or-nothing multicommodity flow problem is to find a maximum weight routable

set. The natural relaxation for this problem is to find a feasible flow that maximizes $\sum_i \sum_{P \in \mathcal{P}_i} w_i x(P)$. To facilitate further discussion, we introduce a new variable x_i for each pair i where $x_i = \sum_{P \in \mathcal{P}_i} x(P)$, and write the LP relaxation below.

$$\begin{aligned} \max \sum_{i=1}^{k} w_i x_i \quad \text{s.t} & \qquad (1) \\ x_i - \sum_{P \in \mathcal{P}_i} x(P) &= 0 \qquad 1 \le i \le k \\ \sum_{i=1}^{k} \sum_{P: e \in \mathcal{P}_i} x(P) &\le u(e) \quad e \in E \\ x_i, x(P) &\in [0,1] \quad 1 \le i \le k, P \in \cup_i \mathcal{P}_i \end{aligned}$$

We prefer this path-formulation for flows to the compact formulation for ease of presentation. Either formulation admits a polynomial-time algorithm to solve this relaxation. We let OPT denote the value of an optimum solution to the above relaxation. We assume that $u(e) \le m$ for each i because of the following reason. We can find a basic solution (1) in polynomial time. One can easily argue that the number of demands for which $x_i \in (0,1)$ is at most m. If the demands for which $x_i = 1$ form $\Omega(1)$ of OPT, then we immediately have a good approximate solution. Thus the difficult case is when most of the LP's profit is accrued through the (at most m) fractionally routed demands. In this case, we may restrict to these demands and note that the total load on any edge is at most m. We may thus assume from now on, that the edge capacities have been appropriately reduced and are hence polynomially bounded in $|V|, |E|$. We also assume that each node v is the endpoint of at most one demand pair in the input instance; it is easy to add dummy nodes to ensure this.

Consider two non-negative vectors π, π' defined on the nodes of G such that $L = \sum_v \pi(v) = \sum_v \pi'(v)$. By *distributing* (or *routing*) $\pi(v)$ units of flow from v to π', we refer to a (single-source) flow f with the property that for each v', there is a flow of value $\pi(v)\pi'(v)$ from v to v'. By distributing flow from π to π', we implicitly refer to a multicommodity flow such that for each v we are distributing $\pi(v)$ units of flow from v to π'. We sometimes say that we are sending flow from the distribution π to the distribution π'.

2.1 Oblivious Routing and Räcke Trees

For a graph G, we call a capacitated tree $T = (V_t, E_t, u_t)$ with a specified *root* node r_t, a ***Räcke tree*** for G if the leaves of T are precisely the nodes of G. In particular, r_t is not one of the leaves. Note that each edge $e \in T$, determines, in a natural way, a partition $(S, V - S)$ of G's nodes, corresponding to the nodes of G that appear in the leaves of the two resulting subtrees in $T - e$. The capacity of an edge e in T, denoted by $u_T(e)$, is defined as the total capacity on edges in the cut $\delta_G(S)$. We also let $h(T)$ denote the height of T. Each Räcke tree induces a canonical routing strategy for G, independent of the set of demands. These (demand) oblivious routings are discussed in more detail later; we first describe some of their basic properties.

An *oblivious routing* strategy consists of defining a unit flow F_{st} for each pair of nodes s, t; this collection of flows is called a routing *template*. Given a collection of demands of say d_i between s_i and t_i, a template induces a flow as follows. For each i, and each $P \in \mathcal{P}_i$, route $F_{st}(P)d_i$ amount of flow along P. The multicommodity flow obtained is said to be *routed according* to the oblivious routing.

For each Räcke tree T, there exists a value $\alpha(G,T)$ that measures the quality of routing according to the demand-oblivious routes.[3] Consider a set X of "demands" in G, i.e., X is a multiset of undirected edges uv with $u, v \in V$. The *congestion* of X on an edge e of T is the congestion on e resulting from routing each demand of X along its unique path in T. We say X is *routable on* T if the congestion is at most 1 on each edge of T.

Theorem 2.1 (Räcke [38]) *Let T be a Räcke tree for G and X a set of demands. If X is routable on T, then Räcke routing these demands in G results in congestion at most $h(T)\alpha(G,T)$ on each edge of G. Moreover, there exists a Räcke tree T with $\alpha(G,T) = O(\log^3(n))$ and $h(T) = O(\log n)$.*

Räcke's original construction did not yield a polynomial-time algorithm to find such a T. Subsequently, two independent papers [9] and [27] obtained polynomial-time algorithms. In [27] it is shown how to construct T with $\alpha(G,T) = O(\log^2 n \log\log n)$ and $h(T) = O(\log n)$. In the following, we let $\alpha(G)$ denote the minimum of $\alpha(G,T)$ over some computable class of Räcke trees T for G after it has possibly been reduced as above.

2.2 The Räcke Routing Scheme

In this section we describe the oblivious routing strategy of Räcke [38] based on hierarchical decomposition of the given graph. We need a refinement of Räcke's result, and hence we delve into the details of how the oblivious routings are constructed. In doing so, we follow the approach and methods of [9] which simplified that of [38], in addition to giving a polynomial time construction of the decomposition in [38].

Let $T = (V_t, E_t, u_t)$ be a Räcke tree for $G = (V, E, u)$. We denote by T_v the subtree of T rooted at v and let L_v denote the leaves of the subtree T_v and G_v, the subgraph of G induced by L_v. Recall then, that $u_t(e)$ equals the capacity of the cut $\delta(L_v)$ in G. A key feature of Räcke's routing is that any demand st with $s, t \in L_v$ is entirely routed within the subgraph G_v. We now describe the oblivious routing of Räcke in detail.

First we set up some notation. Consider a non-leaf node $v \in V_t$ with descendants $v_1, v_2, \ldots v_k$. We deal with two important sets of edges induced by such a node v. The first is the set of edges in the cut induced by the leaves L_v, that is, $\delta_G(L_v)$. Second, we say an edge is *separated by* v if either it lies in $\delta(L_v)$ or it has its endpoints in distinct subtrees T_{v_i}, T_{v_j}. Let $S(v)$ denote the set of such edges. A measure of these edges' capacity is critical in the following; we denote by $U(v)$ the sum $\sum_i u(\delta(L_{v_i}))$. Note that the capacity of each edge in $S(v)$ is accounted for twice except for those in $\delta(L_v)$ which are accounted for once. For a node $a \in V(G)$ such that $a \in L_v$, we denote by $u_v^{\text{cut}}(a)$ the total capacity on the edges incident to a that lie the cut $\delta_G(L_v)$. In other words, the quantity $u(\delta_G(a) \cap \delta_G(L_v))$. We denote by $u_v^{\text{sep}}(a)$ the quantity $u(\delta_G(a) \cap S(v))$. We denote by $\pi_v^{\text{cut}}(a)$ and $\pi_v^{\text{sep}}(a)$ the quantities $u_v^{\text{cut}}(a)/u(\delta_G(L_v))$ and $u_v^{\text{sep}}(a)/U(v)$ respectively. Note that $\sum_{a \in L_v} \pi_v^{\text{cut}}(a) = \sum_{a \in L_v} \pi_v^{\text{sep}}(a) = 1$. We sometimes refer to $\pi_v^{\text{sep}}()$ as the *distribution at* v.

We describe a strategy for routing traffic from a node $s \in V$ to a node $t \in V$. Let $P = (s = v_0, v_1, \ldots, v_p, v_{p+1}, \ldots, v_l = t)$ be the unique path in T joining s and t, where v_p is the "high point" of the path (the least common ancestor of s and t in T). We construct a unit flow from s to t by concatenating a collection of (multicommodity) flow vectors that act as building blocks. We describe these now.

Each edge and each internal node of P sponsor a "transformation" of flow as we now describe. An edge (y, v) (where y is a child of v) transforms the node "supply" vector π_y^{cut} on nodes of L_y into a node "demand" vector π_v^{sep} on nodes of L_v. This is called a *spreading* transformation. In other words, it is distributing flow from π_y^{cut} to π_v^{sep} (see Section 2). This transformation is accomplished by a multicommodity

[3] We note that α can be computed by solving $|V_t - V| \leq n$ multicommodity flow problems in G.

flow denoted by $f_{(y,v)}$ (called the *spreading* flow) in G_v. For each $a \in L_y$ and $b \in L_v$, the flow $f_{(y,v)}$ routes $d_{(y,v)}(a,b) = \pi_y^{\text{cut}}(a) \cdot \pi_v^{\text{sep}}(b)$ flow between a and b. We also denote by $f_{(v,y)}$ the flow obtained by "reversing" the orientation on the flow paths in $f_{(y,v)}$.

If v is an internal node of P, then v transforms the node supply vector π_v^{sep} on the nodes of L_v into a demand π_v^{cut} on the nodes of L_v. This is called a *concentrating* transformation and is accomplished by a multicommodity flow f_v (called the *concentrating* flow) on G_v that routes for $a, b \in L_v$, an amount of flow $d_v(a,b) = \pi_v^{\text{sep}}(a) \cdot \pi_v^{\text{cut}}(b)$ between a and b.

Note that the fractions π_v^{cut} and π_v^{sep} used above, and hence the multicommodity flows are independent of the choice of demand st.

We now return to constructing a unit flow from s to t. This flow, denoted by $F_{s,t}$, is obtained from the path P by merging the flows $f_{(v_0,v_1)}$, f_{v_1}, $f_{(v_1,v_2)}$, f_{v_2}, …, $f_{v_{p-1}}$, $f_{(v_{p-1},v_p)}$, $f_{(v_p,v_{p+1})}$, $f_{v_{p+1}}$, $f_{(v_{p+1},v_{p+2})}$, $f_{v_{p+2}}$, …, $f_{(v_{l-1},v_l)}$.

At this point, we have not mentioned anything about the actual flow paths for these flow vectors. A key aspect of Räcke's strategy is that these flows are used as building blocks for many different demand pairs. He shows that the resulting template is efficient (with regards to congestion) as long as for each non-leaf node v in T, there is an efficient routing for a canonical multicommodity problem in G_v called the *exchange flow problem for* v. For each non-leaf node $v \in T$, we consider a multicommodity flow g_v in the graph G_v for the following *exchange flow problem*. For each pair of nodes a, b in L_v, the exchange flow must route $D_v(a,b) = u_v^{\text{sep}}(a) \cdot u_v^{\text{sep}}(b)/U(v) = \pi_v^{\text{sep}}(a) \cdot \pi_v^{\text{sep}}(b) \cdot U(v)$ (we write $D(a,b)$ if v is clear from the context) amount of flow between a and b. We denote by $q_v \leq 1$ a *throughput guarantee* for this problem, indicating that that we may route $q_v D(a,b)$ times each of these demands simultaneously in the graph G_v (in other words the maximum concurrent flow for the demand matrix D_v).

The heart of Räcke's technique is to show that we may obtain our flow vectors $f_v, f_{(y,v)}$ from a solution for these exchange flow problems (after scaling by q_v). One may show that if some set of demands can be routed in T with congestion 1, then routing along the flow vectors $F_{s,t}$ obtained through the exchange flows has congestion $O(\alpha(G,T))$ in G.

Lemma 2.2 *Let T be a Räcke tree for G and let X be a set of demands that is routable in T and such that for each $st \in X$, the path joining s, t in T contains r. For $v \in V$, let $X(v)$ be the number of demands in X incident to v. Then, in G we can distribute $X(v)$ units of flow from each $v \in V$ to the Räcke distribution at r (that is, we can route $X(v)\pi_r^{sep}(u)$ flow from v to each $u \in L_r$) with congestion $\alpha(G,T)$.*

Proof: Consider the Räcke routing of X in G. For any pair $st \in X$, r is the least common ancestor. It follows that the Räcke routing for st consists of distributing one unit of flow from s to its Räcke distribution π_r^{sep} at r and similarly for t. Note that π_r^{sep} is agnostic to the terminal from which flow originates. Since X is routable in T with congestion 1, it follows that the congestion in G for Räcke routing X is at most $\alpha(G,T)$. ■

3 The Offline Algorithm

In this section, we present an $O((\alpha(G)\log n))$-approximation algorithm for the all-or-nothing multicommodity flow problem. We first develop a scheme that allows us to route a large fraction of demands with low congestion. Specifically, we show that for any $\epsilon(n) > 0$, we can obtain a solution of weight $\Omega(\frac{\epsilon(n)\text{OPT}}{\log^3 n \log\log n})$ such that the flow on any edge e is at most $\epsilon(n)u(e)+1$. We then design a more sophisticated routing scheme that allows us to obtain a solution of weight $\Omega(\frac{\text{OPT}}{\log^3 n \log\log n})$ without exceeding any edge capacities.

3.1 Starting Point

We start by describing a simple algorithm that is the starting point of our approach:

(a) Construct a Räcke tree T for G.

(b) Scale down all tree capacities by setting $\overline{u}_t(e) = \lfloor u_t(e)/\alpha(G) \rfloor$. Let $\overline{T}$ denote this new tree.

(c) Solve the all-or-nothing LP induced on $\overline{T}$, and let $\mathrm{OPT}(\overline{T})$ denote the value of an optimal LP solution.

(d) Find a set X of weight at least $\Omega(\mathrm{OPT}(\overline{T}))$ that can be routed integrally [24, 18].

(e) For each routed pair $st \in X$, send 1 unit of flow from s to t in G using the flow function $F_{s,t}$ specified by the Räcke routing.

Lemma 3.1 *If all edges in T are of capacity at least $\alpha(G)$, there is a simple $O(\alpha(G))$-approximation algorithm for any instance of* AN-MCF *problem in G.*

Proof: Let $\mathrm{OPT}(G)$ denote optimal LP solution value for our instance of AN-MCF. Clearly $\mathrm{OPT}(T) \geq \mathrm{OPT}(G)$. Further, by our assumption that $u_t(e) \geq \alpha(G)$ for all $e \in T$, $\lfloor u_t(e)/\alpha(G) \rfloor \geq u_t(e)/2\alpha(G)$ and hence $\mathrm{OPT}(\overline{T}) \geq \frac{\mathrm{OPT}(T)}{2\alpha(G)}$. We may compute X so that $w(X)$ is at least $\mathrm{OPT}(\overline{T})/4$ by [18]; in the cardinality case by [24] we can choose X so that $|X| \geq \mathrm{OPT}(\overline{T})/2$. Hence X has weight (respectively cardinality) at least $\mathrm{OPT}(G)/4\alpha(G)$ (respectively. $\mathrm{OPT}(G)/4\alpha(G)$). Note that if X is feasible on $\overline{T}$, then in T we can route α units of demand for each $i \in X$. Thus in G we can route α demand for each pair in X with congestion at most α. Hence we can scale down and fractionally route a unit amount for each i in X with congestion 1. ■

We subsequently often refer to an instance as consisting of a triple (G, T, X) where X is a set of demands that can be feasibly routed on an associated Räcke tree T for G. Recall also that each node in V is the endpoint of at most one demand pair in X.

3.2 Preprocessing the Instance

We have seen that the simple approach outlines above does not apply when there are edges in T of capacity less than $\alpha(G)$. In the following we assume (using [24, 18]) that we are starting with a set of demands X of weight $\Omega(\mathrm{OPT}(T))$ which is feasible for the tree T, i.e., can be integrally routed on the capacitated Räecke tree T.

For a node v of T, we denote by ℓ_v, its *level* or the distance from the root. For a demand $st \in X$, we define its level to be the level of the least common ancestor of s and t in T, denoted by $\mathrm{lca}(s, t)$. An instance (G', T', X') of AN-MCF is called NICE, if X' can be routed on T' with congestion 1 and the level of each demand is 0, that is, for each $(s, t) \in X$, $\mathrm{lca}(s, t)$ is the root of T'.

Lemma 3.2 *Given a feasible instance (G, T, X) of* AN-MCF*, we can obtain in polynomial-time $p \leq n$* NICE AN-MCF *instances (G_i, T_i, X_i), $1 \leq i \leq p$ such that (a) each G_i is a subgraph of G and each $X_i \subseteq X$, (b) G_i's are pairwise node-disjoint, and (c) $\sum_i w(X_i) = \Omega(\frac{w(X)}{h(T)})$.*

Proof: By pigeonhole principle, there exists a subset $X' \subseteq X$ with $w(X') \geq w(X)/h(T))$ such that all demands in X' have the same level. We focus on the set X' from here on, and let ℓ be the common level for pairs in X'. Let $r_1, r_2, \ldots, r_p$ be the nodes of T at level l. For each i, let T_i denote the tree T_{r_i} and

X'_i denote the demands of X' with both ends in T_{r_i}. We also let G_i be the subgraph of G induced by the leaves of T_i. Note that the graphs G_i are node-disjoint, and that T_i is a valid Räcke tree for G_i. Thus the instances (G_i, T_i, X_i), $1 \leq i \leq p$ are disjoint, feasible, have the property that the level of each demand is 0, and collectively, they satisfy $\sum_i w(X_i) = \Omega(\frac{w(X)}{h(T)})$. ■

3.3 Low Congestion Routings for NICE Instances

Our next goal is to show that given a NICE instance (G, T, X) and any given $\epsilon(n) > 0$, we can find a large subset $Z \subseteq X$ that can be routed in G with *low congestion* – a flow of $\epsilon(n)u(e) + 1$ on each edge e. We start by establishing a simple lemma about grouping subsets of nodes in an edge-disjoint manner.

Lemma 3.3 *Let G be a connected graph with a weight function $\rho : V \to [0, W]$ such that $\sum_{v \in V} \rho(v) \geq W$. We can find $p > (\sum_{v \in V} \rho(v)/2W - 1/2)$ edge-disjoint connected subgraphs $H_1 = (V_1, E_1)$, $H_2 = (V_2, E_2)$, ..., $H_p = (V_p, E_p)$ and node-disjoint subsets $S_1, S_2, ..., S_p$ such that for each i: (i) $S_i \subseteq V_i$ and (ii) $W \leq \sum_{v \in S_i} \rho(v) \leq 2W$.*

Proof: The *weight* of a node subset X is $\sum_{v \in X} \rho(v)$ and is denoted by $\rho(X)$. X is called *heavy* if $\rho(X) \geq W$. Let T' be a spanning tree of G rooted at an arbitrary node r. For any node x let T'_x be the subtree rooted at x. Choose some x such that $V(T'_x)$ is heavy, but none of x's children has this property (x exists since $\rho(V) \geq W$). Let $x_1, x_2, \ldots, x_\ell$ be the children of x in T'. Let j be the smallest index such that $\gamma = \rho(x) + \sum_{i=1}^{j} \rho(V(T'_{x_i})) \geq W$. Since $\rho(V(T'_{x_i})) < W$ for $1 \leq i \leq \ell$ and $\rho(x) \leq W$, $W \leq \gamma < 2W$. We obtain H_1 by taking the tree obtained from the union of $T'_{x_1}, T'_{x_2}, \ldots, T'_{x_j}$ with x. We set S_1 to be the nodes in H_1 with non-zero weight. We remove $T'_{x_1}, T'_{x_2}, \ldots, T'_{x_j}$ from T' and set $\rho(x) = 0$ to obtain a new tree T''. We iterate this process on T'' until the total weight of the remaining nodes falls below W. Note that a node can participate in many subgraphs that we create, however it can have positive weight only in the first subgraph. At the end of the process we have the desired edge-disjoint connected subgraphs $H_1, H_2, \ldots, H_p$ and disjoint sets of nodes $S_1, S_2, \ldots, S_p$ with $S_i \subseteq V(H_i)$. From the construction, for $1 \leq i \leq p$, $W \leq \rho(S_i) < 2W$. The final tree has weight at most W. Therefore $2pW + W > \rho(V)$, and hence, $p > \rho(V)/(2W) - 1/2$. ■

Let S be the set of terminals for X. Note that a terminal is is the endpoint of a single demand in X.

Lemma 3.4 *Let (G, T, X) be a nice instance. We can distribute $\frac{\epsilon}{\alpha(G)}$ flow from each $v \in S$ according to the Räcke distribution π_r^{sep} on $V(G)$ such that the flow on any edge $e \in G$ is at most $\epsilon u(e)$.*

Proof: Following the proof of Lemma 2.2, we know that X can be routed in G, using Räcke routing, with a congestion of $\alpha(G)$. Scaling down the flow by $\alpha(G)/\epsilon$ and noting that each demand is distributing its flow to the distribution π_r^{sep}, gives us the desired result. ■

For each $v \in V(G)$, we define $\beta(v) \in [0, 1)$ as follows; if v is a terminal $\beta(v) = \epsilon/\alpha(G)$, else $\beta(v) = 0$. Let $\beta = \sum_v \beta(v)$. We use Lemma 3.3 with $W = 1$ to group the terminals S into disjoint *clusters* $S_1, S_2, \ldots, S_p$ where $p = \max(1, \Omega(\beta))$ such that $\sum_{v \in S_i} \beta(v) \geq 1$. Note that the lemma guarantees that each cluster S_i has at least $\alpha(G)/\epsilon$ and at most $2\alpha(G)/\epsilon$ terminals.

We now identify a subset of $Z \subseteq X$ that can be routed in G with low congestion. We order the pairs $st \in X$ in non-increasing order of their weight and consider them one by one and build a set of demands Z which we ultimately route. Initially $Z = \emptyset$. We also maintain a set of *active* clusters: initially all clusters are active. Let st be the current demand pair. We say that the pair st is *feasible at s* if s is in an active cluster.

We add st to Z if both s and t are feasible. Otherwise we reject the pair st. If st is added to Z we mark the cluster containing s as inactive. We do similarly for t. Note that both s, t could be in the same cluster.

Lemma 3.5 *The procedure produces $Z \subseteq X$ such that $w(Z) = \Omega(\frac{\epsilon}{\alpha(G)} w(X))$. Each cluster S_i, contains $0, 1$ or 2 terminals for demands in Z. If it contains 2, then they are the end points of the same demand.*

Proof: We charge the rejected pairs in $X \setminus Z$ to those in Z. Suppose st is rejected because the cluster containing s was already marked. We charge st to the pair $s't' \in Z$ that marked the cluster of s. Note that the weight of pair $s't'$ is no smaller than that of st since it was considered earlier in the ordering. Since each cluster has as most $2\alpha(G)/\epsilon$ terminals, a pair in Z is charged by at most $2\alpha(G)/\epsilon$ other pairs. This proves that $w(Z) = \Omega(\frac{\epsilon}{\alpha(G)} w(X))$. The second part of the lemma is easy to see from the marking procedure. ■

Lemma 3.6 *The subset Z can be routed in G such that on each edge e, the flow is at most $1 + \epsilon u(e)$.*

Proof: Consider a pair st in Z. Suppose first that s, t lie in a common cluster. In this case we simply connect s and t by a path in this cluster. Otherwise, each of s, t sends a unit of flow to the root's distribution π_r^{sep}. Let S_i, S_j be the active clusters that contained s and t respectively when st was added to Z. We distribute one unit of flow from s to the nodes in S_i such that each node $v \in S_i$ gets a flow $\gamma(v) \leq \beta(v)$. This is feasible since $\sum_{v \in S_i} \beta_v \geq 1$ and S_i is connected. Each node $v \in S_i$ then sends $\gamma(v)$ flow to the root's distribution π_r^{sep}. The terminal t similarly sends its unit of flow to the root's distribution π_r^{sep} using nodes in its cluster S_j.

We bound the congestion of an edge e as follows. The edge e can belong to at most one cluster. Within a cluster it can be used to route at most one unit of flow from a terminal. Now consider flow on e from the routings from terminals to the root distribution. For v, let $f(v)$ be the total flow that is routed from v to π_r^{sep}. From our routing above it is easy to see that $f(v) \leq \beta(v) \leq \frac{\epsilon}{\alpha(G)}$. Now we can apply Lemma 3.4 to claim that this flow on e is at most $\epsilon u(e)$. Hence the total flow on e is at most $1 + \epsilon u(e)$. ■

Theorem 3.7 *Given an instance of* AN-MCF *(G, X) and a Räcke tree T, there is a polynomial time algorithm that routes $\Omega(\frac{\epsilon \text{OPT}}{h(T)\alpha(G,T)})$ demands from X where* OPT *is the optimum LP solution for X on G such that the flow on any edge e of G is at most $1 + \epsilon u(e)$.*

Using [27] there is a Räcke tree T for G such that $\alpha(G, T) = O(\log^2 n \log\log n)$ and $h(T) = O(\log n)$; hence we obtain an approximation ratio of $O(\log^3 n \log\log n/\epsilon)$. For planar graphs or graphs that exclude a constant size minor, [27] gives a Räcke tree T such that $\alpha(G, T) = O(\log n \log\log n)$ and $h(T) = O(\log n)$ and hence we obtain an approximation ratio of $O(\log^2 n \log\log n)$ for these graphs.

3.4 Congestion 1 Routings for NICE Instances

We now show how to find a routing that does not violate the capacities. To simplify the description we focus on the cardinality version of the problem; we mention, as needed, the simple modifications that are needed for the weighted version. Again we work with NICE instances. Let (G, T, X) be a NICE instance. We say that a capacitated graph G is 2-edge connected if the uncapacitated multi-graph obtained by making $u(e)$ copies of each edge e is 2-edge connected.

Theorem 3.8 *Given a* NICE *instance (G, T, X) of* AN-MCF*, there is a polynomial-time algorithm that routes $\max\{1, \Omega(|X|/\alpha(G, T))\}$ pairs from X in G without violating capacities.*

The rest of this section is devoted to the proof of the above theorem. We assume without loss of generality that $u(e) = 1$ for all e. The basic idea for the proof of Theorem 3.8 is similar to that in Section 3.3. Given a nice instance (G, T, X) each of the pairs in X can route $\epsilon/\alpha(G)$ flow to the root distribution such that each edge e has a load of at most $\epsilon u(e) = \epsilon$. In Section 3.3 we used tree-based clustering. Each terminal that we chose to route, then sent one unit of flow to $\Omega(\alpha(G)/\epsilon)$ other terminals in its cluster. This additional routing required one unit of capacity on the edges in each cluster. In this section we present a more complicated clustering in which a terminal can send one unit of flow to $\Omega(\alpha(G))$ other terminals using at most $1/2$ unit of capacity on the edges of its cluster subgraph (which may no longer be a tree). Choosing $\epsilon = 1/2$ ensures that no edge capacity is violated.

Recall that that each node $s \in V(G)$ is incident to at most one demand pair in X. From the given nice instance (G, T, X) we define a node weight function ρ as follows: $\rho(v) = 1/(2\alpha(G))$ if v is a terminal in X and $\rho(v) = 0$ otherwise. Hence a node v is a terminal if and only if $\rho(v) > 0$. For a set S, $\rho(S) = \sum_{v \in S} \rho(v)$. Therefore $\rho(V) = |X|/\alpha(G)$. For a graph H and weight vector ρ', a single-source unit flow vector f from node v is *feasible with respect to* π' if the flow on any edge is at most $1/2$ and the total flow terminating at a node $y \neq v$ is at most $\rho'(y)$. A node v for which a feasible flow exists is called a *center* in H. A node v is a center in H *with respect to a subset* $S \subset V(H)$ if the flow is constrained to be terminated only at nodes in S.

Lemma 3.9 *Let $H = (V, E)$ be a connected graph with a node weight function $\rho : V \rightarrow \mathcal{R}^+$ such that $\rho(V) \geq 1$. Then there is a center v in H with respect to ρ.*

Proof: We prove this for the case when H is a tree T and $\rho(V) = 1$. The proof is by induction on $n = |V|$. For $n \leq 2$ the claim can be easily verified. Assume $n > 2$. Then there exists a node v of degree at least 2. Consider the trees $T_1, T_2, \ldots, T_k$ produced by the removal of v from H. If $\max_i \rho(V(T_i)) \leq 1/2$, then v is the desired center. Otherwise let T_1 be such that $\rho(V(T_1)) > 1/2$. Let T' be a tree obtained by removing $T_2, \ldots, T_k$ from H and setting $\rho(v) = \rho(v) + \sum_{i=2}^{k} \rho(V(T_i))$. By induction, the tree T' has a center v'. Note that $v' \neq v$ since v is a leaf in T' and $\rho(v) < 1/2$, and hence $v' \in V(T_1)$. It can be easily checked that v' is a center for T as well. ■

We need a slight relaxation of the definition of a center. We call a node v a *pseudo-center* in a subgraph H of G if either v is a center or if the following is true: there is a single source flow of one unit that originates at v and such that the total flow terminating at a node $y \neq v$ is at most $\rho(y)$ and the flow on any edge other than a cut edge of G is at most $1/2$. Note that the flow on a cut edge can be up to 1.

Lemma 3.10 *Let G be a unit capacity graph and X a set of node pairs. Let C be the set of cut edges in G. Let f be a multicommodity flow in G that sends one unit of flow for each pair in X with the condition that the flow on any edge is less than 2 if it is in C, and at most 1 otherwise. Then there is another multicommodity flow f' that sends one unit of flow for each pair in X and such that the load on any $e \in E$ is at most 1.*

Proof: Given f we decompose it into flow paths for each pair. We ensure that all the flow paths are simple paths. Let f' be the resulting flow. We claim that f' satisfies the desired properties. We have that the flow on any edge under f' is at most that under f, so all we need to show is that the flow under f' is at most 1 for each $e \in C$. Let G_1 and G_2 be the components of $G - e$. If a pair $st \in X$ is such that $s, t \in G_1$ or $s, t \in G_2$ then it is clear that the flow between s and t does not use e since the flow paths are simple. Hence the only flow that uses e is for a pair $s't'$ such that $s' \in V(G_1)$ and $t' \in V(G_2)$. For such a pair the whole unit of their flow crosses e. Since the load on any edge is less than 2, there is at most one such pair, and hence the flow on e is at most 1. ■

Our goal is to prove the following clustering lemma.

Lemma 3.11 *Given a nice instance (G, T, X) we can find a subset $Z \subset X$ with $|Z| = \Omega(\rho(V) = |X|/\alpha(G))$ with the following properties. There is a collection of connected edge-disjoint subgraphs $H_1, H_2, \ldots, H_p$ such that (i) each H_i contains terminals for at most one pair in Z, (ii) each H_i contains a set of nodes S_i such that the S_i are node-disjoint, and (iii) for each pair $st \in Z$, if $s \in H_i$ then either s is a pseudo-center of H_i with respect to ρ and S_i, or t is also in H_i (and similarly for t).*

Outline of Congestion 1 Routing. We prove Theorem 3.8 from the above lemma. A pair $st \in Z$ is *mated* if there is a H_i with $s, t \in H_i$. From the properties of the above lemma, we can route all the mated pairs in an edge disjoint fashion. We now show that the unmated pairs in Z can also be routed. It thus follows that we can route at least half the pairs in Z with congestion 1 in G.

Let st be an unmated pair. We have that $s \in H_i$ and $t \in H_j$ with $i \neq j$ and further H_i and H_j contain no other terminals from Z. Since s is a pseudo-center in H_i with respect to S_i, it can send one unit of flow to S_i such that each node $v \in S_i$ receives flow of at most $\gamma(v) \leq \rho(v)$ and the total flow on each edge $e \in H_i$ is at most $1/2$, unless e is a cut edge of G in which case the flow can be up to 1. Since the S_i's are node disjoint, the total flow received by a node v is at most $\gamma(v)$. Now each node v sends $\gamma(v)$ flow to the root's distribution π_r^{sep}. Lemma 3.4 implies that we can route this flow sending at most $1/2$ unit of flow on each edge. Thus, all the terminals in unmated pairs can simultaneously route one unit of flow to the root's distribution π_r^{sep} such that the flow on any edge e is at most $3/2$ if e is a cut edge of G and 1 otherwise. From Lemma 3.10, we conclude that all the unmated pairs can be routed without violating edge capacities.

In the weighted case, Lemma 3.11 can be generalized to ensure that $w(Z) = \Omega(w(X)/\alpha(G))$. The above outline can be easily extended to the weighted case.

The rest of the subsection is devoted to the proof of Lemma 3.11.

Initial Clustering: We first describe an algorithm to find the clusters H_i. We start with a basic clustering as given by Lemma 3.3 with $W = 1$ and $\rho(v) = 1/(2\alpha(G))$ for terminals in X. Let $H_1, \ldots, H_p$ and $S_1, \ldots, S_p$ be as guaranteed by Lemma 3.3. We have that $p = \Omega(|X|/\alpha(G))$. Instead of using a tree for H_i as before, we define our *clusters* H_i to be the subgraph induced by $V(H_i)$. These larger clusters remain edge-disjoint since the proof of Lemma 3.3, actually guarantees that for $i \neq j$, H_i and H_j have at most one node in common.

Lemma 3.9 implies that in each cluster H_i, we can choose a center node, denoted by $c_i = c(H_i)$. Note that there may be many choices for a cluster center. For a terminal s, we denote by $H(s)$, the unique cluster H_i such that $s \in S_i$.

Phase 1 Demands: The terminal pairs Z guaranteed by Lemma 3.11 are obtained in two phases, starting with the clusters $H_1, H_2, \ldots, H_p$. Let $X' = X$. We create an initial set Z greedily as follows. We order pairs in non-increasing weight and consider them one by one. If st is the current pair, we add st to Z and remove from X' all pairs $s't'$ if either s' or t' is in $H(s) \cup H(t)$. We repeat this until X' is empty. This procedure is very similar to the one in Section 3.3. Using a counting argument similar to that of Lemma 3.5 we claim that $|Z| = \Omega(p)$. By construction, every cluster H contains either 0, 1, or 2, terminals from Z and if it contains 2, then these are the endpoints of a pair from Z. We could assume, as before, that no cluster contains two terminals from Z but as it is unnecessary we dispense with this assumption.

At this point we could proceed as outlined above, if each of the terminals in Z is a center of its cluster. However this need not be the case and for this we need to refine Z and also allow the merging of multiple clusters. The following characterization is useful for working with nodes that cannot be cluster centers.

Lemma 3.12 *Let H be a connected graph with $\rho(V(H)) \geq 1$. For any $v \in V(H)$, if v is not a center, then there exists a cut edge e in H such that every center of H lies in a distinct component from v in $H - e$.*

Proof: Let H' be a graph obtained by adding a new node t to H and an edge of capacity $\rho(w)$ from t to each $w \in V(H)$, and setting the capacity of each edge in H to $1/2$. If v is not a center, then there exists a $v - t$ cut $\delta_{H'}(S)$ of capacity less than 1 in H', where v is chosen to lie in S. Since $\rho(H) \geq 1$, if $S = V(H)$, then this cut has capacity at least 1. Hence $V(H) - S$ is nonempty. But then $|\delta_H(S)| = 1$ for otherwise the cut has capacity at least 1 in H'. By definition of a center node, any center cannot lie in S, hence the lemma. ■

The preceding lemma provides a structure on when a node s is a terminal in Z but not a center for $H(s)$. It guarantees that there is a cut edge e such that in $H(s) - e$, s is in a component F that does not contain $c(H(s))$. Let $e_1, e_2, \ldots, e_\ell$ be cut edges in $H(s)$ that separate s from $c(H(s))$ such that if F_i is the component containing s in $H(s) - e_i$ then $F_i \subset F_{i+1}$ for $1 \leq i < \ell$. Call an edge $e' = xy$ *external* if $x \in H(s)$ and $y \notin H(s)$. Suppose F_ℓ has an external edge incident to one of its nodes. Then let j be the smallest index such that there is an external edge incident to F_j and let $e(s) = xy$ be this edge and let H' be the cluster that contains y. We refer to H' as being *tagged* by s (or by $H(s)$) and denote it by $R(s)$. We observe that $e_1, \ldots, e_{j-1}$ are actually cut edges in G for otherwise it would contradict the minimality of j.

We need a method for turning terminals into centers in larger, merged clusters; this is captured in the following lemma.

Lemma 3.13 (Tagging Lemma) *Let s be a terminal that is not a center in $H(s)$. Suppose $e(s) = xy$ exists and P is a path from x starting with $e(s)$ to a cluster $H(s')$ such that P is edge-disjoint from $H(s)$. Then, s is a pseudo-center in $H(s) \cup P \cup H(s')$. If $e(s)$ does not exist, then s is a pseudo-center in $H(s)$.*

Proof: We first consider the case that $e(s)$ exists. Let c and c' be centers of $H(s)$ and $H(s')$ respectively. Let $G' = H(s) \cup P \cup H(s')$. To prove that s is a pseudo-center in G', we add a new node t to G' and connect it to each node a in G' with an edge of capacity $\rho(a)$. We make the capacity of each edge in G' equal to $1/2$ except the cut edges $e_1, e_2, \ldots, e_{j-1}$ to which we assign a capacity 1. Consider a minimum s-t cut S in G' that contains s. By Lemma 3.12, neither c nor c' are in S since they are centers in G'. If one of $e_1, \ldots, e_{j-1}$ are in $\delta_{G'}(S)$, then this cut has capacity at least 1, so suppose this is not the case. Note that there are two paths in G' from s to c and c' such that each edge, other than $e_1, e_2, \ldots, e_{j-1}$, is used in at most one of the paths. Since $\delta_{G'}(S)$ contains no cut edges, it must contain at least 2 edges, and hence the cut has capacity at least 1.

In the case that $e(s)$ does not exist, we have that $e_1, \ldots, e_\ell$ are cut edges in G. An argument similar to the above works by considering the auxiliary graph obtained by adding t to nodes in $H(s)$. ■

We could apply a greedy procedure if each cluster is tagged only by a few terminals. However a cluster can be tagged by many terminals in Z. We thus capture the tagging structure by creating a digraph D whose nodes are the clusters. We put an arc in D from H to H' if H tags H' and H does not contain both terminals of a pair in Z. If H does not tag any cluster and contains a terminal from Z, we simply put a loop arc from H to itself. Note that a cluster node has out-degree 1 if and only if it contains a single terminal from Z and 0 otherwise. Let D' be the digraph obtained from D by identifying to a single node the two clusters associated with each pair $st \in Z$, and then eliminating any loops. One sees that this results in $|Z|$ new nodes and each of these has out-degree at most 2. We call the nodes that represent the pairs in Z as *special*. The remaining nodes, one each for cluster that does not have terminals in Z, have out-degree 0.

We create a stable set I of size $|Z|/5$ among special nodes as follows. The total out-degree of nodes in D' is at most $2|Z|$. Thus there must be a special node x with in-degree at most 2. We add x to I and

remove x and nodes that have arcs to and from x from D'. Repeating this process we get the desired set I. In the weighted case one needs more care. A similar argument as above can be used to show that D' can be colored with 5 colors where each color class is a stable set. We simply pick I to be the color class with the largest weight where the weight of a node in D' is equal to the weight of the corresponding pair in Z.

We trim down Z to the demand pairs corresponding to the special nodes in I. We now consider the clusters in D corresponding to the terminals in Z. Consider a pair $st \in Z$. There are three types of pairs in Z: (i) both s and t belong to the same cluster, (ii) s and t belong to different clusters and $H(s)$ tags $H(t)$ or vice-versa (iii) s and t belong to different clusters and neither of the clusters tags the other. Let Z_1, Z_2 and Z_3 be the pairs in Z according to the above classification. For a pair st in Z_1, we do not need to do anything since $H(s)$ contains both s and t. For a pair $st \in Z_2$ we amalgamate $H(s)$ and $H(t)$ into a single connected cluster since they have an edge between them. If $Z_1 \cup Z_2$ contained at least half the weight of the pairs from Z, then we could simply connect these pairs by disjoint paths in their corresponding clusters. Hence we may focus on Z_3 from now on.

We call a cluster a *transit* cluster if it is tagged by some terminal in Z_3. From the construction of I, there is no transit cluster amongst terminals in Z. The basic approach is captured by the Tagging Lemma. If clusters H, H' both tag a cluster C, then by connecting H and H' by a path P through C, we may have any node in H or H' act as a pseudo-center in $H \cup H'$. We show that we may pair of all clusters tagging C by edge-disjoint paths through C and hence by merging clusters we get a large number of clusters with pseudo-centers. The following simple lemma shows how to match up the tagging clusters.

Lemma 3.14 *Let T be a tree and A be some even multiset of nodes in $V(T)$. Then there exists $|A|/2$ edge-disjoint paths in T such that each node $v \in A$ is the endpoint of exactly n_v of these paths, where v occurs n_v times in A.*

Proof: If $n_v > 1$, then we may take as one of our paths the singleton path v, and remove two copies of v from A. So we assume that $n_v = 0, 1$ for each node of T. Let S be the set of nodes with $n_v = 1$. Consider rooting T at an arbitrary node r. For each node v, let T_v be the subtree of T rooted at v. Let T_x be minimal such that $S \cap V(T_x) \geq 2$. Let $x_1, \ldots, x_h$ be the children of x. By minimality of T_x, each T_{x_i} has at most one node in S. We can assume without loss of generality that each T_{x_i} has a node v_i in S, otherwise we can remove T_{x_i} from T. If $x \in S$, then we connect v_1 to x by a path and remove T_{x_1} from T. Otherwise we match up v_1 to v_2 by a path and remove T_{x_1} and T_{x_2} from T. We then repeat this process. It is easy to check that this inductively matches up the pairs in S by edge-disjoint paths. ■

For a transit cluster C let $d(C)$ denote the number of clusters that tag C. Without loss of generality let $H_1, \ldots, H_\ell$ be the clusters that tag C. Let $e_i = u_i v_i$ denote the edge responsible for H_i tagging C, with $u_i \in H_i$ and $v_i \in C$. Note that the v_i need not be distinct. For $v \in C$ let n_v be the number of e_i that are incident to v. We make use of Lemma 3.14, by considering a spanning tree of C, to connect the clusters that tag C. If $d(C) = 1$ we do not do anything. Otherwise we connect $2\lfloor d(C)/2 \rfloor$ clusters that tag C by edge-disjoint paths. Note that a path P that connects clusters $H(s)$ and $H(s')$ has $e(s)$ and $e(s')$ as its first and last edges respectively. We call this the *pairing procedure* on C. See Fig 1. Based on this procedure we define a graph B on the terminals in Z_3 as follows. We put a self-loop on a node s either if s is a center in $H(s)$ or if the cluster C that $H(s)$ tags has $d(C) = 1$. We add an edge between s and s' in B if $R(s) = R(s') = C$ and are connected by a path from the pairing procedure on C.

If a node s in B does not have an edge incident to it, then $d(R(s))$ is an odd number greater than 1 and $H(s)$ was left unmatched in the pairing procedure on $R(s)$. Let Y be the pairs st in Z_3 such that neither s nor t is isolated in the graph B; these are the pairs that have a chance to have their clusters expanded appropriately. We claim that $|Y| \geq |Z_3|/3$. Let a be the number of isolated terminals. Since

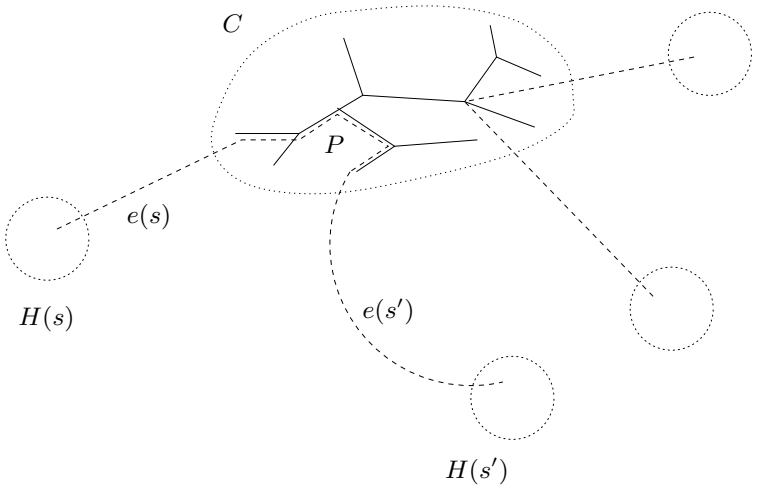

Figure 1: Pairing using C.

each isolated terminal is the unique unmatched terminal in a transit cluster of degree at least 3, the total number of terminals is at least $3a$, and the number of pairs with an isolated terminal is at most a. Hence $|Y| \geq |Z_3| - a \geq |Z_3|/3$ as claimed. In the weighted case we need to ensure that the left behind pairs are not of heavy weight. We can ensure this as follows. Lemma 3.14 guarantees that any even multiset of nodes in $V(T)$ can be matched up. Thus in the pairing procedure for C we ensure that the unpaired terminal is the one with the lowest weight among all terminals tagging C. The rest of the details are simple yet tedious and we omit them.

¿From Y we obtain a subset Y' such that $|Y'| \geq |Y|/3$ pairs with the following property: if $st \in Y'$, then either s has a self loop in B or ss' is an edge in B and s' is not an endpoint of a pair in Y'. We obtain Y' by greedily picking pairs from Y in non-increasing weight order. Each pair that is picked can result in the removal of at most 2 pairs from Y. For a terminal s from a pair st in Y' we do the following. If s is a pseudo-center in $H(s)$ we do nothing. If $R(s)$ has degree one, then we amalgamate $H(s)$ and $R(s)$ into a single cluster via the edge e that $H(s)$ uses to tag $R(s)$. If $H(s)$ is connected to $H(s')$ via a path P through the transit cluster $R(s)$, then we amalgamate $H(s)$ and $H(s')$ using the path P. By Lemma 3.13, s is a pseudo-center in the amalgamated cluster.

The pairs that remain in Y' are in clusters that satisfy Lemma 3.11. Thus either $|Y'| = \Omega(|Z|)$ or $|Z_1 \cup Z_2| = \Omega(|Z|)$, and in either case we obtain the conditions asserted in Lemma 3.11.

3.5 Multicommodity Demand Flows

When pairs in X have a demand associated with them, we can translate our result for the unit demand case to AN-DMCF at the expense of an additional constant factor in the approximation ratio. This works under the no-bottleneck assumption that $d_{\max} \leq u_{\min}$ which is commonly made in this context. Alternatively, one needs to allow an additive $d_{\max}$ congestion. The translation is obtained via a grouping and scaling technique of [31] that works for a large class of column-restricted packing integer programs (CPIPs); in [18] the relevant theorems are stated in a transparent and directly applicable form, in particular we refer to Theorem 3.1. This gives us Theorem 1.2.

4 The Online Algorithm

We now describe a randomized online algorithm for the AN-MCF problem. The reader would be served well if the earlier offline clustering arguments are fresh in mind. We focus here on the unit demand case, and sketch the extension to routing arbitrary demands at the end of the section. Recall that we seek to maximize the throughput, which in the unit-demand case, is equivalent to the number of routed pairs. For any $\epsilon > 0$, our algorithm achieves a competitive ratio of $\Omega(((h(T))^3\alpha(G))/\epsilon)$ in expectation, provided we allow a congestion of $(2+\epsilon)$ on the edges. (Recall that $h(T), \alpha(G)$ denote the height and congestion parameters associated with the Räcke tree for G.) For the unit demand case, we in fact achieve a slightly stronger guarantee on the congestion, namely, the total flow on each edge is bounded by $2 + \epsilon u(e)$.

As before, our starting point is a Räcke tree T for the input graph G, which we can assume is a connected graph. We compare the performance of the online algorithm against the maximum possible number of demands that can be routed in the Räcke tree T. The latter is clearly an upper bound on the optimal solution value. We borrow the essential ideas from the offline algorithm, however, the clustering scheme requires non-trivial modifications since we do not have an apriori LP solution to work with.

We assume that all edges in the Räcke tree are directed towards the root, and routing a pair (s,t) is thus equivalent to routing both s and t to the root in accordance with the Räcke routing. We also assume, for technical reasons that will be clear soon, that each demand pair (s,t) is an *ordered pair* where s is the *source* node and t is the *destination* node; given an unordered pair st we can produce a consistent ordered pair by using an ordering of the nodes of G and using the lower numbered node in s,t as the source. For any node x in T, let $T(x)$ denote the subtree of T rooted at x, and let $G(x)$ denote the connected subgraph of G that corresponds to $T(x)$. For a leaf node z and an ancestor x in T, we let $P[z \rightsquigarrow x]$ denote the path from z to x in T.

4.1 The Pre-processing Phase

We start with a pre-processing phase that allows the online algorithm to consider only demand pairs (s,t) for which $\text{lca}(s,t)$ is the root of the tree. We lose a factor of $h(T)$ in the (expected) competitive ratio in order to ensure this property. We also modify the capacities in the Räcke tree to satisfy some useful properties; this alteration does not change value of an optimal solution.

As a first pre-processing step, the algorithm guesses uniformly at random a level $\ell^* \in \{1, \ldots, h(T)\}$. The algorithm considers routing a request pair (s,t) only if $\text{lca}(s,t)$ is a node at level ℓ^* in the tree. Thus the Räcke tree is implicitly partitioned into disjoint trees $T_1, T_2, \ldots, T_q$ with roots $r_1, r_2, \ldots, r_q$, and we only consider a pairs (s,t) if $\text{lca}(s,t) \in \{r_1, r_2, \ldots, r_q\}$. This process loses at most a factor of $h(T)$ in expectation. Without loss of generality, from here on, we focus our attention on a single Räcke tree $T = T_i$ with root node $r = r_i$, and an underlying connected graph $G = G_i$. We abuse notation and assume that $G = G_i$ and $T = T_i$.

The second pre-processing step reduces capacity of some tree edges so as to ensure the following two properties:

(P1) Along any path from a leaf node to the root, the edge capacities are non-decreasing.

(P2) For each edge $e = (x,y)$ of the oriented T, the capacity $u_T(e)$ is at most the *maximum* amount of flow in T that can be routed to x from the leaves in the subtree of $T(x)$.

In doing this capacity reduction, we preserve the "routing capacity" of the tree (recall that at this point, we are only considering demands routed via the root) by proceeding as follows. We say that an edge

$e = (x, y)$ is a *violating edge* if either there exists an edge $e' = (y, z)$ such that $u_T(e') < u_T(e)$ (property **(P1)** is violated), or the capacity of $u_T(e)$ is greater than the maximum flow that can be routed from leaves of $T(x)$ to x (property **(P2)** is violated). As long as there is a violation of either **(P1)** or **(P2)**, we find the deepest violating edge and reduce its capacity by the least amount needed to fix the violation.

The following proposition is easy to see.

Proposition 4.1 *The tree T' obtained by the capacity reduction steps satisfies properties **(P1)** and **(P2)**. Morever, if S is a set of request pairs such that the root is the least common ancestor of each pair in the set, then S is routable in T if and only if it is routable in T'.*

4.2 The Algorithm

Recall that all edges in T are directed towards the root. For a leaf node z, we say that a node x is the *last bottleneck* node in T if and only if the capacity of every edge on the path $P[z \rightsquigarrow x]$ from z to node x is smaller than $\alpha(G)/\epsilon$, and the capacity of every edge on the path $P[x \rightsquigarrow r]$ is at least $\alpha(G)/\epsilon$. By property **(P1)**, such a node x must exist. Note that the last bottleneck node x for a leaf node z may be the node z itself or the root r. If $x = z$, then every edge in $P[z \rightsquigarrow r]$ has capacity at least $\alpha(G)/\epsilon$, and if $x = r$, then every edge in $P[z \rightsquigarrow r]$ has capacity less than $\alpha(G)/\epsilon$.

Overview of the Algorithm: The online algorithm chooses a pair of integers $\ell_1, \ell_2 \in \{1, \ldots, h(T)\}$ uniformly at random, and routes only demands (s, t) such that the last bottleneck node for s is a node at level ℓ_1, and the last bottleneck node for t is a node at level ℓ_2. This worsens the expected competitive ratio achieved by the online algorithm by a factor of $(h(T))^2$. Prior to the arrival of any demands, the algorithm identifies once and for all, a collection of *edge-disjoint connected* subgraphs of G, called *clusters*, such that each leaf node z has a unique cluster C_z assigned to it. In the following, we use some clustering arguments similar to those used in Section 3.3. The cluster of z always contains z as a node. Note that multiple leaf nodes may map to the same cluster. If the algorithm accepts a demand pair (s, t) for routing, it first distributes a unit of flow from s to a cluster C_s in $G(x)$, where x is the last bottleneck node for s. We ensure that C_s has "routing capacity" of at least $\alpha(G)/\epsilon$ to the root. Similarly, it distributes a unit of flow from t to a cluster C_t in $G(y)$ whose "routing capacity" is at least $\alpha(G)/\epsilon$; here y is the last bottleneck node for t. It then routes a unit of flow from C_s to r, and from C_t to r, using the Räcke distribution. The cluster to cluster routing induces a congestion of ϵ on the edges of G. The algorithm ensures that (i) no cluster gets used more than once, (ii) no edge appears in more than 2 clusters, and (iii) the cluster-to-cluster routing induces an overall congestion of ϵ. Thus the overall congestion of the resulting routing solution is bounded by $(2 + \epsilon)$. It now remains to describe the clustering scheme.

The Clustering Scheme: Let Γ_1 denote the set of all nodes at level ℓ_1, and let Γ_2 denote the set of all nodes at level ℓ_2 in the Räcke tree T. Note that for any pair of distinct nodes $x, x' \in \Gamma_1$, the graphs $G(x)$ and $G(x')$ are node-disjoint connected subgraphs of G. Similarly for any pair of distinct nodes $y, y' \in \Gamma_2$, the graphs $G(y)$ and $G(y')$ are node-disjoint connected subgraphs of G. For each node $x \in \Gamma_1$, we do an edge-disjoint clustering using only edges in $G(x)$, and the resulting clusters are only used for routing demands originating at a leaf node in $L(x)$; moreover the leaf should have x as its last bottlneck node. Similarly, we perform a clustering in $G(y)$ for each $y \in \Gamma_2$; hence overall, each edge is in at most 2 clusters.

Fix a node x in T. We describe the clustering scheme for the leaves $L(x)$ in the graph $G(x)$. If x is a leaf then x is in its own cluster and there is nothing further to do. Otherwise, consider the capacitated tree $T(x)$ and recall that it satisfies properties **(P1)** and **(P2)**. We define a new capacity function u'_T for edges

in $T(x)$ as follows: $u'_T(e) = \min\{u_T(e), \alpha(G)/\epsilon\}$. Let $F(x)$ denote $\sum_{i=1}^{p} u'_T(x_i, x)$ where $x_1, x_2, ..., x_p$ denote the children of x in the tree $T(x)$. We now compute the maximum amount of flow that the leaf nodes in $T(x)$ can send to x *simultaneously*, using the edge capacity function u'_T. By properties **(P1)** and **(P2)**, the maximum flow that can arrive at each x_i with respect to the capacity function u_T is exactly the capacity $u_T(x_i, x)$. Since $u_T(x_i, x) \geq u'_T(x_i, x)$, the maximum flow that can arrive at x with respect to capacity function u'_T must equal $F(x) = \sum_{i=1}^{p} u'_T(x_i, x)$. Fix a flow solution of value $F(x)$, and for each leaf node z in $T(x)$, let $f(z)$ denote the amount of flow that z sends to x in this flow solution; note that $f(z) \leq \alpha(G)/\epsilon$ for all z and $\sum_{z \in L(x)} f(z) = F(x)$. We can assume that $F(x) \geq \alpha(G)/\epsilon$ for the following reason. First, if x is the root and $F(x) < \alpha(G)/\epsilon$ then it implies that at most $\alpha(G)/\epsilon$ pairs can be routed via the root in which case the algorithm that accepts the first pair is $\alpha(G)/\epsilon$-competitive. We ignore this trivial case. Second, suppose x is an internal node of T and let x' be the parent of u. If $F(x) = u_T(x, x') < \alpha(G)/\epsilon$ then no leaf in $L(x)$ has x as its last bottlneck node and hence we do not need to cluster $G(x)$.

At a high-level, our goal now is to cluster leaf nodes in $G(x)$ into clusters of weight $\Theta(\alpha(G)/\epsilon)$ such that each cluster satisfies an additional property, namely, at most $\Theta(\alpha(G)/\epsilon)$ demands can originate from any cluster in any feasible solution. In order to do this clustering, we consider the graph H on p nodes defined as follows. The graph H has a node $v(x_i)$ for each $G(x_i)$, and there is an edge between two nodes $v(x_i)$ and $v(x_j)$ if and only if there is an edge between a node in $G(x_i)$ and a node in $G(x_j)$. The node $v(x_i)$ inherits the total f-weight of leaves in $G(x_i)$. We refer to the nodes $v(x_1), ..., v(x_p)$ as the *hub* nodes. Let $\mathcal{T}$ be an arbitrary rooted spanning tree of H. If $\mathcal{T}$ contains a node such that the f-weight of the nodes in the subtree of the node is more than $\alpha(G)/\epsilon$, we find a deepest such node η, remove the sub-tree rooted at η from $\mathcal{T}$, and label η as a *heavy* hub node. We repeat this process on the remaining tree; let $\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_k$ denote the subtrees removed in this manner. Let $\mathcal{T}'_0$ denote the remainder of tree $\mathcal{T}$ at the end of this process. Any node in H not labeled as heavy in this process, is called a *light* hub node. Observe that only the roots of the sub-trees procduced are heavy. Note that this clustering scheme is essentially the same as the scheme in Lemma 3.3; the main difference is that H is a virtual graph and hence edge-disjoint clusters in H may not translate into edge-disjoint clusters in G; in particular a heavy hub node $v(x_i)$ may participate in multiple clusters in H and we need to ensure that edges in $G(x_i)$ do not participate in multiple clusters in the eventual clusters of G that we produces. The clustering process is hence more involved and details are given below.

For $1 \leq i \leq k$, we now create an expanded tree $\mathcal{T}'_i$ from $\mathcal{T}_i$ as follows. We replace the root node $v(x_i)$ by an arbitrary rooted spanning tree of the graph $G(x_i)$, and any edge $(v(x_\ell), v(x_i))$ in $\mathcal{T}_i$ is replaced by an edge $(v(x_\ell), v)$ where v is some node in $G(x_i)$ that is connected to some node in $G(x_\ell)$. Thus $\mathcal{T}'_i$ is a rooted tree that has the property that for each hub node $v(x_\ell) \in \mathcal{T}'_i$, the total weight of the subtree of $\mathcal{T}'_i$ rooted under $v(x_\ell)$ is at most $\alpha(G)/\epsilon$. For $0 \leq i \leq k$, we now use clustering scheme of Lemma 3.3 (again based on spanning trees) on each $\mathcal{T}'_i$ to create edge-disjoint clusters using the f-values as weights such that each cluster has an f-weight between $\alpha(G)/\epsilon$ and $2\alpha(G)/\epsilon$. Let $\mathcal{C}'_i$ denote the set of clusters for the tree $\mathcal{T}'_i$. Since any hub node in $\mathcal{T}'_0, \mathcal{T}'_1, ..., \mathcal{T}'_k$ has the property that its subtree has f-weight at most $\alpha(G)/\epsilon$, the clustering scheme ensures that each hub node appears in exactly one cluster. Given any cluster in $\mathcal{C}'_i$, we can "expand" it to a cluster in $G(x)$ by replacing any hub node $v(x_\ell)$ in the cluster by a spanning tree of $G(x_\ell)$. Also, for $1 \leq i \leq k$, we define C_i to be the single cluster defined by an arbitrarily chosen spanning tree of the graph $G(x_i)$ and let $\mathcal{C}_i = \{C_i\}$. Let $\mathcal{C}(x) = \bigcup_{j=1}^{k} \mathcal{C}_i$ and $\mathcal{C}'(x) = \bigcup_{j=0}^{k} \mathcal{C}'_i$. Both $\mathcal{C}(x)$ and $\mathcal{C}'(x)$ satisfy the property that clusters contained in them are pairwise edge-disjoint. Moreover, for each of the nodes $x_1, x_2, ..., x_p$, if the node x_i was labeled heavy, then all leaf nodes in $L(x_i)$ belong to the single cluster C_i, and otherwise, all leaf nodes in $L(x_i)$ belong to a single cluster in $\mathcal{C}'(x)$.

Remark: The clustering scheme can be somewhat simplified if we settle for a congestion of $4 + \epsilon$ instead of $2 + \epsilon$. We could allow an edge to be in up to two clusters and have a single set of clusters for each x instead

of keeping track of $\mathcal{C}'(x)$ and $\mathcal{C}(x)$ and accounting for them separately.

The lemma below summarizes a key property of the clustering scheme defined above.

Lemma 4.2 *Let C be any cluster in $\mathcal{C}(x) \cup \mathcal{C}'(x)$. Let $S = \{(s'_1, t'_1), \ldots, (s'_j, t'_j)\}$ be a set of requests routable in T such that for $1 \leq i \leq j$, (i) lca$(s'_i, t'_i) = r$, and (ii) the last bottleneck node for s'_i is x. Then the number requests in S with a source in C is at most $3\alpha(G)/\epsilon$.*

Proof: Let $L'(x)$ be the subset of $L(x)$ such that $z \in L'(x)$ iff x is the last bottleneck node for z. Note that $\{s'_1, \ldots, s'_j\} \subseteq L'(x)$. Any $z \in L'(x)$ has an edge of capacity less than $\alpha(G)/\epsilon$ on the path $P[z \rightsquigarrow x]$. Since S is routable in T, for each pair (s'_i, t'_i) the path from s'_i to r goes via x. In computing $F(x)$ we truncate edge-capacities to $\alpha(G)/\epsilon$; this does not affect the routability of the pairs in S.

Now first consider the case that $C \in \mathcal{C}(x)$. Then C corresponds to some heavy hub node, say $v(x_i)$, and contains only the leaf nodes in $L(x_i)$. Since $u_{T'}(x_i, x) \leq \alpha(G)/\epsilon$ and S is routable in T, it follows that at most $\alpha(G)/\epsilon$ requests in S can originate in C. Now consider a cluster $C \in \mathcal{C}'(x)$. Suppose C was created from the tree $\mathcal{T}'_i$ for some $0 \leq i \leq k$. Then for any light hub node x_j, the cluster C either contains all leaves in $L(x_j)$ or no leaves from $L(x_j)$, along with a subset of leaf nodes in the set $L(x_i)$ (recall that $v(x_i)$ is the heavy hub node identified with the tree $\mathcal{T}'_i$). Let $x_{j_1}, ..., x_{j_q}$ be the light hub nodes whose leaves appear in C. Then by our clustering process, $\sum_{j=1}^{q} u_{T'}(x_{i_j}, x_i) \leq 2(\alpha(G)/\epsilon)$, and thus the number of demands in S that originate from C can not exceed

$$\sum_{\ell=1}^{q} u_{T'}(x_{j_\ell}, x) + u_{T'}(x_i, x) \leq \frac{2\alpha(G)}{\epsilon} + \frac{\alpha(G)}{\epsilon} = \frac{3\alpha(G)}{\epsilon}$$

as claimed in the lemma. ■

Overall Algorithm: Recall that in pre-processing we have already restricted to demands with least common ancestor at some fixed level ℓ^* in the Räcke tree. We summarize below the online algorithm for routing demands whose ancestor is a particular node r. At the outset we fix two integers $\ell_1, \ell_2 \in \{1, 2, \ldots, h(T)\}$ which are chosen independently at random. For each node in $x \in \Gamma_1$ (the nodes at level ℓ_1) we compute the set of clusters $\mathcal{C}(x)$ and $\mathcal{C}'(x)$. We choose uniformly at random between the clustering $\mathcal{C}(x)$ and $\mathcal{C}'(x)$. If $\mathcal{C}(x)$ is chosen, each leaf node v such that v appears in the subtree of a heavy hub node is assigned the unique cluster in $\mathcal{C}(x)$ in which it appears, and all other leaf nodes do not get a cluster assignment. If $\mathcal{C}'(x)$ is chosen, then each leaf node v such that v appears in the subtree of a light hub node is assigned the unique cluster in $\mathcal{C}'(x)$ in which it appears, and all other leaf nodes do not get a cluster assignment. Thus each leaf node in $G(x)$ gets assigned a cluster with probability $1/2$. We do the same random cluster assignment for each node $y \in \Gamma_2$.

(a) Consider an arriving request (s, t). Reject it outright if either of the following conditions holds:

- lca(s, t) is not r, or
- the levels of last bottleneck nodes of s and t are not ℓ_1, ℓ_2.

(b) Let x, y be the last bottleneck nodes of s, t respectively; x is at level ℓ_1 and y is at level ℓ_2. We accept the request (s, t) if all three conditions below are satisfied:

(b1) s was assigned a Γ_1-cluster and t was assigned a Γ_2-cluster at the beginning of the algorithm,

(b2) s belongs to an "unused" Γ_1-cluster and t belongs to an "unused" Γ_2-cluster, and

(b3) there is a path in T of residual capacity at least $\alpha(G)/\epsilon$ from each of x and y to r.

(c) If the pair (s,t) is accepted, we proceed as follows. The node s distributes 1 unit of flow to nodes in its cluster in accordance with the weight function f such that a node z with weight $f(z)$ received a flow of $f(z) \cdot \epsilon/\alpha(G)$. Each of the nodes in the cluster in turn sends its flow to the root according to the Räcke distribution at the root. If we route using a cluster, we mark it as "used". We then remove a capacity of $\alpha(G)/\epsilon$ from the edges along the path from x to r. We proceed in a similar manner for the node t, and remove a capacity of $\alpha(G)/\epsilon$ from the edges along the path from y to r.

Analysis: We first observe that the algorithm correctly routes the accepted pairs. For each routed pair (s,t), the algorithm creates a unit flow from s to the root r in T. This corresponds to s distributing one unit of flow to the nodes in G according to the Räcke distribution given by r. Similarly t distributes one unit of flow to the nodes in G according to the Räcke distribution given by r. This ensures that each accepted pair has a unit flow sent from s to t.

We now bound the congestion on the edges induced by the routing of the accepted pairs.

Lemma 4.3 *The algorithm routes the accepted pairs such that the flow on an edge e with capacity u_e is at most $2 + 2\epsilon u(e)$.*

Proof: For each demand pair (s,t) routed by the online algorithm, the routing uses a cluster for the source s, a cluster for the destination t, and a routing to distribute flow from the nodes in the cluster of s to the nodes n the cluster of t. The routing inside the clusters for source nodes creates at most a congestion of 1 on each edge in G since for any pair of nodes $x', x'' \in \Gamma_1$, the graphs $G(x')$ and $G(x'')$ are node-disjoint, the clusters for a node x are edge-disjoint and each cluster is used at most once. Similarly, the routing inside the clusters for destination nodes creates at most a congestion of 1 on each edge in G. Finally, we claim that for any edge e in G, the total capacity utilized in the Räcke routings over all pairs is at most $2\epsilon u(e)$. Consider a leaf node z and let $f'(z)$ be the total flow it receives from all end nodes of accepted pairs that it then sends to the root using the Räcke distribution. It suffices to prove that T supports a simultaneous flow of $\alpha(G)/(2\epsilon) \cdot f'(z)$ from each leaf node z to the root r; this implies that in G these routings induce a congestion of at most $2\epsilon u(e)$. To see this, focus on a node $x \in \Gamma_1$. If x is a leaf then it belongs to its own cluster each time a pair with end point x is routed, we remove $\alpha(G)/\epsilon$ capacity on the path $P[x \rightsquigarrow r]$. Now suppose x is an internal node. For each routed pair (s,t) with $s \in L(x)$, we remove a capacity of $\alpha(G)/\epsilon$ on the path $P[x \rightsquigarrow r]$. Therefore, what remains is to ensure each leaf $z \in L(x)$ can route $\alpha(G)/(2\epsilon)f'(z)$ to x. Note that z is in at most one cluster in $G(x)$ and since a cluster is used at most once, $f'(z) \leq 2\epsilon/\alpha(G) \cdot f(z)$ by the algorithm's routing (the factor of 2 is to account for the fact that x may also be in γ_2 for the destination nodes). Recall that, by definition, the leaves in $L(x)$ can simultanesouly send a flow of $f(z)$ each to x in T.

Thus the overall flow on an edge e at most $2 + 2\epsilon u(e)$. ■

We now show that the online algorithm routes in expectation a poly-logarithmic fraction of demands routed by an optimal solution. Fix an optimal solution OPT where we only consider demands (s,t) with $\text{lca}(s,t) = r$. Let $\mathcal{S}_{\text{OPT}}$ be the set of demand pairs st routed in OPT such that the levels of last bottleneck nodes of s and t are ℓ_1 and ℓ_2, respectively. Note that $\mathcal{S}_{\text{OPT}}$ is a random set, and that $\mathbf{E}[\mathcal{S}_{\text{OPT}}] \geq \text{OPT}/(h(T))^2$. Let $\mathcal{S}_{\text{ON}}$ denote the set of demands routed by the online algorithm.

Lemma 4.4 *The expected size of the set $\mathcal{S}_{\text{OPT}} \setminus \mathcal{S}_{\text{ON}}$ is bounded by $(3|\mathcal{S}_{\text{OPT}} \setminus \mathcal{S}_{\text{ON}}|)/4 + O(\alpha(G)/\epsilon)|\mathcal{S}_{\text{ON}}|$. Thus in expectation, we have $|\mathcal{S}_{\text{ON}}| = \Omega(\alpha(G)/\epsilon) \cdot |\mathcal{S}_{\text{OPT}}|$.*

Proof: Consider any demand $(s,t) \in \mathcal{S}_{\mathrm{OPT}} \setminus \mathcal{S}_{\mathrm{ON}}$. The probability that (s,t) is rejected in Step (b1) above is bounded by $1/4$, and thus the expected number of demands rejected in this manner is at most $(3|\mathcal{S}_{\mathrm{OPT}} \setminus \mathcal{S}_{\mathrm{ON}}|)/4$. We now account for the remaining demand pairs in $\mathcal{S}_{\mathrm{OPT}} \setminus \mathcal{S}_{\mathrm{ON}}$ and show that their number can be bounded by $O(\alpha(G)/\epsilon)|\mathcal{S}_{\mathrm{ON}}|$.

If (s,t) is rejected in Step (b2) above, then $\mathcal{S}_{\mathrm{ON}}$ contains a demand (s',t') such that either s, s' share a cluster of node $x \in \Gamma_1$ or t, t' share a cluster of a node $y \in \Gamma_2$. If s, s' share a cluster we charge (s,t) to the demand (s',t') routed in $\mathcal{S}_{\mathrm{ON}}$. From Lemma 4.2, the total number of requests in $\mathcal{S}_{\mathrm{OPT}}$ that can originate in the clusters of s and t is bounded by $2\alpha(G)/\epsilon$ each. Therefore we charge each demand in $\mathcal{S}_{\mathrm{ON}}$ at most $4\alpha(G)/\epsilon$ times in this manner.

We now account for demands that are rejected in Step (b3) above. We observe that if a demand routed in $\mathcal{S}_{\mathrm{OPT}}$ shares an edge in T with a demand routed in $\mathcal{S}_{\mathrm{ON}}$ along the path from x to r, then it must share all the edges on the path from x to r (since we only consider demands with least common ancestor r). Similarly, if a demand routed in $\mathcal{S}_{\mathrm{OPT}}$ shares an edge with a demand routed in $\mathcal{S}_{\mathrm{ON}}$ along the path from y to r, then it must share all the edges on the path from y to r. Since we remove $\alpha(G)/\epsilon$ capacity along the paths from x and y to r for each demand routed in $\mathcal{S}_{\mathrm{ON}}$, we can account for all demands in $\mathcal{S}_{\mathrm{OPT}}$ rejected in Step (2b) by charging at most $2\alpha(G)/\epsilon$ demands to each demand in $\mathcal{S}_{\mathrm{ON}}$.

Thus combining together accounting of demands rejected in Steps (b1), (b2), and (b3), we get $|\mathcal{S}_{\mathrm{OPT}} \setminus \mathcal{S}_{\mathrm{ON}}| \leq (3|\mathcal{S}_{\mathrm{OPT}} \setminus \mathcal{S}_{\mathrm{ON}}|)/4 + 6(\alpha(G)/\epsilon)|\mathcal{S}_{\mathrm{ON}}|$. Hence $|\mathcal{S}_{\mathrm{ON}}| = \Omega(\epsilon/\alpha(G) \cdot |\mathcal{S}_{\mathrm{OPT}}|$. ■

Since $\mathbf{E}[\mathcal{S}_{\mathrm{OPT}}] \geq \mathbf{OPT}/(h(T))^2$, we have that $\mathbf{E}[\mathcal{S}_{\mathrm{ON}}] = \Omega(\mathbf{OPT} \cdot \epsilon/(\alpha(G)(h(T))^2)$. Finally, since we lose an additional factor of $h(T)$ in the pre-processing phase, we have that get the following theorem.

Theorem 4.5 *For any $\epsilon > 0$, the online algorithm routes in expectation an $\Omega\left(\frac{\epsilon}{(h(T))^3\alpha(G)}\right)$-fraction of demands routed in an optimal solution. Moreover, the flow on each edge e in the graph G is bounded by $2 + \epsilon u(e)$ in the solution generated by the online algorithm.*

Extension to non-unit demands: Now we consider the case when request pairs may have different demands. The ideas here are standard and have been used several times in the literature and therefore we only give a sketch of the arguments. Let $d(st)$ denote the demand for pair st. The objective function is the amount of total demand routed; in other words routing a demand st gives a value $d(st)$. We assume that $d_{\max} \leq u_{\min}$. We assume by scaling that $u_{\min} = 1$ and hence $d(st) \leq 1$ for each pair. Call a demand pair st *large* if $d(st) > 1/2$, otherwise it is *small*. The algorithm randomly chooses upfront whether to route only demands that are large or only demands that are small. This affects the expected profit only by a factor of 2. If all demands are large, then we can simply use the unit-demand online algorithm by pretending that each of the large demands has a unit demand. If all demands are less than $1/2$ we can apply the same algorithm as the one described above for the unit demand problem with the following minor changes. When routing a demand on the tree we keep track of the capacity used by the demand. When using a cluster, we let multiple demands use a cluster as long as the total sum of the demands using the cluster does not exceed 1. The analysis is similar to the unit-demand problem. The reason to split into large and small demands follows standard ideas and avoids the following situation. Consider a graph consisting of a single edge of capacity 1. If a small demand of size ϵ arrives first and the algorithm accepts, then the adversary gives a demand of size 1 which cannot be routed. If the algorithm rejects the first demand, then the adversary gives no further demands. Clearly no deterministic algorithm can overcome this situation. On the other hand if all demands are guaranteed to be at most $1/2$, then it can be seen that accepting the first demand is not a problem in the throughput measure.

5 Conclusions

We obtained a poly-logarithmic approximation for the AN-MCF problem, a natural relaxation of the edge-disjoint path problem. A main technical tool is the hierarchical graph decomposition that Räcke developed for oblivious routing. In subsequent work [14] we introduce a general framework for approximating routing problems that relies on *well-linked* decompositions. The Räcke tree based algorithm in this paper can be viewed in that framework; the tree implicitly gives a well-linked decomposition (in fact the general framework is inspired by this observation). In this paper the approximation ratio we obtained for AN-MCF is $O(\alpha(G,T)h(T))$ where T is a Räcke tree for G and $\alpha(G,T)$ is the congestion bound and $h(T)$ is the height of the tree. If one is interested in the cardinality version of the problem, then an $O(\alpha(G,T))$ bound can be obtained; thus the ratios that we give in this paper can be improved by a logarithmic factor for the cardinality version. In particular it can be shown that the tree T yields an $O(\alpha(G,T))$-well-linked decomposition. Well-linked decompositions can also be obtained for node-capacitated routing problems [14] while it is known that there are no oblivious routings with poly-logarithmic congestion in the node-capacitated setting [26]. The well-linked framework and related ideas have led to several new algorithms [13, 14, 15, 16, 30, 17] for disjoint paths and AN-MCF. In some recent work Räcke [39] obtained an asymptotically optimal bound of $O(\log n)$ for oblivious routing. This new result also relies on hierarchical graph decompositions but differs in an important technical way from the previous approach [38, 27]. The desired oblivious routing is constructed from a convex combination of hierarchical decompositions (trees); each tree in the decomposition is a Räcke tree. Several algorithmic applications that rely on a single hierarchical decomposition can be modified to rely on the convex decomposition (see [39]). However, it is not apparent yet whether the convex decomposition would yield a well-linked decomposition with an $O(\log n)$ ratio; if it did, the ratio would match a lower bound (shown in the conference version of [17] but to appear in the journal version of [13]).

It is an interesting open problem to obtain improved online algorithms for AN-MCF problem. We believe that a poly-logarithmic competitive ratio can be obtained with congestion $(1+\epsilon)$. Finally, is there a deterministic algorithm that achieves a poly-logarithmic competitive ratio with constant congestion?

Acknowledgements: The second author was supported in part by an Alfred P. Sloan Research Fellowship and by an NSF Career Award CCR-0093117. The first and third authors did most of the work on this paper while at Lucent Bell Labs; they acknowledge support from a basic research grant number N00014-02-M-0125 from the Office of Naval Research to Bell Labs. The first author acknowledges recent support from NSF on grant CCF-0728782.

References

[1] M. Andrews, J. Chuzhoy, S. Khanna and L. Zhang. Hardness of the undirected edge-disjoint paths problem with congestion. *Proc. of IEEE FOCS*, 226–244, 2005.

[2] D. Applegate and E. Cohen. Making Intra-Domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Tradeoffs. *Proc. of ACM SIGCOMM*, 313–324, 2003.

[3] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *JACM*, 44(3):486-504, 1997. Preliminary version in *Proc. of ACM STOC*, 1993.

[4] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. on Computing*, 27(1):291–301, 1998.

[5] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. *Proc. of IEEE FOCS*, 32–40, 1993.

[6] Y. Azar and O. Regev. Combinatorial Algorithms for the Unsplittable Flow Problem. *Algorithmica*, 44(1): 49–66, 2006. Preliminary version in *Proc. of IPCO*, 2001.

[7] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke. Optimal oblivious routing in polynomial time. *JCSS*, 69(3):383–394, 2004. Preliminary version in *Proc. of ACM STOC*, 2003.

[8] N. Bansal, A. Blum, S. Chawla, and A. Meyerson. Online Oblivious Routing. *Proc. of SPAA*, 44–49, 2003.

[9] M. Bienkowski, M. Korzeniowski, and H. Räcke. A Practical Algorithm for Constructing Oblivious Routing Schemes. *Proc. of SPAA*, 24–33, 2003.

[10] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation Algorithms for the Unsplittable Flow Problem. *Algorithmica*, 47(1): 53–78, 2007. Preliminary version in *Proc. of APPROX*, 2002.

[11] C. Chekuri and S. Khanna. On Multidimensional Packing Problems. *SIAM J. on Computing*, 33(4): 837–851, 2004. Preliminary version in *Proc. of ACM-SIAM SODA*, 1999.

[12] C. Chekuri and S. Khanna. Edge Disjoint Paths Revisited. *ACM Trans. on Algorithms*, 3(4), 2007. Preliminary version in *Proc. of ACM-SIAM SODA*, 2003.

[13] C. Chekuri, S. Khanna, and F. B. Shepherd. Edge-Disjoint Paths in Planar Graphs. *Proc. of IEEE FOCS*, 71–80, 2004.

[14] C. Chekuri, S. Khanna, and F. B. Shepherd. Multicommodity Flow, Well-linked Terminals, and Routing Problems. *Proc. of ACM STOC*, 183–192, 2005.

[15] C. Chekuri, S. Khanna, and F. B. Shepherd. An $O(\sqrt{n})$ approximation and Integrality Gap for Disjoint Paths and Unsplittable Flow. *Theory of Computing*, Vol 2, 137–146, 2006.

[16] C. Chekuri, S. Khanna, F.B. Shepherd. A Note on Multiflows and Treewidth. *Algorithmica*, 54(3): 400–412, 2009.

[17] C. Chekuri, S. Khanna, F.B. Shepherd. Edge-disjoint paths in Planar graphs with constant congestion *SIAM J. on Computing*, 39(1): 281–301, 2009. Preliminary version in *Proc. of ACM STOC*, 2006.

[18] C. Chekuri, M. Mydlarz and F.B. Shepherd. Multicommodity Demand Flow in a Tree and Packing Integer Programs. *ACM Trans. on Algorithms*, 3(3), 2007.

[19] J. Chuzhoy, V. Guruswami, S. Khanna and K. Talwar. Hardness of Directed Routing with Congestion. *Proc. of ACM STOC*, 165–178, 2007.

[20] J. Chuzhoy and J. Naor. New inapproximability results for congestion minimization and machine scheduling. *JACM*, 53(5): 707–721, 2006. Preliminary version in *Proc. of ACM STOC*, 2004.

[21] Y. Dinitz, N. Garg and M. X. Goemans. On the single source unsplittable flow problem. *Combinatorica*, 19, 17–41, 1999. Preliminary version in *Proc. of IEEE FOCS*, 1998.

[22] S. Even, A. Itai and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. on Computing*, Vol 5, 691-703, 1976.

[23] A. Frank. Packing paths, cuts, and circuits - a survey. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, eds., *Paths, Flows and VLSI-Layout*, 49–100. Springer Verlag, Berlin, 1990.

[24] N. Garg, V. Vazirani and M. Yannakakis. Primal-Dual Approximation Algorithms for Integral Flow and Multicut in Trees. *Algorithmica*, 18(1):3-20, 1997. Preliminary version appeared in *Proc. of ICALP*, 1993.

[25] V. Guruswami, S. Khanna, R. Rajaraman, F. B. Shepherd, and M. Yannakakis. Near-Optimal Hardness Results and Approximation Algorithms for Edge-Disjoint Paths and Related Problems. *JCSS*, 67:473–496, 2003. Preliminary version in *Proc. of ACM STOC*, 1999.

[26] M. T. Hajiaghayi, R. D. Kleinberg, T. Leighton, and H. Räcke. Oblivious routing on node-capacitated and directed graphs *ACM Trans. on Algorithms*, 3(4), 2007. Preliminary version in *Proc. of ACM-SIAM SODA*, 2005.

[27] C. Harrelson, K. Hildrum, and S. Rao. A polynomial-time tree decomposition to minimize congestion. *Proc. of SPAA*, 34–43, 2003.

[28] J. M. Kleinberg. Single-Source Unsplittable Flow. *Proc. of IEEE FOCS*, 68–77, 1996.

[29] J. M. Kleinberg. Approximation algorithms for disjoint paths problems. PhD thesis, MIT, Cambridge, MA, May 1996.

[30] J. M. Kleinberg. . An Approximation Algorithm for the Disjoint Paths Problem in Even-Degree Planar Graphs. *Proc. of IEEE FOCS*, 627–636, 2005.

[31] S. G. Kolliopoulos and C. Stein. Improved approximation algorithms for unsplittable flow problems. *SIAM J. on Computing*, 31(3): 919–946, 2001. Preliminary version in *Proc. of IEEE FOCS*, 426–435, 1997.

[32] P. Kolman and S. Scheideler. Improved bounds for the unsplittable flow problem. *J. of Algorithms*, 61(1): 20–44, 2006. *Proc. of ACM-SIAM SODA*, 2002.

[33] P. Kolman. A Note on the Greedy Algorithm for the Unsplittable Flow Problem. *Information Processing Letters*, 88(3):101–105, 2003.

[34] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *JACM*, 46(6):787–832, 1999. Preliminary version appeared in *Proc. of IEEE FOCS*, 1988.

[35] T. Leighton, S. Rao, and A. Srinivasan. Multicommodity flow and circuit switching. *Proc. of the Hawaii International Conference on System Sciences (HICSS)*, pages 459-465, 1998.

[36] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. Preliminary version in *Proc. of IEEE FOCS*, 1994.

[37] B. Maggs, G. Miller, O. Parekh, R. Ravi, and S. L. M. Woo. Finding Effective Support-Tree Preconditioners. *Proc. of SPAA*, 176–185, 2005.

[38] H. Räcke. Minimizing congestion in general networks. *Proc. of IEEE FOCS*, 43–52, 2002.

[39] H. Räcke. Optimal Hierarchical Decompositions for Congestion Minimization in Networks. *Proc. of ACM STOC*, 255–264, 2008.

[40] S. Plotkin. Competitive Routing of Virtual Circuits in ATM Networks. Survey in *IEEE Journal on Selected Areas in Communications*, 13(6):1128-1136, 1995.

[41] P. Raghavan and C. D. Thompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[42] S. Rao. Small distortion and volume preserving embeddings for planar and Euclidean metrics. *Proc. of ACM SoCG*, 300–306, 1999.

[43] N. Robertson and P. D. Seymour. Outline of a disjoint paths algorithm. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, Eds., *Paths, Flows and VLSI-Layout*. Springer-Verlag, Berlin, 1990.

[44] A. Srinivasan. Improved approximations for edge-disjoint paths, unsplittable flow, and related routing problems. *Proc. of IEEE FOCS*, pp. 416–425, 1997.

[45] K. Varadarajan and G. Venkataraman. Graph Decomposition and a Greedy Algorithm for Edge-disjoint Paths. *Proc. of ACM-SIAM SODA*, 379–380, 2004.