From Switch Scheduling to Datacenter Scheduling: Matching-Coordinated Greed is Good

Rachit Agarwal
ragarwal@cs.cornell.edu
Cornell University
Ithaca, USA

Shijin Rajakrishnan
shijin@cs.cornell.edu
Cornell University
Ithaca, USA

David B. Shmoys
david.shmoys@cornell.edu
Cornell University
Ithaca, USA

ABSTRACT

Packet scheduling over a switch (interconnect) fabric is a well-studied problem in distributed computing, with known near-optimal distributed bipartite matching based protocols.

We initiate a theoretical study of distributed *flow* scheduling in datacenter networks. Building upon the observation that modern datacenter networks use Clos-like topologies similar to switch fabrics, we introduce a new *k-sparse flow-matching* (k-SFM) problem, a variant of the classical matching problem that captures the unique constraints imposed by flow scheduling in datacenter networks. In the k-SFM problem, we are given a weighted graph and an integer k. The goal is to assign a fractional flow value to each edge under the following three constraints: (1) for each edge, the assigned flow value is no greater than its input weight; (2) for each vertex, the sum of flow values assigned to edges incident to the vertex is at most the capacity of the vertex; and (3) for each vertex, at most k incident edges are assigned a non-zero flow value. The goal is to compute a feasible solution with the largest total fractional weight.

We design centralized and distributed algorithms for the k-SFM problem on bipartite graphs. For the centralized setting, we present a greedy $1/2$-approximation algorithm. For the distributed setting, we present three algorithms under the CONGEST model: a randomized $1/4$-approximation algorithm that runs in $O(k \log n)$ rounds, a randomized $\Omega(1)$-approximation algorithm that runs in $O(\log k \log n)$ rounds, and a deterministic $1/(4+\varepsilon)$-approximation algorithm that runs in $O(k \log^2 n \log(1/\varepsilon))$ rounds. The key idea in all of our algorithms is that existing approaches for computing maximum-weight matching can be used as a "coordination" mechanism, alongside local and greedy decisions on updating residual weights, to design near-optimal algorithms for the k-SFM problem.

CCS CONCEPTS

• **Theory of computation → Distributed algorithms**; *Scheduling algorithms*; • **Networks** → *Network protocol design.*

KEYWORDS

Datacenter networks, Flow scheduling protocols, Distributed Matching Algorithms

ACM Reference Format:
Rachit Agarwal, Shijin Rajakrishnan, and David B. Shmoys. 2022. From Switch Scheduling to Datacenter Scheduling: Matching-Coordinated Greed is Good. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC '22), July 25–29, 2022, Salerno, Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3519270.3538443

1 INTRODUCTION

Scheduling packets over a switch interconnect fabric is a classical problem in computer networking [2, 38]: each input port has fixed-sized packets for one or more output ports, and, in each time unit, the fabric can transfer at most one packet from each input port and at most one packet to each output port. A scheduling algorithm decides the input and output ports that will exchange data in each time unit. The goal is to maximize throughput, defined as the total number of packets transferred across the switch fabric per unit time. The problem of scheduling packets over a switch fabric is often modeled as a bipartite matching problem [2, 36, 38]—the input and output ports constitute vertex sets L and R, and each packet at an input port $\ell \in L$ destined to an output port $r \in R$ is represented as an edge $e = (\ell, r)$ in the graph. Then, in each time step, a schedule that maximizes throughput over the switch fabric is equivalent to finding the largest-size matching in the bipartite graph [2, 38]. This connection, among others, has motivated a large and active body of research on both switch scheduling [2, 37–40, 48, 49, 52] and distributed bipartite matching [4, 8, 12, 23, 26, 32, 36, 53].

A recent line of work in the computer networking community has explored a new connection between the problem of scheduling packets on switch fabrics, and the problem of scheduling flows (informally defined as a sequence of packets between the same input and output ports) on datacenter networks [1, 9, 22, 25, 41]. In particular, building upon the insight that modern datacenter networks use Clos-like topologies that are similar to switch fabrics [20, 51], these works adapt distributed bipartite matching algorithms from the switch scheduling literature to schedule flows on datacenter networks. However, the specific context of datacenter flow scheduling introduces a number of unique challenges (*e.g.*, large number of input/output ports, different flows having different sizes, etc.); to handle these challenges, existing protocols use various heuristics—*e.g.*, trimming down the matching algorithm to use only one round of matching, allowing each port to match with a small number of other ports, etc. Using these heuristics, state-of-the-art protocols for datacenter flow scheduling achieve good *empirical* performance across a wide variety of real-world workloads. While interesting, exploring the connection between packet scheduling on switch fabrics and flow scheduling on datacenter networks from a theoretical perspective has the potential to allow datacenter flow scheduling protocols to benefit from decades of foundational work on near-optimal algorithm design for switch scheduling and bipartite matching.

1.1 Our contributions

We initiate a theoretical study of flow scheduling in datacenter networks. We make three core contributions:

- We introduce a new problem—k-sparse flow-matching (k-SFM)—that generalizes the classical matching problem, while capturing the unique constraints imposed by flow scheduling in datacenter networks. We provide more details in §2, but briefly, an instance of k-SFM is a weighted bipartite graph with edge weights in $(0, 1]$, and an integer k. A feasible solution to an instance of the k-SFM problem consists of a fractional "flow" value for each edge, capacitated by the edge weight, such that every vertex is incident to at most unit total fractional weight, and at most k edges with non-zero support. The objective is to compute a feasible solution with the largest total fractional weight.
- The k-SFM problem generalizes two problems that have been studied extensively in the distributing computing community—fractional b-matchings [21, 27, 28, 53] and maximum-weight flow [13, 24, 44]. Unsurprisingly, the k-SFM problem turns out to be strongly NP-hard (§2.4). We thus focus on designing approximation algorithms. We present a centralized $1/2$-approximation algorithm for the k-SFM problem; while centralized, this algorithm and its analysis are interesting because they demonstrate that an iterative weight-refinement procedure, when combined with the folklore heuristic for maximum-weight matching (one that greedily adds edges in decreasing order of weights to the matching as long as vertex capacities are not violated), can produce a near-optimal centralized solution for the k-SFM problem.
- Our core result is the design of three distributed algorithms for the k-SFM problem under the CONGEST model:
  - a randomized $1/4$-approximation algorithm that runs in $O(k \log n)$ rounds;
  - a deterministic $1/(4 + \varepsilon)$-approximation algorithm that runs in $O(k \log^2 n \log(1/\varepsilon))$ rounds;
  - a randomized $\Omega(1)$-approximation algorithm that runs in $O(\log k \log n)$ rounds.

  The first two algorithms build upon the weight-refinement idea from the centralized algorithm, and add on a maximum-weight matching algorithm as a black-box "coordination" mechanism to help with the local, greedy, decisions made to achieve an approximate solution. Here, number of rounds have a linear dependence on k; this is because we use a maximum-weight matching algorithm to optimize for the weight constraint in every round, with an underlying greedy weight-refinement on the edges that preserves the cardinality constraints. Our third algorithms overcomes this linear dependence: we consider a dual approach by using a maximum-cardinality matching algorithm to optimize for the cardinality constraint in every round, guided by a greedy vertex weight-refinement procedure that preserves the weight constraint.

	Approx	Number of rounds	Run-time complexity
Centralized	$\frac{1}{2}$	-	$O(m\Delta \log m)$
Distributed	$\frac{1}{4}$	$O(k \log n)$	-
	$1/(4+\varepsilon)$	$O(k \log^2 \Delta \log(1/\varepsilon))$	-
	$\Omega(1)$	$O(\log k \log n)$	-

Table 1: Summary of our results for k-SFM on a graph with m edges, n nodes, and max-degree Δ.

Table 1 summarizes our results. Overall, the centralized algorithm and the offshoots for the distributed settings (which involve a careful balancing act between the two types of constraints, and lead to efficient algorithms) help us investigate the rich structure and the subtle complexities of the k-SFM problem. Our work is only the first step in bridging the gap between the scheduling mechanisms in switch fabrics and in datacenter networks; we outline a number of intriguing open problems in §6.

2 THE k-SPARSE FLOW-MATCHING PROBLEM

We begin this section by providing more context on flow scheduling in datacenter networks (§2.1). We then formally define the k-SFM problem in §2.2. We are not aware of prior work on the k-SFM problem, but describe the most closely related work in §2.3. Finally, we prove that the k-SFM problem is strongly NP-hard (§2.4).

2.1 Context for the k-SFM problem

The k-sparse flow-matching (k-SFM) generalizes the matching problem from the switch scheduling literature (*e.g.*, PIM [2] and iSLIP [38]) to capture the constraints imposed by flow scheduling on datacenter networks. Informally, similar to prior work from the computer networking community [1, 22, 41, 43], we exploit the fact that datacenter fabrics can be modeled similar to switch fabrics: a bipartite graph with end hosts as vertices, and an edge between a sender and a receiver vertex if they have data to exchange. Similar to classical distributed switch scheduling algorithms [2, 38], vertices exchange multiple rounds of "control plane" messages to compute a matching, upon which the matched vertices can exchange data.

The k-SFM captures three additional constraints. The first constraint relates to the large input-to-output port latency difference between switch fabrics (of the order of picoseconds) and datacenter networks (of the order of microseconds). Small latency in switch fabrics allows scheduling at per-packet granularity—multiple rounds of the matching algorithm require picoseconds, which is much smaller than the time taken to transmit a single packet (equal to the ratio of packet size to port bandwidth, usually of the order of hundreds of nanoseconds) after a matching has been computed. Tens of microseconds of latency in datacenter networks means that the overhead of multiple rounds of messaging before transmitting a single packet will be significantly higher. We will capture this in our first constraint: to achieve high throughput, it is beneficial for ports to exchange multiple packets (say, p packets) after matching*. The second constraint relates to vastly different sizes across different flows†. In particular, not all input port may have p outstanding packets to send to an output port; thus, in addition to allowing multiple rounds of matching, the k-SFM problem also allows each port to match with multiple ports. However, the observed flow performance can degrade significantly if a port is simultaneously sending/receiving data from more than a certain small number of ports [10, 41, 50]. We will capture this in the second constraint: we want to allow each input port to match with a small number of output ports (*e.g.*, 4 ports, as in some of the existing protocols [41]). Finally, the third constraint captures the fact that each output port can receive a fixed total number of packets (in aggregate, from all input ports that it matches with) per unit time due to bandwidth constraints at the sender and/or receiver host.

Thus, as we describe formally below, an instance of k-SFM is a weighted bipartite graph with edge weights in $(0, 1]$, and an integer k; the edge weight represents the total number of packets to be sent between corresponding ports, normalized by p; if an input port has more than p packets to send to an output port, we divide the data into flowlets of size p. A feasible solution to an instance of the k-SFM problem consists of a fractional "flow" value for each edge, capacitated by the edge weight, such that every vertex is incident to at most unit total fractional weight, and at most k edges with non-zero support. This corresponds to the restrictions discussed above—the cardinality constraint at every vertex ensures that ports send/receive to/from only a small number of other ports, and the capacity constraint at the vertices ensures that the number of packets a port sends/receives upon each matching is fixed. The objective now is to compute a feasible solution with the largest total fractional weight.

2.2 Formally defining the k-SFM problem

We consider the input bipartite graph $G = (L \cup R, E)$ where each edge $e \in E$ has a positive capacity $w_e \in (0, 1]$ associated with it, along with an integer k that bounds the degree of the support of feasible solutions. We let n denote the number of vertices in the vertex set $V = L \cup R$, m denote the number of edges $|E|$, and $\delta(v)$ denote the set of edges that are incident to the node v. Furthermore, we use $\Delta = \max_{v \in V} |\delta(v)|$ to denote the maximum degree of a node in the graph G. In the rest of the paper, we use the terms node and vertex interchangeably.

The goal is to select a fractional matching in the graph such that every node is incident to at most k edges in the support of this fractional matching. In other words, a feasible solution to the problem is a flow value $0 \leq f_e \leq w_e$ assigned to each edge $e \in E$ such that at each node v we satisfy the capacity constraint $\sum_{e \in \delta(v)} f_e \leq 1, \quad \forall v \in V$, and the cardinality constraint, which restricts the number of edges with positive support, $|\{e \in \delta(v) : f_e > 0\}| \leq k, \quad \forall v \in V$, and the objective is to find a feasible matching which maximizes $\sum_e f_e$. We shall refer to this as the *k-sparse flow matching problem* (k-SFM). We note that the constraint set is an intersection of a fractional b-matching constraint and a maximum weight flow constraint and thus generalizes both of these problems.

We let OPT_k denote the value of the optimal solution to this instance, and let $\{f_e^{OPT_k}\}_{e \in E}$ denote the optimal solution. Note that for a given graph and edge-capcities, the optimum value, as a function of k is non-decreasing in k. For brevity, we will drop the subscript k when it is clear from the context.

For a feasible solution $\{f_e\}_{e \in E}$, we will denote the total fractional weight at a node v as $b_v = \sum_{e \in \delta(v)} f_e$. Likewise, we define b_v^{OPT} for the total fractional node weights in the optimum solution.

For the distributed results in §4 and §5, we assume the standard CONGEST model, where nodes in the graph are processors that at each time step can transfer messages of size $O(\log n)$ to neighboring vertices, and can perform local computation. We also assume that the nodes have unique identifiers of $O(\log n)$ bits.

2.3 Prior Work

Maximum (weighted) matching and its cardinality-bounded variants (*e.g.*, b-matchings) are cornerstone problems in combinatorial optimization. These problems have been studied extensively, starting with the breakthrough work of Edmonds [17], who showed that the maximum weighted matchings can be computed in polynomial time for general graphs. Indeed, maximum-weight b-matchings, being generalizations of both maximum cardinality matchings and maximum-weight matchings, are considered to belong to a "well-solved class of integer linear programs" [18] in the sense that they can all be solved in (strongly) polynomial time. There are excellent surveys on matching theory by Lovász and Plummer [45], and by Pulleyblank [46].

Matching algorithms have a wide range of real-world applications, including matching children to schools [15, 31], donor organs to patients [7, 14], reviewers to manuscripts [11, 34], in protein structure alignments [29], computer vision [5], to name a few. Despite these problems admitting polynomial-time exact algorithms, there has been a flurry of activity in developing approximation algorithms since they are usually more scalable than their exact counterparts for (massive) real-world graphs. The best sequential approximation algorithm for weighted matchings is the $(1 - \varepsilon)$-approximation algorithm based on the scaling approach designed by Duan and Pettie [16] and runs in $O(\frac{m}{\varepsilon} \log \frac{1}{\varepsilon})$, where m is the number of edges in the graph.

In contrast to the centralized setting, where maximum cardinality and weighted b-matching problems belong to the class of "well-solved" combinatorial problems, designing distributed algorithms for matching problems is far more complex. For instance, even the most natural $\frac{1}{2}$-approximate centralized greedy algorithm which repeatedly adds the heaviest remaining edge to the current matching has no straightforward counterpart in the distributed setting. The classical result here is that of Israeli and Itai [26], who provide an elegant randomized distributed algorithm for computing a $\frac{1}{2}$-approximate maximum (cardinality) matching in a graph in $O(\log n)$ time. This result was the best known (from a worst-case complexity point of view) until the improvement by Lotker et al. [35] where they give a randomized $(1 - \varepsilon)$-approximation

*p can be computed such that, upon each matching, the overhead of multiple rounds of the matching algorithm is a small fraction of the time taken to transmit these packets.

†Traffic analysis studies from real-world production datacenters show that the amount of data that hosts exchange can vary over multiple orders of magnitude [6, 47].

algorithm that runs in $O(\log n/\varepsilon^3)$ rounds. For the case of weighted matchings, Lotker et al. [36] provide a randomized $(\frac{1}{4}-\varepsilon)$- approximation using $O(\log n)$ time, and explain how to get $(\frac{1}{2}-\varepsilon)$-approximate maximum weighted matching using $O(\log^2 n)$. This was improved upon in [35], where the authors provide a randomized $(\frac{1}{2}-\varepsilon)$-approximation algorithm that runs in $O(\log\frac{1}{\varepsilon}\log n)$ rounds. There have since been improvements for special cases of bounded degree graphs [19].

When considering distributed approximation algorithms, a natural tradeoff is between the approximation factor and the round complexity of the algorithm. In this context, several lower bounds are known. For instance, it is known that no single-round distributed graph matching algorithm can achieve approximation factor $\Omega(\mu^{1/3})$, where μ is the size of an optimal matching [3]; in fact, a recent result [30] establishes that any (deterministic or randomized) distributed algorithm that computes a constant approximation to maximum matching on a bipartite graph with n nodes must use $\Omega(\sqrt{\log n/\log\log n})$ rounds.

When considering the problem of computing an approximate maximum weighted b-matching, one of the first constant factor approximation algorithms was from Panconesi et al. [42], who provide a randomized $\frac{1}{6+\varepsilon}$-approximation algorithm that requires $O(\frac{\log^3 n}{\varepsilon^3}\log^2\frac{w_{max}}{w_{min}})$ rounds. Around the same time, independent work by Koufogiannakis and Young [28] provides a randomized $\frac{1}{2}$-approximation requiring O(log n) rounds for the case of maximum weighted b-matchings, which runs in expectation and with high probability, and thus meeting the best known results for the case of weighted matchings.

The results mentioned above all require randomization, and while the landscape for deterministic distributed graph algorithms is in general bleaker, it is worthwhile to consider the parallel improvements in deterministic algorithms for the matching problems we discuss. One of the first results comes from Linial [33] who provides a poly $\log n$-round deterministic algorithms for maximum cardinality matchings. When considering weights, Panconesi and Sozio [42] provide a primal-dual distributed $\frac{1}{6+\varepsilon}$-approximation algorithm for weighted matchings, which takes $O(\frac{\log^4 n}{\varepsilon}\log\frac{w_{max}}{w_{min}})$ rounds, where w_{max} and w_{min} are the maximum and minimum weight of any edge in the graph, respectively. This was recently improved by Fischer [21] using linear programming(LP) rounding methods, with their deterministic distributed algorithm for computing a $\frac{1}{2+\varepsilon}$-approximation of a maximum weighted b-matching in $O(\log^2\Delta\log\frac{1}{\varepsilon})$ rounds, where Δ is the maximum degree of a vertex in the graph.

Finally, several recent protocols from the computer networking community focus on flow scheduling in datacenter networks [1, 9, 22, 25, 41]. Most of these focus on heuristics and empirical improvements [1, 22, 25, 41]; the closest to our work is dcPIM [9]. dcPIM also explores connections between switch scheduling and datacenter scheduling, but focuses on a complementary problem where sparsity in datacenter traffic matrices (and thus, in the resulting bipartite graph) can be exploited to improve the theoretical bounds for classical switch scheduling protocols.

2.4 Strong NP-hardness

We prove that the k-SFM problem is *strongly* NP-hard through a reduction from Numerical 3-dimensional matching (N3DM), a strongly NP-hard decision problem. An instance of N3DM consists of 3 sets of positive integers, $A=\{a_1,\dots,a_n\}$, $B=\{b_1,\dots,b_n\}$, and $C=\{c_1,\dots,c_n\}$, such that

$$\sum_{i=1}^{n} a_i+b_i+c_i=nD.$$

The objective is to determine if there exist permutations $\sigma,\pi:[n]\mapsto[n]$ such that

$$a_{\sigma(i)}+b_{\pi(i)}+c_i=D,\quad \forall i\in[n].$$

Given such an instance of N3DM, we construct the following instance of the k-sparse flow matching problem, consisting of the bipartite graph $G=(L\cup R,E)$ and $k=3$.

The left partition L consists of $3n$ nodes, $L=L_1\cup L_2\cup L_3$ with $L_i=\{u_1^i,\dots,u_n^i\}, i\in\{1,2,3\}$. The right partition consists of the following $5n$ nodes, $R=A'\cup B'\cup C_1'\cup C_2'\cup C_3'$, with $A'=\{\overline{a}_1,\dots,\overline{a}_n\}$, $B'=\{\overline{b}_1,\dots,\overline{b}_n\}$, and $C_i'=\{\overline{c}_1^i,\dots,\overline{c}_n^i\}, i\in\{1,2,3\}$.

The edge set E consists of $3n+6n^2$ edges, $E=E_A\cup E_B\cup E_C$, where $E_A=\{(u,\overline{a})|\overline{a}\in A',u\in L\}$, $E_B=\{(u,\overline{b})|\overline{b}\in B',u\in L\}$, and $E_C=\{(u_j^i,\overline{c}_j^i)|i\in\{1,2,3\},j\in[n]\}$.

The edge weights are defined as follows - every edge of the form $(u,\overline{a}_i)$ has weight $\frac{2D+a_i}{15D}$, every edge of the form $(u,\overline{b}_i)$ has weight $\frac{4D+b_i}{15D}$, and every edge of the form $(u,\overline{c}_j^i)$ has weight $\frac{8D+c_j}{15D}$.

Theorem 1. *An instance $\{A,B,C\}$ of N3DM has a solution if and only if the constructed k-SFM instance has a solution of value $3n$.*

Proof. If the N3DM instance has a solution, i.e. permutations σ,π s.t.

$$\sum_{i=1}^{n} a_{\sigma(i)}+b_{\pi(i)}+c_i=D,\quad \forall i\in[n],$$

then we can construct a solution to the k-SFM instance of value $3n$ where every vertex u_j^i on the left is incident to the three edges $(u_j^i,\overline{a}_{\sigma(j)})$, $(u_j^i,\overline{b}_{\pi(j)})$, and $(u_j^i,\overline{c}_j^i)$. It is easy to note that the total weight incident to every vertex on the left is exactly 1, and that every vertex in A' and B' have degree 3 in the matching, but since the weight of each indicent edge to A' and B' is less than $\frac{1}{3}$, this leads to a feasible k-SFM solution of value $3n$.

Now suppose that the k-SFM instance has a solution of value $3n$. Note that the maximum value of any feasible solution is bounded by

$$\begin{aligned}\mathrm{OPT} &\le 3\sum_{i=1}^{n}\frac{2D+a_i}{15D}+3\sum_{i=1}^{n}\frac{4D+b_i}{15D}+3\sum_{i=1}^{n}\frac{8D+c_i}{15D}\\ &=3\left[\frac{14nD}{15D}+\frac{1}{15D}\sum_i(a_i+b_i+c_i)\right]\\ &=3n\end{aligned}$$

where the inequality uses the fact that every node in C' has degree one and every node in $A'\cup B'$ can have degree at most 3 (due to the cardinality constraint).

Thus if the k-SFM instance has a solution of value $3n$, then we can see that every edge of the form $(u_j^i, \bar{c}_j^i)$ is in the solution with its full weight, and every node in $A' \cup B'$ has exactly 3 outgoing edges, each of full weight. Since the nodes in C' are each incident to exactly one node in L, the remaining weight that every node in L can have after matching the vertices in C' is strictly smaller than $\frac{7D}{15D}$, and thus each can be matched to at most one edge with an endpoint in B'. Similarly, after each vertex is matched with a vertex in B', the remaining weight (and cardinality constraint as well) at every node in L ensures that it can only be matched to at most one node in A'. The corresponding matchings in L_1, L_2, and L_3 each leads to a feasible solution to the N3DM instance.

□

3 A CENTRALIZED APPROXIMATION ALGORITHM

Since the k-SFM problem is strongly NP-hard, we explore approximate solutions. In this section, we provide a centralized greedy $\frac{1}{2}$-approximation algorithm to the k-sparse flow-matching (k-SFM) problem. The algorithm and its analysis demonstrates the power of an adaptive greedy approach that, at each step of the algorithm, makes a myopic augmentation to the solution based on centrally coordinated information. Of course, although this finds near-optimal solutions, its design suggests a significant impediment to a distributed solution; in Section 4, we will show that matchings, which in essence allow for a maximal set of "non-interfering" such augmentations, provide a mechanism that these potential impediments can be overcome with only a slight degradation in the quality of the solution obtained.

The algorithm runs for m rounds, processing one edge in each round; the basic idea of the algorithm is to maintain, for each edge $e \in E$, a residual weight that reflects an updated upper bound on the fraction that can be feasibly assigned to e, and in each round, we consider an edge of maximum residual weight, and add that edge (fractionally) provided that it does not violate either endpoints' cardinality or weight constraints. Note that the residual edge weight of each edge is monotone nonincreasing: the residual weight of an edge not considered in earlier rounds is modified to be the smaller of its input weight and the maximum weight it can be assigned in a feasible solution that augments the current solution.

More specifically, suppose we are in round r, and the edge set considered previously is $F^{(r)}$, of cardinality $r-1$, and the current fractional solution is $\{f_e^{(r)}\}_{e \in F^{(r)}}$. We define the current total fractional weight at each node,

$$b_v^{(r)} = \sum_{e \in (\delta(v) \cap F^{(r)})} f_e^{(r)}.$$

Starting with all residual edge weights equal to their actual weights, $w_e^{(0)} = w_e$, we now consider the residual weight of an edge $e = (u,v) \notin F^{(r)}$ in round r, $w_e^{(r)} = \min\{w_e^{(r-1)}, 1 - b_u^{(r)}, 1 - b_v^{(r)}\}$.

We then choose the edge with the largest residual edge weight in this round and continue.

We now prove that the greedy algorithm described produces a feasible fractional matching of value at least half as large as that of the optimal solution.

The idea behind the proof is to account for the weight of every edge in an optimum solution with the weight of the edges in the solution returned by the algorithm. To this end, consider the edges e in the optimum solution, chosen with value f_e^{OPT}. Now if the same edge is chosen in ALG with a value at least this large, and so $f_e^{ALG} \geq f_e^{OPT}$, then we can account for these edges easily. We now know that every remaining edge in OPT has a strictly smaller weight in ALG than in OPT. When considering this edge in some iteration r, the chosen weight can be smaller either due to a weight constraint or a cardinality constraint becoming tight at one of the endpoints (or both). We will now consider both cases separately. For the former, let E_1 be the set of edges we consider. We can map each such edge $e \in E_1$ to a tight vertex v_e in ALG. We can see that if a weight constraint is tight then the corresponding vertex is already incident to a large weight, i.e., $b_{v_e}^{(r)} \geq 1 - f_e^{ALG}$. It is now easy to see that the total weight of these edges in the optimum solution is at most the total incident weight to the corresponding tight vertices, i.e.,

$$\sum_{e \in E_1} f_e^{OPT} \leq \sum_{\substack{v: \exists e \in E_1 \\ v = v_e}} b_v$$

We can thus account for the edges in E_1 by using at most 2 copies of the edges incident to the tight vertices.

Finally, consider E_2, the set of edges in OPT that were not added to ALG due to a cardinality constraint becoming tight at a vertex v_e, $e \in E_2$. Since the cardinality constraint is tight, we know that ALG has k edges incident to that vertex, and we can pair up every edge $e \in E_2$ with an edge $p(e) \in ALG$ such that $e \cap p(e) = v_e$. The easy case is if the paired edge has a weight in ALG larger than the weight to be covered (f_e^{OPT}). For the harder case, the paired edge in ALG has a smaller weight than e - since we choose edges in non-increasing order of residual weights, it is clear that at the iteration that $p(e)$ was considered, say iteration r, the residual weight of e was smaller than or equal to the residual weight of $p(e)$. This can only be because one of the vertices of e had more than $1 - w_e^{(r)}$ weight! We can thus account for the weight of the edges in E_2 using the weight of the heavily loaded endpoints for each edge. The analysis described above yields a constant factor approximation, and we define the partition of edges in OPT with more care below to avoid frivolous over-counting, yielding the $\frac{1}{2}$-approximation.

Theorem 2. *Let* ALG *be the value of the solution produced by the greedy algorithm. Then* $ALG \geq \frac{1}{2} OPT$.

Throughout this analysis, fix one optimal solution and let $F_{OPT} = \{e \in E : f_e^{OPT} > 0\}$ be the set of edges with strictly positive support in this optimal solution. The value of the optimal solution is thus

$$OPT = \sum_{e \in F_{OPT}} f_e^{OPT} = \frac{1}{2} \sum_{v \in V} b_v^{OPT}$$

Similarly, we define F_{ALG} to be the set of edges with strictly positive support in the solution returned by the greedy algorithm.

We will account for the total weight $\sum_v b_v^{OPT}$ by noting that each edge $e = (u,v) \in F_{OPT}$ contributes f_e^{OPT} twice to this summation, once at the vertex u and once at the vertex v. We will keep track of this by considering edge-vertex pairs, whereby each edge $e = (u,v)$ has two edge-vertex pairs associated with it; in this case, (e,u) (which contributes to b_u^{OPT}) and (e,v) (which contributes to b_v^{OPT}).

Let L be the list of all edge-vertex pairs corresponding to the edges in F_{OPT}. Thus we can see that

$$OPT = \frac{1}{2}\sum_{v} b_v^{OPT} = \frac{1}{2}\sum_{(e,v)\in L} f_e^{OPT}.$$

We will define four disjoint sets that will consist of edge-vertex pairs from L and help in the analysis.

(1) Let $S \triangleq \{v \in V \mid b_v^{ALG} \geq b_v^{OPT}\}$ be the set of vertices for which the total fractional weight in the solution returned by the greedy algorithm is at least the total fractional weight in the optimum solution. For each node $v \in S$, consider each edge $e = (u, v) \in \delta(v) \cap F_{OPT}$. In the optimum solution, this edge contributes f_e^{OPT} to both endpoints of e, u and v. We will use the fractional weight of the edges incident at v in the greedy solution to account for the edge-vertex pairs, (e, u) and (e, v). Add (e, v) to the set A, and remove both (e, u) and (e, v) from the list L.

(2) For each edge-vertex pair $(e, v) \in L$ such that $f_e^{ALG} \geq f_e^{OPT}$, we can account for the fractional weight f_e^{OPT} using the corresponding weight in the greedy solution. Letting u denote the other endpoint of the edge e, we add both (e, u) and (e, v) to the set B, and remove them from the list L.

Consider any edge-vertex pair (e, v) with $e = (u, v)$ that remains in L at this point. Since (1) did not remove this pair, we can conclude that both $b_u^{ALG} < b_u^{OPT} \leq 1$ and $b_v^{ALG} < b_v^{OPT} \leq 1$. Furthermore, since (2) did not delete (e, v), we can conclude that $f_e^{ALG} < f_e^{OPT} \leq w_e$. But when the edge e was processed by the greedy algorithm, why didn't we set f_e^{ALG} to be a larger value than its current value? It must be the case, therefore, that making f_e^{ALG} positive would violate the cardinality constraint at u, v, or both.
Let $S_2 \triangleq \{v \in V : |\delta(v) \cap F_{ALG}| = k\}$ be the set of vertices that have k edges incident to it with positive support in the greedy solution.
For each vertex $v \in S_2$, let $E_v^{OPT} \triangleq \delta(v) \cap F_{OPT}$ be the set of edges with positive support in the optimum solution incident to the vertex. Note that $|E_v^{OPT}| \leq k, v \in S_2$. Similarly, we define $E_v^{ALG} \triangleq \delta(v) \cap F_{ALG}, v \in S_2$ and note tha $|E_v^{ALG}| = k, v \in S_2$.
Now for each vertex $v \in S_2$, we will pair every edge $e \in E_v^{OPT}$ with an edge in E_v^{ALG}, and let $p : E \times V \mapsto E$ be the pairing function. Note that the cardinalities of the sets E_v^{ALG} and E_v^{OPT}, $v \in S_2$ ensures that the function is well defined. Of course, not all edges in F_{ALG} will be an image of some pairing.

(3) As argued above, for each edge e such that $(e, \cdot) \in L$, at least one of its endpoints is in the set S_2. Each such remaining edge-vertex pair (e, u), where $u \in S_2$, has a pair $p(e, u) \in E_u^{ALG}$. Now if $f_{p(e,u)}^{ALG} \geq f_e^{OPT}$, we can account for the fractional weight of e in the optimum solution using edges incident to u in the greedy solution.
If the other endpoint v of this edge $e = (u, v)$ does not belong to S_2, then we will account for that endpoint using the same accounting as for vertex u. So in such a case, add (e, u) to the set C, and remove both (e, u) and (e, v) from the list L. Otherwise, if $v \in S_2$, we add (e, u) to the set C' and remove (e, u) from the list L. (Note that (e, v) will also be processed and removed in this step).

Now consider a remaining edge-vertex pair $(e, v) \in L$, where $e = (u, v)$. We now claim that at least one of u and v is not in S_2. To see why, suppose instead that both endpoints u and v belong to S_2. Then both of the edges $p(e, u)$ and $p(e, v)$ exist. Now consider the iteration in the algorithm when the edge $p(e, v)$ was considered, say iteration r. If $w_e^{(r)} < w_e$, then either $\max\{b_u^{ALG}, b_v^{ALG}\} = 1$, and thus both (e, u) and (e, v) would have been deleted from L in (1). Therefore, $w_e^{(r)} = w_e$. Now since, at iteration r, the edge e was not chosen, we know that

$$w_e = w_e^{(r)} \leq w_{p(e,v)}^{(r)} \leq w_{p(e,v)},$$

and similarly

$$w_e = w_e^{(r)} \leq w_{p(e,u)}^{(r)} \leq w_{p(e,u)}.$$

However, this would imply that both (e, u) and (e, v) would have been removed from L in (3). This proves the claim.

(4) Consider a remaining edge-vertex pair $(e, v) \in L$ where $e = (u, v)$. (Note that from the way we have removed entries from L so far, the edge-vertex pair (e, u) is also necessarily in L at this stage). From the above discussion, we know that exactly one of u and v is in the set S_2. Without loss of generality, let us assume that $v \in S_2, u \notin S_2$.
Now consider the iteration r in which the edge $p(e, v)$ was considered (and subsequently chosen). Combining the facts that $f_{p(e,v)}^{ALG} = w_{p(e,v)}^{(r)}$ and that $w_e^{(r)} \leq w_{p(e,v)}^{(r)}$, we can observe that

$$1 - b_u^{(r)} \leq w_e^{(r)} \leq w_{p(e,v)}^{(r)} = f_{p(e,v)}^{ALG}. \tag{1}$$

We will need to account for two copies of f_e^{OPT}, one for each endpoint, and we will use the fractional weights at the node u to do so. We add (e, u) to the set D and remove both (e, u) and (e, v) from L. *n.b. the vertex in the edge-vertex pair added to D is the endpoint not in S_2.*

At this point, the list L is empty. The $\frac{1}{2}$-approximation now follows by noting that the definition of the various sets above ensures that for each set, the vertices in that set pay for the weight of the edges OPT using at most twice the weight of that node in ALG. The formal details of the proof are in the full version of the paper.

4 A DISTRIBUTED $\frac{1}{4}$-APPROXIMATION ALGORITHM

In this section, we extend the ideas developed in the centralized greedy algorithm to design a distributed algorithm for the problem. Recall that the centralized algorithm runs for m rounds, where in each round we select the edge with the highest residual weight. To reduce the number of rounds, we will select a larger number of edges in every round, and then modify the edge weights as before in between each round. More specifically, the algorithm runs for k rounds, where in each round we use a (known) distributed algorithm to compute a near-optimal maximum-weight matching. In between each round, we adjust the weights as in the centralized section, and this ensures that the resulting solution does not violate the capacity constraint. Furthermore, since each node is a part of at most k matchings, it is easy to see that the cardinality constraints also hold.

A precise description of our distributed algorithm is described below in Algorithm 1.

Algorithm 1: Distributed Algorithm

```
$w_e^{(0)} = w_e, \forall e \in E$;
$F = \emptyset$;        // set of edges already considered
$b_v^{(0)} = 0, \forall v \in V$;    // cumulative total fractional
 weight at each node
$f_e^{ALG} = 0, \forall e \in E$;
for $r = 1$ to $k$ do
    Compute a $\delta$-approximate maximum-weight matching,
     $M_r^{ALG}$;
    $F \leftarrow F \cup M_r^{ALG}$;
    $b_v^{(i)} \leftarrow b_v^{(r-1)} + \sum_{e \in M_r^{ALG} \cap \delta(v)} w_e^{(r-1)}$;
    $w_e^{(r)} \leftarrow \min\{w_e^{(r-1)}, 1 - b_u^{(r)}, 1 - b_v^{(r)}\}, \forall e = (u,v) \notin F$;
    $f_e^{ALG} \leftarrow w_e^{(r)}, \quad \forall e \in M_r^{ALG}$;
end
```

Now we analyze the value of our solution, $M = \bigcup M_i^{ALG}$, when compared to the value of the optimal solution, OPT .

Theorem 3. *Let ALG be the value of the solution returned by Algorithm 1 when using a δ-approximate maximum-weight matching subroutine, and OPT the value of the optimal solution for the same instance. Then $ALG \geq \left(2 + \frac{1}{\delta}\right)^{-1} \cdot OPT$.*

As in the previous section, we shall compare $\sum_v b_v^{ALG}$ to $\sum_v b_v^{OPT}$, and place the edges in OPT in different sets based on their properties.

We let $F_{OPT} = \{e \in E : f_e^{OPT} > 0\}$ be the set of edges in the optimal solution with positive support and $F_{ALG} = \{e \in E : f_e^{ALG} > 0\}$ be the set of edges in the solution returned by Algorithm 1 with positive support.

To account for the cost of the optimal solution, we initialize L with all of the edges in the optimal solution with positive support (and so $L \triangleq F_{OPT}$), so that we have $OPT = \sum_{e \in L} f_e^{OPT}$.

We will define three disjoint sets which will form a partition of the edges in L.

(1) First consider each edge $e \in L$ that is incident to a vertex $v \in V$ such that $b_v^{ALG} = 1$. We will account for this edge using the total fractional weight of edges incident to the vertex v in F_{ALG}. Add the edge e to the set A and remove it from L. Define the set $V_A \triangleq \{v : b_v^{ALG} = 1\}$.

(2) Consider the remaining edges in L. We will now split them into sets based on whether the edge is present in the solution returned by Algorithm 1.

Claim 1. *For each remaining edge $e \in L \cap F_{ALG}$, we have that $f_e^{OPT} \leq f_e^{ALG}$.*

Proof. Suppose not, and thus $f_e^{ALG} < f_e^{OPT} \leq w_e$.

It is easy to observe from the algorithm that if an edge $e = (u,v)$ with a residual weight $w_e^{(r)} < w_e$ is chosen in round r, then $\max\{b_u^{ALG}, b_v^{ALG}\} = 1$. Noting that the fractional weight of an edge chosen in round r equals its residual weight in that round, we can see that any edge $e = (u,v) \in L \cap F_{ALG}$ such that $f_e^{ALG} < w_e$ implies $e \in A$, and thus would have been removed from L in (1), which is the desired contradiction. □

Thus we see that we can account for edges in $L \cap F_{ALG}$ using the fractional weight of the same edge in our solution. We add the edges in $L \cap F_{ALG}$ to the set B and remove them from L.

(3) Consider the bipartite graph with edge set consisting of the edges remaining in L. By the feasibility of F_{OPT}, this is a bipartite graph of maximum degree at most k, and hence can be edge-colored with k colors; that is, we can partition the remaining edges in L into a set of at most k matchings, $M_1^{OPT}, \ldots, M_k^{OPT}$. For an $i \in [k]$, consider the matchings M_i^{ALG} and M_i^{OPT}. The removal of edges in (2) implies that the edges in M_i^{OPT} are not in F_{ALG}, and thus are present in round i when the matching M_i^{ALG} was selected. The idea now is to use the fact that in round i we augment the solution with a δ-approximate maximum-weight matching, which accounts for the fractional weight (in the optimal solution) of the edges in M_i^{OPT}. The important point to note, however, is that the approximate maximum-weight matching is computed with respect to the residual weights of the edges in M_i^{ALG}, and thus using this idea, we can only account for $\sum_{e \in M_i^{OPT}} w_e^{(i)}$. To account for the remaining fractional weight of $\max\{0, f_e^{OPT} - w_e^{(i)}\}$ for an edge $e \in M_i^{OPT}$, note that for an edge $e = (u,v)$ such that $w_e^{(i)} < f_e^{OPT} \leq w_e$, the fact that the residual weight of this edge is strictly smaller than its weight implies that $\max\{b_u^{(i)}, b_v^{(i)}\} = 1 - w_e^{(i)}$.
We assume, without loss of generality that $b_v^{(i)} \geq b_u^{(i)}$, and so

$$b_v^{ALG} \geq b_v^{(i)} = 1 - w_e^{(i)}. \tag{2}$$

We will pair up this edge e with the vertex v using the pairing function $p : F_{OPT} \mapsto V$ so that $p(e) = v$. Let C be the remaining edges in L, and hence $F_{OPT} = A \cup B \cup C$.
Furthermore, we define the set $V_C \triangleq \{v \in C : \exists e \text{ s.t. } p(e) = v\}$ to be the image set of the pairing function p. Note that $V_C \cap V_A = \emptyset$, from (1).

This partition of the set of edges into the three sets A, B, and C allows us to prove the theorem. The proof is similar to the one for the centralized algorithm, and involves accounting for the weight of the edges in the optimal solution with the weight of the edges in each set. We present details in the full version of the paper.

We can combine the theorem with the result from Koufogiannakis and Young [28] on a distributed $\frac{1}{2}$-approximate solution for the maximum-weight matching that runs in $O(\log n)$ rounds in expectation and with high probability, and with the result from Fisher [21] on the $\frac{1}{2+\varepsilon}$-approximate maximum-weight matching that runs in $O(\log^2 \Delta \log \frac{1}{\varepsilon})$ rounds, to get the following approximation guarantees for Algorithm 1.

Corollary 4. *There is a randomized distributed $\frac{1}{4}$-approximate solution that runs in $O(k \log n)$ rounds both in expectation and with high probability.*

Corollary 5. *There is a deterministic distributed $\frac{1}{4+\varepsilon}$-approximate solution that runs in $O(k \log^2 \Delta \log \frac{1}{\varepsilon})$ rounds.*

5 A $O(\log n \log k)$-ROUND DISTRIBUTED ALGORITHM

In this section, we present a $\Omega(1)$-approximate distributed algorithm for k-SFM that runs in $O(\log k \log n)$ rounds. Our approach is to prove that we incur only a constant-factor loss in the approximation when considering a more discrete version of the problem. In the "all-or-nothing" variant of the problem, the flow f_e on an edge e has to belong to the set $\{0, w(e)\}$. In other words, a feasible solution consists of a set of edges in the graph such that the cardinality constraints are satisfied at every node and the sum of weights of the edges in the solution incident at a node is at most 1. Thus, solving the all-or-nothing variant yields us an approximation algorithm to the k-SFM problem as well.

Our current understanding of the problem makes it easier to solve the all-or-nothing variant, primarily due to the following observation where we consider two special instances: if the weights of all the edges in the graph are at most $\frac{1}{k}$, then the sparsity constraint dominates the weight constraint at a node, and conversely, if the weights of all the edges in the graph are at least $\frac{1}{k}$, then the weight constraint dominates the cardinality constraint. Thus in these special cases, we can consider the simpler problems that deal with only one constraint. We can leverage this observation by constructing two subinstances by partitioning the set of input edges into "heavy"-weight and "light"-weight edges. Of course, for this split to be useful we need three things - distributed algorithms for each subinstance, a method to combine the solutions from the subinstances, and a bound on the approximation-factor loss in considering the all-or-nothing variant.

We will now exhibit each of these parts, to prove the following main result.

Theorem 6. *There is a randomized distributed $\Omega(1)$-approximate algorithm to the k-SFM that runs in $O(\log k \log n)$ rounds both in expectation and with high probability.*

We will begin by analyzing the loss in considering the all-or-nothing variant.

Lemma 1. *Let $G = (L \cup R, E)$, $(w_e)_{e \in E}$, and k be an input instance to the k-SFM problem. Let OPT be the value of the optimal solution to this instance, and let OPT^{aon} be the value of the optimal solution to this instance for the all-or-nothing variant. Then $\text{OPT}^{aon} \geq \frac{1}{6}\text{OPT}$.*

The proof of this lemma consists of comparing, node-by-node, the node-weights due to incident edges in the solutions OPT and OPT^{aon}. The difficult case is when a node has a small incident weight in OPT^{aon} and not in OPT. However if a node has small incident weight, then it must have k edges incident to it, preventing it from adding incident edges in $\text{OPT} \setminus \text{OPT}^{\text{aon}}$. We can then create disjoint augmenting paths with such edges and use the optimality of the solution OPT^{aon} to bound the weight of the edges in the current solution with the weight of the incident edges in $\text{OPT} \setminus \text{OPT}^{\text{aon}}$.

Proof. As before, we define $b_v^{\text{OPT}^{\text{aon}}}$ to be the total weight incident on a node v in OPT^{aon}.

Our goal is to account for every edge in OPT (some of which might be taken fractionally). Let $S_1 := \{v \in V : b_v^{\text{OPT}^{\text{aon}}} \geq \frac{1}{3}\}$. We can account for any edge $e = (u, v) \in \text{OPT}$ that is incident on a node in S_1, by losing at most a factor 6 (since the total weight of the edges incident on that node in OPT is at most 1, and the total weight incident on that node in OPT^{aon} is at least $\frac{1}{3}$, and edges might be counted twice, one for each endpoint).

Consider a remaining edge $e = (u, v)$ in OPT to be accounted for - from the above step, both endpoints of the edge e cannot be in the set S_1. We can now argue for why the edge e is not included in OPT^{aon} - it either leads to a weight violation or a cardinality violation (or both).

First consider the case where adding this edge leads to a weight violation, say at endpoint u. This implies that $b_u^{\text{OPT}^{\text{aon}}} + w_e > 1$, and thus $w_e > 1 - b_u^{\text{OPT}^{\text{aon}}} > \frac{2}{3}$. But if this is true, then dropping all the edges incident to u and v in OPT^{aon} and adding the edge e instead leads to a strictly better solution to the all-or-nothing case, which is a contradiction to the optimality of OPT^{aon}, and therefore this case cannot arise.

Therefore the edge e can only be part of a cardinality violation (if added), and in this scenario, we can create disjoint augmenting paths of length 2 (if only one endpoint is k-tight in OPT^{aon}) or of length 3 (if both endpoints are k-tight in OPT^{aon}). The augmenting paths imply that the weight of the edge e is bounded by the sum of the weights of the edges in the augmenting path and in OPT^{aon}. Adding this over all the remaning edges implies that the cost of these edges e is at most twice the cost of the edges in OPT^{aon} incident on vertices not in S_2, which proves our claim. □

Next, we consider a distributed approximation algorithm for an instance consisting only of heavy-weight edges.

Consider the graph $G = (L \cup R, E)$, edge weights $w : E \mapsto \mathbb{R}$ and the non-negative integer k, where the edge weights are all at least $\frac{1}{k}$. Note that in this case, we only need to ensure that a feasible solution F satisfies the weight constraint at a node, since this ensures the cardinality constraint is also satisfied,

$$|\{e \in \delta(v) \cap F\}| = k \sum_{e \in \delta(v) \cap F} \frac{1}{k} \leq k \sum_{e \in \delta(v) \cap F} w_e \leq k.$$

We partition the set of edges into different weight classes, $\mathcal{C}_1$, $\mathcal{C}_2, \ldots, \mathcal{C}_R$, where $\mathcal{C}_r = \{e \in E : \frac{1}{(1+\varepsilon)^r} < w_e \leq \frac{1}{(1+\varepsilon)^{r-1}}\}$. We can observe that the number of weight classes is $R = \frac{\log k}{\log(1+\varepsilon)}$. We define modified weights for every edge, which consist of rounding up the weights to the upper limit of the weight class to which it belongs, i.e., for an edge e that belongs to the weight class r, we let the modified weight be $w'_e = \frac{1}{(1+\varepsilon)^{r-1}}$.

From the definition of the weight classes, it is easy to see that for every edge, $w'_e \leq (1 + \varepsilon) w_e$.

We run the following distributed algorithm for R rounds, where in each round we run a distributed maximum cardinality algorithm.

Lemma 2. *On an all-or-nothing instance consisting only of heavy-weight edges and optimal value OPT, the algorithm 2, when using a δ-approximate maximum-cardinality b-matching subroutine that*

Algorithm 2: Distributed algorithm for heavy-weight edges

```
$c_v^{(1)} = 1, \quad \forall v \in V$; // dynamic total weight at each node
$F = \emptyset$;
for $r = 1$ to $R$ do
    $b_v^{(r)} := \lfloor c_v^{(r)} (1+\varepsilon)^{r-1} \rfloor$;
    Solve the max-cardinality $b$-matching instance on $G^{(r)} = (L \cup R, \mathcal{C}_r)$, with cardinality constraints $\{b_v^{(r)}\}_{\{v \in V\}}$, to obtain the solution $F^{(r)}$;
    $F \leftarrow F \cup F^{(r)}$;
    $c_v^{(r+1)} = c_v^{(r)} - \sum_{e \in F^{(r)} \cap \delta(v)} w_e$;
end
```

runs in λ rounds, returns a set of edges F such that

$$w(F) \geq \frac{\delta}{3+2\varepsilon} \cdot w(\mathsf{OPT})$$

and runs in at most $\lambda \frac{\log k}{\log(1+\varepsilon)}$ rounds.

The main idea of the proof of this lemma is that we are considering a greedy algorithm, where we select a maximal set of edges of a weight class, in decreasing order of weight. This is similar to the constant-factor greedy approximation algorithm for the maximum weight matching problem, but where we have to deal with a b-matching and selecting multiple edges at every round.

Proof. The bound on the total number of rounds is immediate from the bound on the matching subroutine used within Algorithm 2.

Consider the optimal solution OPT. We divide the set of edges in this solution into R groups based on the weight class the edge falls into, so that

$$w(\mathsf{OPT}) = w(\mathsf{OPT}^{(1)}) + \cdots + w(\mathsf{OPT}^{(R)}).$$

Now consider a round r, and let $M^{(r)}$ be the opt solution to the b-matching instance at that round, so that $|F^{(r)}| \geq \delta |M^{(r)}|$.

Let $\mathsf{OPT}_1^{(r)} = \mathsf{OPT}^{(r)} \cap M^{(r)}$ and $\mathsf{OPT}_2^{(r)} = \mathsf{OPT}^{(r)} \setminus \mathsf{OPT}_1^{(r)}$. We can then see that we can bound the weight of the edges in the first set,

$$w(\mathsf{OPT}_1^{(r)}) \leq w'(M^{(r)}) \leq \frac{1}{\delta} w'(F^{(r)}).$$

Now consider an edge $e = (u, v) \in \mathsf{OPT}_2^{(r)}$. From the fact that it is not included in $M^{(r)}$, we can see that at least one of the endpoints, say u, is $b^{(r)}$-tight, i.e., $|M^{(r)} \cap \delta(u)| = b_u^{(r)}$. We assign edge e to the vertex u. We do a similar assignment for all the edges in $\mathsf{OPT}_2^{(r)}$.

Consider a node u and the set of edges assigned to it, E_u. Let e be the edge assigned to u in the latest round (and so e belongs to the smallest-weight weight-class among the edges in E_u), and let r be the index of the corresponding weight-class. Now since the $b^{(r)}$ values were chosen so that adding one more violates the weight constraint, we can see that $w([F^{(1)} \cup \cdots \cup F^{(r-1)} \cup M^{(r)}] \cap \delta(u)) \geq 1 - w'_e \geq 1 - \frac{1}{(1+\varepsilon)^{r-1}}$..

We can also observe that since we add edges in decreasing order of weight classes, the fact that the optimal solution to the b-matching at that round could not add an edge that is in the optimum solution implies that the node has at least one edge incident to it from a higher weight class, and therefore $w([F^{(1)} \cup \cdots \cup F^{(r-1)}] \cap \delta(u)) \geq \frac{1}{(1+\varepsilon)^{r-2}}$.

Combining both, we can see that

$$\begin{aligned} w(F \cap \delta(u)) &\geq \delta \cdot w([F^{(1)} \cup \cdots \cup F^{(r-1)} \cup M^{(r)}] \cap \delta(u)) \\ &\geq \delta \cdot \max\{1 - \frac{1}{(1+\varepsilon)^{r-1}}, \frac{1}{(1+\varepsilon)^{r-2}}\} \geq \delta \cdot \frac{1+\varepsilon}{2+\varepsilon}, \end{aligned}$$

and therefore,

$$\begin{aligned} w(\mathsf{OPT}_2^{(r)}) = \sum_{u \in V} \sum_{e \in E_u} w_e &\leq \frac{1}{\delta} \cdot \frac{2+\varepsilon}{1+\varepsilon} \cdot \sum_{u \in V} w'(F^{(r)} \cap \delta(u)) \\ &\leq \frac{1}{\delta} \cdot \frac{2+\varepsilon}{1+\varepsilon} \cdot w'(F^{(r)}). \end{aligned}$$

We can sum over all the rounds to get,

$$\begin{aligned} w(\mathsf{OPT}) = \sum_{r \in [R]} w(\mathsf{OPT}^{(r)}) &\leq \frac{1}{\delta} \cdot \frac{3+2\varepsilon}{1+\varepsilon} \sum_{r \in [R]} w'(F^{(r)}) && (3) \\ &\leq \frac{1}{\delta} \cdot (3+2\varepsilon) \cdot w(F), && (4) \end{aligned}$$

and therefore the weight of the solution we obtain is bounded by

$$w(F) \geq \frac{\delta}{3+2\varepsilon} \cdot w(\mathsf{OPT}).$$

□

We now consider the (simpler) case of an instance consisting only of light-weight edges.

Lemma 3. *Given an input instance for the all-or-nothing variant consisting only of light edges and a δ-approximate maximum-weight b-matching subroutine that runs in λ rounds, we can construct, in λ rounds, a δ-approximate solution to this instance.*

Proof. Consider the subinstance of the all-or-nothing flow-matching problem, consisting of the graph $G = (L \cup R, E)$, edge weights $w : E \mapsto \mathbb{R}$ and the non-negative integer k, where the edge weights are all at most $\frac{1}{k}$. Note that in this case, we only need to ensure that a feasible solution F satisfies the cardinality constraint at a node, since this ensures the weight constraint is also satisfied,

$$\sum_{e \in \delta(v) \cap F} w_e \leq \sum_{e \in \delta(v) \cap F} \frac{1}{k} = \frac{1}{k} |\{\delta(v) \cap F\}| \leq 1.$$

The bound on the number of rounds and the approximation guarantee is thus an immediate consequence of using the provided subroutine for the maximum-weight b-matching problem. □

The final piece of the puzzle is in combining the two solutions we have obtained so far - one consisting of the heavy-weight edges and the other consisting of the light-weight edges.

The following lemma provides the means to handle the situation where we have the edge sets F_H and F_L, where $w_e > \frac{1}{k}$ for all $e \in F_H$ and $w_e \leq \frac{1}{k}$ for all $e \in F_L$.

Lemma 4. *Given disjoint edge sets $F_H, F_L \subseteq E$, where $w_e > \frac{1}{k}$ for all $e \in F_H$ and $w_e \leq \frac{1}{k}$ for all $e \in F_L$, we can construct an edge set F in a constant number of rounds, such that $w(F) \geq \frac{1}{2}\max\{w(F_L), w(F_H)\}$.*

Proof. We use the following algorithm - we start with the heavy-weight solution F_H, and each node u selects a maximal set of edges from the light-weight solution incident on that node, $E_u \subseteq F_L(u)$ that it can add without violating either the cardinality or the weight constraint at that node. We then only keep the edges from F_L that both endpoints would like, i.e, the final solution we consider is $F = F_H \cup \{e = (u,v) \in F_L : e \in E_u \text{ and } e \in E_v\} := F_H \cup F'_L$.

It is immediate that $w(F) \geq w(F_H)$ and so we only have to see how $w(F)$ compares to $w(F_L)$.

Consider a node u. We will bound the weight of the edges not chosen by that node, $F_L(u) \setminus E_u$ by the weight of the heavy edges at that node, $F_H(u)$. To prove this, consider an edge $e \in F_L(u) \setminus E_u$. Since the node u selects a maximal set of edges from F_L, the reason e is not selected is because it either violates a cardinality constraint or a weight constraint. First, suppose it violates a cardinality constaint (note that if at least one edge in $F_L(u) \setminus E_u$ viotes a cardinality constraint, then all the edges in that set do). We can then pair up every edge in $F_L(u) \setminus E_u$ with an edge in $F_H(u)$, and since the edges in the heavy-weight set have a larger weight than the edges in F_L, the claim follows. On the other hand, suppose that every edge in $F_L(u) \setminus E_u$ was part of a weight constraint violation.

Since F_L is a feasible solution, we know that $w(F_L(u) \setminus E_u) + w(E_u) \leq 1$, and maximality of the chosen edges implies that $w(F_H(u)) + w(E_u) + w_e > 1$, which together imply that $w(F_L(u) \setminus E_u) \leq w(F_H(u)) + w(e) \leq 2w(F_H(u))$.

Now we can bound the total weight of the light-weight edges as

$$\begin{aligned} w(F_L) &= w(F'_L) + \bigcup_{u \in V} w(F_L(u) \setminus E_u) \\ &\leq w(F'_L) + \bigcup_{u \in V} 2w(F_H(u)) \leq 2w(F'_L \cup F_H) = 2w(F). \end{aligned}$$

Therefore, we need only a constant number of rounds to arrive at the combined solution, and we can bound the value of the final set of edges as $w(F) \geq \max\{\frac{1}{2}w(F_L), w(F_H)\}$. □

We are now ready to prove the main theorem, restated here.

Theorem 6. *There is a randomized distributed $\Omega(1)$-approximate algorithm to the k-SFM that runs in $O(\log k \log n)$ rounds both in expectation and with high probability.*

Proof of Theorem 6. Note that we have not optimized for the constant in what follows.

We can combine lemmas 2, 3, and 4, together with the $\frac{1}{2}$-approx maximum-weight b-matching algorithm from [28] that runs in $O(\log n)$ rounds both in expectation and in high probability to obtain a $\frac{1}{12+\varepsilon}$-approximate solution to the all-or-nothing variant that runs in $O(\log n \frac{\log k}{\log(1+\varepsilon)})$ rounds. Combining this with lemma 1 gives us a $\frac{1}{72+\varepsilon}$-approximate solution to the k-SFM problem in $O(\log n \frac{\log k}{\log(1+\varepsilon)})$ rounds. □

6 CONCLUSION

This paper initiates a theoretical study of flow scheduling in datacenter networks. Motivated by the fact that modern datacenter networks use Clos-like topologies similar to switch fabrics, we explore new connections between the classical problem of packet scheduling in switch fabrics, and the increasingly important problem of flow scheduling in datacenter networks. To model the latter, we introduced and studied a new variant of the matching problem, that we refer to as k-sparse flow-matching problem, in which a vertex in the graph has both a weight and a cardinality constraint associated with it. We present constant-factor centralized and distributed approximation algorithms for this problem.

Our work opens up several avenues of future research. The most natural question is to improve the approximation bound and/or the number of rounds needed to achieve near-optimal solutions in the distributed setting. It is *a priori* conceivable that, in distributed settings, it should be possible to achieve performance similar to the best known upper bounds for the b-matching problem (which the k-SFM problem generalizes). Furthermore, our current understanding of the analysis of the faster distributed algorithm with $O(\log n \log k)$ rounds does not preclude results with a smaller constant factor approximation.

ACKNOWLEDGMENTS

The authors would like to thank PODC reviewers for their insightful feedback. This work was supported in part by NSF CCR-1522054, CCR-1526067, CCR-1839346, CCR-1740822, CSR-1704742, CNS-2047283, a Sloan Research Fellowship, and a gift from Google.

REFERENCES

[1] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: minimal near-optimal datacenter transport. In *SIGCOMM*, 2013.
[2] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker. High speed switch scheduling for local area networks. *ACM Transactions on Computer Systems*, 1993.
[3] B. Awerbuch, M. T. Hajiaghayi, R. D. Kleinberg, and T. Leighton. Online client-server load balancing without global information. In *SODA*, 2005.
[4] R. Bar-Yehuda, K. Censor-Hillel, M. Ghaffari, and G. Schwartzman. Distributed approximation of maximum independent set and maximum matching. In *PODC*, 2017.
[5] S. J. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(4):509–522, 2002.
[6] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *IMC*, 2010.
[7] D. Bertsimas, V. F. Farias, and N. Trichakis. Fairness, efficiency, and flexibility in organ allocation for kidney transplantation. *Oper. Res.*, 61(1):73–87, 2013.
[8] G. E. Blelloch, J. T. Fineman, and J. Shun. Greedy sequential maximal independent set and matching are parallel on average. In *SPAA*, 2012.
[9] Q. Cai, M. T. Arashloo, and R. Agarwal. dcPIM: Near-optimal proactive datacenter transport. In *SIGCOMM*, 2022.
[10] Q. Cai, S. Chaudhary, M. Vuppalapati, J. Hwang, and R. Agarwal. Understanding host network stack overheads. In *SIGCOMM*, 2021.
[11] L. Charlin and R. Zemel. The toronto paper matching system: an automated paper-reviewer assignment system. 2013.
[12] S. Chattopadhyay, L. Higham, and K. Seyffarth. Dynamic and self-stabilizing distributed matching. In *PODC*, 2002.
[13] T. Cheung. Graph traversal techniques and the maximum flow problem in distributed computation. *IEEE Transactions on Software Engineering*, 1983.
[14] J. P. Dickerson and T. Sandholm. Futurematch: Combining human value judgments and machine learning to match in dynamic environments. In *AAAI*, 2015.
[15] J. Drummond, A. Perrault, and F. Bacchus. SAT is an effective and complete method for solving stable matching problems with couples. In *IJCAI*, 2015.
[16] R. Duan and S. Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM*, 2014.
[17] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 1965.
[18] J. R. Edmonds and E. L. Johnson. Matching: A well-solved class of integer linear programs. In *Combinatorial Optimization - Eureka, You Shrink!, Papers Dedicated*

to Jack Edmonds, 5th International Workshop, volume 2570 of *Lecture Notes in Computer Science*, pages 27–30. Springer, 2001.

[19] G. Even, M. Medina, and D. Ron. Distributed maximum matching in bounded degree graphs. In *ICDCN*, 2015.

[20] N. Farrington, E. Rubow, and A. Vahdat. Data center switch architecture in the age of merchant silicon. In *HOTI*, 2009.

[21] M. Fischer. Improved deterministic distributed matching via rounding. *Distributed Computing*, 2020.

[22] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker. phost: distributed near-optimal datacenter transport over commodity network fabric. In *CoNext*, 2015.

[23] M. Ghaffari, T. Gouleakis, C. Konrad, S. Mitrovic, and R. Rubinfeld. Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *PODC*, 2018.

[24] M. Ghaffari, A. Karrenbauer, F. Kuhn, C. Lenzen, and B. Patt-Shamir. Near-optimal distributed maximum flow: Extended abstract. In *PODC*, 2015.

[25] S. Hu, W. Bai, G. Zeng, Z. Wang, B. Qiao, K. Chen, K. Tan, and Y. Wang. Aeolus: A building block for proactive transport in datacenters. In H. Schulzrinne and V. Misra, editors, *SIGCOMM*, 2020.

[26] A. Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 1986.

[27] A. M. Khan, A. Pothen, M. M. A. Patwary, M. Halappanavar, N. R. Satish, N. Sundaram, and P. Dubey. Designing scalable *b*-matching algorithms on distributed memory multiprocessors by approximation. In *SC*, 2016.

[28] C. Koufogiannakis and N. E. Young. Distributed fractional packing and maximum weighted b-matching via tail-recursive duality. In *DISC*, 2009.

[29] E. Krissinel and K. Henrick. Secondary-structure matching (ssm), a new tool for fast protein structure alignment in three dimensions. *Acta Crystallographica Section D: Biological Crystallography*, 2004.

[30] F. Kuhn, T. Moscibroda, and R. Wattenhofer. Local computation: Lower and upper bounds. *Journal of the ACM*, 2016.

[31] R. Kurata, N. Hamada, A. Iwasaki, and M. Yokoo. Controlled school choice with soft bounds and overlapping types. *J. Artif. Intell. Res.*, 58:153–184, 2017.

[32] S. Lattanzi, B. Moseley, S. Suri, and S. Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *SPAA*, 2011.

[33] N. Linial. Distributive graph algorithms-global solutions from local data. In *FOCS*, 1987.

[34] X. Liu, T. Suel, and N. D. Memon. A robust model for paper reviewer assignment. In *RecSys*, 2014.

[35] Z. Lotker, B. Patt-Shamir, and S. Pettie. Improved distributed approximate matching. In *SPAA*, 2008.

[36] Z. Lotker, B. Patt-Shamir, and S. Pettie. Improved distributed approximate matching. *Journal of the ACM*, 2015.

[37] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri. Multicast traffic in input-queued switches: optimal scheduling and maximum throughput. *IEEE/ACM Transactions on Networking*, 2003.

[38] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 1999.

[39] N. McKeown, A. Mekkittikul, V. Anantharam, and J. C. Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Transactions on Communications*, 1999.

[40] N. McKeown, P. Varaiya, and J. Walrand. Scheduling cells in an input-queued switch. *Electronics Letters*, 1993.

[41] B. Montazeri, Y. Li, M. Alizadeh, and J. K. Ousterhout. Homa: a receiver-driven low-latency transport protocol using network priorities. In *SIGCOMM*, 2018.

[42] A. Panconesi and M. Sozio. Fast primal-dual distributed algorithms for scheduling and matching problems. *Distributed Computing*, 2010.

[43] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: a centralized "zero-queue" datacenter network. In *SIGCOMM*, 2014.

[44] T. L. Pham, I. Lavallée, M. Bui, and S. H. Do. A distributed algorithm for the maximum flow problem. In *(ISPDC)*, 2005.

[45] M. D. Plummer and L. Lovász. *Matching theory*. Elsevier, 1986.

[46] W. R. Pulleyblank. Matchings and extensions. *Handbook of combinatorics*, 1:179–232, 1995.

[47] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *SIGCOMM*, 2015.

[48] D. Shah, P. Giaccone, and B. Prabhakar. Efficient randomized algorithms for input-queued switch scheduling. *IEEE Micro*, 2002.

[49] D. Shah and D. Wischik. Optimal scheduling algorithms for input-queued switches. In *INFOCOM*, 2006.

[50] R. Shashidhara, T. Stamler, A. Kaufmann, and S. Peter. FlexTOE: Flexible TCP offload with fine-grained parallelism. In *NSDI*, 2022.

[51] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *SIGCOMM*, 2015.

[52] R. E. Tarjan. *Data structures and network algorithms*, volume 44 of *CBMS-NSF regional conference series in applied mathematics*. SIAM, 1983.

[53] M. Wattenhofer and R. Wattenhofer. Distributed weighted matching. In *DISC*, 2004.