Scheduling Flows on a Switch to Optimize Response Times

Hamidreza Jahanjou
Google
hamidrj@google.com

Rajmohan Rajaraman
Northeastern University
rraj@ccs.neu.edu

David Stalfa
Northeastern University
stalfa@ccs.neu.edu

ABSTRACT

We study the scheduling of flows on a switch with the goal of optimizing metrics related to the response time of the flows. The input is a sequence of flow requests on a switch, where the switch is represented by a bipartite graph with a capacity on each vertex (port), and a flow request is an edge with associated demand. In each round, a subset of edges can be scheduled under the constraint that the total demand of the scheduled edges incident on any vertex is at most the capacity of the vertex. This class of scheduling problems has applications in datacenter networks, and has been extensively studied. Previous work has essentially settled the complexity of metrics based on *completion time*. The objective of average or maximum *response time*, however, is more challenging.

We present approximation algorithms for flow scheduling over a switch to optimize response time based metrics. For the average response time metric, whose NP-hardness follows directly from past work, we present an offline $O(1+O(\log(n))/c)$ approximation algorithm for unit flows, assuming that the port capacities of the switch can be increased by a factor of $1+c$, for any given positive integer c. For the maximum response time metric, we first establish that it is NP-hard to achieve an approximation factor of better than 4/3 without augmenting capacity. We then present an offline algorithm that achieves *optimal maximum response time*, assuming the capacity of each port is increased by at most $2d_{max}-1$, where d_{max} is the maximum demand of any flow. Both algorithms are based on linear programming relaxations. We also study the online version of flow scheduling using the lens of competitive analysis, and present preliminary results along with experiments that evaluate the performance of fast online heuristics.

ACM Reference Format:
Hamidreza Jahanjou, Rajmohan Rajaraman, and David Stalfa. 2020. Scheduling Flows on a Switch to Optimize Response Times. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '20), July 15–17, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3350755.3400218

1 INTRODUCTION

With the advent of software-defined networking (SDN) and OpenFlow switch protocol, routing and scheduling in modern data center networks is increasingly performed at the level of flows. A *flow* is a particular set of application traffic between two endpoints that receive the same forwarding decisions. As a consequence of the shift towards centralized flow-based control, efficient algorithms for scheduling and routing of flows and their variants have gained prominent importance [11, 15, 28, 39, 47].

In order to model the datacenter network carrying the flows, it is common to represent the entire datacenter network as one non-blocking switch (see Figure 1) interconnecting all machines [2, 5, 35, 47]. This simple model is attractive because of advances in full-bisection bandwidth topologies [29, 46]. In this model, every input (ingress) port is connected to every output (egress) port. Bandwidth limits are at the ports and the interconnections are assumed to have unlimited bandwidth. We model the datacenter network as a general bipartite graph (which includes the full-bisection as a special case) with capacities at each vertex (port).

In the context of scheduling and client-server applications, *response time*–also known as flow time or sojourn time–is a very natural and important objective. Indeed, response time is directly related to quality of service experienced by clients [6, 19]. In the job scheduling literature, metrics related to response times have been extensively studied in diverse frameworks, including approximation algorithms [8, 9, 14, 21, 37], competitive analysis [7, 34, 45], and queuing-theoretic analysis [12, 30]. For flow scheduling, however, response time optimization is not as well-understood as completion time optimization; to the best of our knowledge, there is no prior work on approximation algorithms for flow scheduling to optimize response time metrics. In this paper, we study the problem of scheduling flows on a switch network to minimize average response time and maximum response time.

1.1 Results

We present approximation algorithms for flow scheduling on a bipartite switch network to minimize response time metrics.

- We present a $(1+c, O(\log n/c))$-approximation algorithm, running in polynomial time, for scheduling n *unit flows* under the

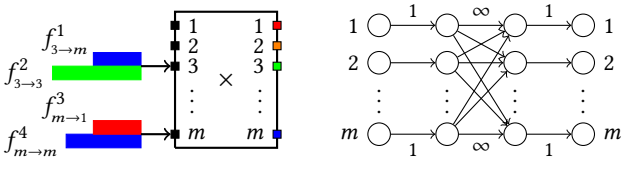

Figure 1: (left) An $m\times m$ non-blocking switch with unit port capacities. Each incoming flow is shown as a bar on the left, with the length of the bar proportionate to the flow size. Each flow also specifies its input and output ports. For instance, two flows f^1 and f^4 share the same destination port. (right) The switch can be regarded as a complete $m\times m$ bipartite graph augmented with two sets of parallel edges.

SPAA '20, July 15–17, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-6935-0/20/07…$15.00
https://doi.org/10.1145/3350755.3400218

average response time metric, for any given positive integer c; that is, our algorithm achieves an average response time of $O(\log n)/c$ times the optimal assuming it is allowed port capacity that is $1+c$ times that of the original. Our results on average response time appear in Section 3.

- We show that it is NP-hard to attain an approximation factor smaller than 4/3 for the maximum response time metric. We next present a polynomial-time algorithm that achieves *optimal* maximum response time, assuming it is allowed port capacity that is at most $2d_{max} - 1$ more than that of the optimal, where d_{max} is the maximum demand of any flow request. For the special case of unit demands, note that this is best possible, given the hardness result. Our results on maximum response time appear in Section 4.

Both of our algorithms are based on rounding a suitable linear programming relaxation of the associated problem. The algorithm for average response time uses the iterative rounding paradigm, along the lines of previous work in scheduling jobs on unrelated machines [8]. A challenge we need to address is that a "job" in flow scheduling uses two different capacitated "resources" (ports) simultaneously. We are able to overcome this challenge if we allow resource augmentation. An important open problem is to determine whether polylogarithmic- or better approximations for average response time are achievable without resource augmentation.

For maximum response time, our hardness reduction is through the classic Timetable problem [20] and provides a useful target for practitioners developing heuristics. Our approximation algorithm is achieved by applying a rounding theorem of [36], and in fact extends to the more general problem in which we need to meet *distinct deadlines* for individual flows.

Both the algorithms above are offline approximations. In Section 5, we study online algorithms for response time metrics.

- We present preliminary theoretical results including a resource-augmented constant-factor competitive algorithm for maximum response time, which builds on our offline algorithm. We next present experimental evaluations of natural online heuristics for average and maximum response time metrics.

Our work leaves some intriguing open problems and several directions for future research, which are highlighted in Section 6.

1.2 Related Work

There is considerable work on scheduling flows on non-blocking switch networks as well as more general topologies, primarily for completion time metrics. There is extensive literature on scheduling matchings over high-speed crossbar switches; these studies largely adopt a queuing theoretic framework (e.g., see [25, 27, 50]). In [16], Chowdhury et al. present effective heuristics for scheduling generalizations of flows, called co-flows, without release times on a non-blocking switch network. More recently, Luo et al. [44] provide heuristics for scheduling multicast flows over a reconfigurable switch. Approximation algorithms for average completion time of co-flows on a non-blocking switch are given in [1, 38, 48, 49]. Scheduling over general network topologies is studied in [15, 31, 51], including approximation algorithms for average completion time.

Average response time. The single machine preemptive case with release times, $1|pmtn, q_i|\sum_i R_i$, is solvable in polynomial time using the shortest remaining processing time (SRPT) rule [4]. Without preemption, $1||\sum_i R_i$ is solvable using the shortest processing time (SPT) rule; but, $1|q_i|\sum_i R_i$ is hard to approximate within a factor of $n^{\frac{1}{2}-\epsilon}$ for all $\epsilon > 0$ [37]. For two machines or more, $P2|pmtn, q_i|\sum_i R_i$ is **NP**-hard [18]. Leonardi and Raz show that SRPT is an $O(\log(\min(\frac{n}{m}, P)))$-competitive algorithm for the problem $Pm|pmtn, q_i|R_i$ where P is the ratio between the largest and the smallest job processing times [43]. From a technical standpoint, a related paper for our work is that of Garg and Kumar, who consider the problem of minimizing total response time on related machines $(Q|pmtn, q_i|\sum_i R_i)$ and present an offline $O(\log P)$-approximation algorithm and an online $O(\log^2 P)$-competitive algorithm [22]. In a later paper, the same authors consider the problem of minimizing total response time on multiple identical machines where each job can be assigned to a specified subset of machines. They give an $O(\log P)$-approximation algorithm as well as an $\Omega(\frac{\log P}{\log\log P})$ lower bound [23]. The same ideas were used to get an $O(k)$-approximation algorithm for the unrelated case $(R|pmtn, q_i|\sum_i R_i)$ when there are k different processing times [24]. In the same paper, the authors showed an $\Omega(\log^{1-\epsilon} P)$ hardness of approximation for $P|pmtn, q_i|\sum_i R_i$. More recently, Bansal and Kulkarni design an $O(\min(\log^2 n, \log n \log P))$-approximation algorithm for $R|pmtn, q_i|\sum_i R_i$, which provides a basis for our algorithm for average response time [8].

Independently, Dinitz and Moseley [17] have recently studied online scheduling of flows in reconfigurable networks and provide an $O(1/\varepsilon^2)$-competitive algorithm, assuming that the speed of each machine is $2+\varepsilon$ times that in an optimal solution. One consequence of their result is an $O(1/c^2)$-competitive algorithm for average response time in our model, assuming a $(2+c)$ factor blowup in port capacity, for any positive integer c. In contrast, our result for average response time requires a $(1+c)$-factor blowup, for any positive integer c, but incurs a logarithmic approximation ratio and holds only for the offline model. We refer the reader to the full paper for a comparison of our models [17].

Maximum response time. The problem of minimizing maximum response time has not been studied extensively. $P|pmtn, q_i|R_{\max}$ is polynomial-time solvable [42]. The first-in first-out (FIFO) heuristic is known to be $(3-\frac{2}{m})$-competitive for $Pm|pmtn, q_i|R_{\max}$ and $Pm|q_i|R_{\max}$ [10, 45]. On the other hand, Ambühl and Mastrolilli give a $(2-\frac{1}{m})$-competitive algorithm for $Pm|pmtn, q_i|R_{\max}$ and show that FIFO achieves the best possible competitive ratio on two identical machines when preemption is not allowed [3]. [8] gives an $O(\log n)$-approximation algorithm for $R|pmtn, q_i|R_{\max}$.

2 PROBLEM DEFINITIONS AND NOTATION

We consider two scheduling problems in which flows arrive in fixed intervals on a non-blocking switch. In this model, we are given a switch $S_{m,m'} = (P, F)$ where P is a set of m *input* ports and m' *output* ports where each port p has a corresponding *capacity* c_p. F is a set of flows $e = pq$ with one input port p and one output port q. Each flow e has a corresponding *demand* d_e and *release time* r_e. We assume throughout that for any $e = pq$, $d_e \leq \kappa_e = \min(c_p, c_q)$.

For an given instance $S_{m,m'}$, we define a family of functions $\sigma : F \times \mathbb{N} \to \{0, 1\}$. We say that σ *schedules* flow e in *round* t if $\sigma_{e,t} = 1$ (for ease of notation, we use $\sigma_{e,t} \equiv \sigma(e, t)$). A function σ is a *schedule* of $S_{m,m'}$ if the following conditions are met: every flow e, is entirely scheduled across all rounds (i.e. $\sum_t \sigma_{e,t} \geq 1$), every flow e is scheduled only in rounds after its release time (i.e. for all t, $\sigma_{e,t} = 1 \Rightarrow t \geq r_e$), and for all ports p the total size of all flows scheduled on port p in a given round is no more than p's capacity (i.e. for all t, $\sum_{e:p\in e} d_e \sigma_{e,t} \leq c_p$). For a given flow e and schedule σ, the *response time* ρ_e is the difference in its *completion time* $C_e = 1 + \min\{t : \sigma_{e,t} = 1\}$ and its release time, i.e. $\rho_e = C_e - r_e$.

The first problem we study in this model is Flow Scheduling to Minimize Average Response Time (FS-ART) in which we seek to minimize $\sum_{e\in F} C_e - r_e$. The second problem we study in this model is Flow Scheduling to Minimize Maximum Response Time (FS-MRT) in which we seek to minimize $\max_{e\in F}\{C_e - \rho_e\}$.

Throughout the paper we use pq to denote a flow (directed edge) from input port p to output port q. We use $[i]$ to denote the set of positive integers less than or equal to i. An instance with equal numbers of input and output ports is referred to as S_m. The main notation is given in the table below.

$S_{m,m'}$	:	m-in, m'-out	e, pq	:	flow
P	:	all ports	d_e	:	e's demand
F	:	all flows	r_e	:	e's release time
n	:	$\lvert F\rvert$	ρ_e	:	e's response time
p, q	:	port	C_e	:	e's completion time
c_p	:	p's capacity	t	:	round
κ_{pq}	:	$\min\{c_p, c_q\}$	σ	:	schedule
F_p	:	all $e : p \in e$	$\sigma_{e,t} = 1$	$\Leftrightarrow$	e scheduled at t

3 AVERAGE RESPONSE TIME

We study Flow Scheduling to Minimize Average Response Time (FS-ART), for instances with identical numbers of input and output ports. Specifically, we assume each instance is an $m \times m$ switch S_m.

From a complexity viewpoint, FS-ART generalizes classic scheduling problems. The special case of FS-ART with arbitrary demands, unit capacity, and $m = 1$ is equivalent to preemptive single-machine scheduling with release times, which is strongly **NP**-hard when the objective is weighted sum of completion times ($1|r_i; pmtn|\sum w_i c_i$). Note that, $1|r_i; pmtn|\sum c_i$ is polynomial-time solvable while the complexity of $1|r_i; \sigma_i = \sigma; pmtn|\sum w_i c_i$ is still open.

For $m > 1$, FS-ART instances incur coupling issues, even for unit demands. Each flow requires resources at two ports *simultaneously*. In [26], the authors consider the closely related *biprocessor scheduling* problem: there are m identical machines and n unit-sized jobs which require simultaneous use of two pre-specified (dedicated) machines. The objective is to minimize total completion time of jobs. The hardness of this problem is related to the graph that arises from the pre-specified machine pairs (machines correspond to nodes and edges to jobs). It is shown in [26] that the problem is strongly NP-hard if the graph is cubic, and remains NP-hard if the graph is bipartite and subcubic (i.e. $\forall v : deg(v) \leq 3$), which implies that FS-ART is NP-hard even for unit demands and unit capacities and identical release times for all flows. While constant-factor approximations [26, 40] are known for makespan and average completion time, no results are known for response time metrics.

Section 3.1 presents a linear programming approach based on iterative rounding,. Section 3.2 uses this approach to establish the main approximation result of this section.

3.1 A linear-programming approach

In this section, we investigate linear programming approaches used in the context of machine scheduling and adapt them to our setting. On a conceptual level, our problem is harder than parallel/related/unrelated machine scheduling in the sense that we have to deal with simultaneous use of ports, but is easier in the sense that we do not have to worry about the assignment of flows/jobs to machines as each flow specifies its source and destination ports.

Our starting point is the following linear program similar to the one used by Garg and Kumar [22].

$$\text{Minimize} \sum_e \sum_{t\geq r_e} \left(\frac{t - r_e}{d_e} + \frac{1}{2\kappa_e}\right) b_{et} \qquad \text{subject to} \qquad (1)$$

$$\sum_{t\geq r_e} b_{et} \geq d_e \qquad \forall e \qquad (2)$$

$$\sum_{e\in F_p} b_{et} \leq c_p \qquad \forall p, t \qquad (3)$$

$$b_{et} \geq 0 \qquad \forall e, t \qquad (4)$$

Informally, the variable b_{et} gives the amount of flow e that is scheduled in round t. Constraint (2) ensures that each flow is completed. Constraint (3) ensures that no port is overloaded in any round. We can rewrite the objective function as $\sum_e \Delta_e$ where

$$\Delta_e = \sum_{t\geq r_e} \left(\frac{t - r_e}{d_e} + \frac{1}{2\kappa_e}\right) b_{et}$$

is the *fractional response time* of e. We show that, for a given instance $S_{n,m}$ of FS-ART, the optimal solution to (1) - (4) lower bounds the total response time of any schedule of $S_{n,m}$.

Lemma 3.1. *For an arbitrary $S_{n,m}$, let σ be some (non-integral) schedule of $S_{n,m}$ and let b^* (Δ^*_e) be the optimal solution to (1) - (4) corresponding to $S_{n,m}$. Then $\sum_e \Delta^*_e \leq \sum_e \rho_e$.*

Proof. Given σ, we construct a solution to (1) - (4) by setting $b_{et} \leftarrow (1/d_e)\sigma_{e,t}$, for all flows e and rounds t. To prove the lemma, we prove the stronger claim that, for any flow e, $\Delta_e \leq \rho_e$.

Suppose that the completion time of flow e in schedule σ is C_e. Then the response time of e is $\rho_e = C_e - r_e$. Notice that

$$\Delta_e = \sum_{t=r_e}^{C_e} \left(\frac{t - r_e}{d_e} + \frac{1}{2\kappa_e}\right) b_{et} \;\leq\; \sum_{t=C_e - d_e/\kappa_e}^{C_e} \left(\frac{t - r_e}{d_e} + \frac{1}{2\kappa_e}\right) \kappa_e.$$

That is, Δ_e is maximized when as much of flow e is scheduled in each round as possible to ensure that e completes in round C_e. But,

$$\sum_{t=C_e - d_e/\kappa_e}^{C_e} \left(\frac{t - r_e}{d_e} + \frac{1}{2\kappa_e}\right) \kappa_e = \sum_{t=1}^{d_e/\kappa_e} \left(\frac{C_e - r_e - t}{d_e} + \frac{1}{2\kappa_e}\right) \kappa_e$$

$$= C_e - r_e - \frac{1}{2} \leq \rho_e$$

which completes the proof. □

We now consider another linear programming formulation first used by Bansal and Kulkarni [8] for the problem of job scheduling

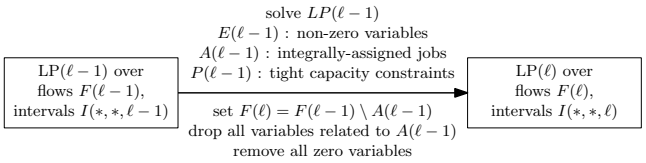

Figure 2: The ℓ-th iteration of the rounding scheme, $\ell \geq 1$, starts by solving $LP(\ell-1)$ and ends by defining $LP(\ell)$.

on unrelated machines. The authors use iterative rounding to get a tentative schedule with low additive overload for any interval of time. We do the same. This linear program and the subsequent ones, used in iterative rounding, are all interval-based. In the initial program, which we denote $LP(0)$, the interval size is 4. In subsequent relaxations, the interval size can grow. $LP(0)$ is the following program along with constraints (2) and (4).

$$\text{Minimize} \sum_{e} \sum_{t \geq r_e} \left(\frac{t - r_e}{d_e} + \frac{1}{2} \right) b_{et} \qquad \text{subject to} \qquad (5)$$

$$\sum_{e \in F_p} \sum_{t \in (4(a-1), 4a]} b_{et} \leq 4c_p \qquad \forall p, a \qquad (6)$$

As before, the real variable b_{et} is the amount of flow e scheduled in round $[t, t+1)$. Constraint (6) ensures that the total sum of flows scheduled on a given port p in any four consecutive rounds is no more than four times the capacity of p. Clearly, this new LP is a relaxation of the previous one; consequently, the value of an optimal solution to this LP is a lower bound to the response time for any integral schedule. Following [8], we use an iterative rounding scheme to get the following result.

Lemma 3.2. *The exists a solution $b^* = \{b^*_{et}\}_{e,t}$ satisfying the following properties*

(1) For each flow e, there is exactly one round t for which $b^*_{et} = d_e$.
(2) The cost of b^* is at most that of an optimal solution to the LP.
(3) For any port p and any time interval $[t_1, t_2]$,

$$\sum_{e \in F_p} \sum_{t \in [t_1, t_2]} b^*_{et} \;\leq\; c_p(t_2 - t_1) + O(c_p \log n).$$

Remark 3.3. We can regard a solution satisfying the properties in the lemma as a sequence of bipartite graphs $\{G_t\}_t$. Then, for any given (time) interval $[a, b]$, the degree of any vertex p in the "combined" graph $\cup_{t \in [a,b]} G_t$ is at most $(b-a)c_p + O(c_p \log n)$. In Section 3.2, we convert this sequence to a sequence of matchings.

Iterative rounding. To establish Lemma 3.2, we iteratively relax variable assignments with a sequence of linear programs which we denote by $LP(\ell)$ for $\ell = 0, 1, \ldots$. Recall that $LP(0)$ is the initial linear program above. We denote the set of flows that appear in $LP(\ell)$ by $F(\ell)$ and an optimal solution to $LP(\ell)$ by $b^\ell = \{b^\ell_{et}\}_{e,t}$. Let $E(\ell)$ be the set of variables in $LP(\ell)$ with non-zero assignments. Let $A(\ell)$ be the set of flows e such that, for all t, b^ℓ_{et} is integral. Let $P(\ell)$ be the set of tight capacity constraints (7) in $LP(\ell)$ given b^ℓ. Let $d_{\max} = \max_e\{d_e\}$. See Figure 2 for a high level overview.

In each iteration $\ell \geq 1$, we construct $LP(\ell)$ as follows.

- Initialize $F(\ell) = F(\ell-1)$.
- Find an optimal solution $\{b^{\ell-1}_{et}\}_{e,t}$ to $LP(\ell-1)$.
- Eliminate zero variables. In other words the variables b_{et} in $LP(\ell)$ are only defined for variables in $E(\ell-1)$, the support of $b^{\ell-1}$.
- Fix integral assignments. For all $e \in A(\ell-1)$, assign e to those rounds t such that $b^{\ell-1}_{et} > 0$ (i.e. set $b^*_{et} \leftarrow d_e$) and drop all variables b_{et} in $LP(\ell)$. We also update $F(\ell) = F(\ell) \setminus \{e\}$.
- Define intervals for the current iteration as follows. Fix a port p and consider the flows in $F(\ell) \cup F_p$. Sort all the variables in $\{b^{\ell-1}_{et} \in E(\ell-l) : e \in F(\ell) \cup F_p\}$ in increasing order of t, breaking ties lexicographically. Next, iteratively partition $b^{\ell-1}_{et}$ variables into groups $I(p, 1, \ell), I(p, 2, \ell), \ldots$ as follows. To construct group $I(p, a, \ell)$, start from the earliest non-grouped variable and greedily group consecutive $b^{\ell-1}_{et}$ variables until their sum first exceeds $4c_p$. The size of the interval $I = I(p, a, \ell)$ is

$$\text{Size}(I) = \sum_{b_{et} \in I} b^{\ell-1}_{et}.$$

  Note that $\text{Size}(I(p, a, \ell)) \in [4c_p, 5c_p)$. The time duration of I can be much larger than its size. On the other hand, for $\ell = 0$, all intervals are of size 4 as evident in the initial LP.

For $\ell \geq 1$, $LP(\ell)$ is given by objective (5) subject to constraints (2), (4), and the following constraint.

$$\sum_{e \in F_p \cap F(\ell)} \sum_{b_{et} \in I(p,a,\ell)} b_{et} \leq \text{Size}(I(p, a, \ell)) \cdot c_p \qquad \forall p, a \qquad (7)$$

Since $LP(\ell)$ is a relaxation of $LP(\ell-1)$, the second requirement of Lemma 3.2 is satisfied. Also, by construction of $LP(\ell)$, the sequence of iterations results in an integral assignment of all flows and so the first requirement of Lemma 3.2 is satisfied. It remains to bound the number of iterations and calculate the backlog.

Recall that $F(\ell)$ is the set of flows e such that variables b_{et} appear in $LP(\ell)$. Note that, for $\ell > 0$, these are the non-zero variables which correspond to non-integrally-assigned jobs after solving $LP(\ell-1)$.

Lemma 3.4. *For all $\ell \geq 1$, $|F(\ell)| \leq |F(\ell-1)|/2$.*

Proof. Consider a linearly independent set of tight constraints in $LP(\ell-1)$. Since a tight non-negativity constraint (4) results in a zero variable, the number of non-zero variables, $E(\ell-1) \subseteq \{b^{\ell-1}_{f,t}\}_{f,t}$, is at most the number of tight flow constraints (2) plus the number of tight capacity constraints (7). That is

$$|E(\ell-1)| \leq |F(\ell-1)| + |P(\ell-1)|. \qquad (8)$$

since $|F(\ell-1)|$ is the number of flow constraints.

Now, each flow which is not integrally assigned by $b^{\ell-1}$ (i.e. not in $A(\ell-1)$) contributes at least two to $|E(\ell-1)|$. Thus,

$$\begin{aligned} |E(\ell-1)| &\geq |A(\ell-1)| + 2(|F(\ell-1)| - |A(\ell-1)|) \\ &= |F(\ell-1)| + |F(\ell)|. \end{aligned} \qquad (9)$$

The equality holds since $F(\ell) = F(\ell-1) \setminus A(\ell)$ by construction. Inequalities (8) and (9) together imply $|F(\ell)| \leq |P(\ell-1)|$.

Next, we show that $|P(\ell-1)| \leq |F(\ell-1)|/2$ which completes the proof. This is accomplished by a simple combinatorial argument. Let's give 2 tokens to every flow in $F(\ell-1)$. Now, each flow $e \in F(\ell-1)$, gives a portion equal to $b^{\ell-1}_{et}/d_e$ of its tokens to the interval that contains b_{et}. This token distribution is valid since

$$\sum_p \sum_t b^{\ell-1}_{et} = 2 \sum_t b^{\ell-1}_{et} = 2d_e,$$

where we have used the fact that each flow appears in exactly two port constraints. At the same time, each tight capacity constraint for port p receives at least 4 tokens since interval sizes are $\geq 4c_p$ by definition and $d_e \leq c_p$ by assumption. Now, as each job distributes exactly 2 tokens and each tight port constraint receives at least 4, we conclude that $|P(\ell-1)| \leq |F(\ell-1)|/2$. □

Lemma 3.4 shows that the number of iterations needed before arriving at an integral solution is no more than $O(\log n)$. What remains is to bound amount of extra load that any interval has taken on. Recall that $A(\ell)$ denotes the set of flows which are integrally assigned by the optimal solution b^ℓ to $LP(\ell)$. Let $A(\ell, p, t_1, t_2) \subseteq A(\ell)$ be the set of flows which are integrally assigned to port p in the interval $[t_1, t_2]$ by the optimal solution b^ℓ of $LP(\ell)$. Furthermore, we define

$$\mathrm{Vol}(p, \ell, t_1, t_2) = \sum_{e \in F_p \cap F(\ell)} \sum_{t \in [t_1, t_2]} b^\ell_{et} + \sum_{\ell' \leq \ell} |A(\ell', p, t_1, t_2)|,$$

which is the total size of flows assigned to port p in the interval $[t_1, t_2]$by $b^0, \ldots, b^\ell$. The following lemma states that the amount of extra load taken on any port in any interval is no more than a constant additive over the load in the previous iteration.

Lemma 3.5. *For any $[t_1, t_2]$, any port p, and any round $\ell \geq 1$,*

$$\mathrm{Vol}(p, \ell, t_1, t_2) \leq \mathrm{Vol}(p, \ell-1, t_1, t_2) + 10c_p. \tag{10}$$

Proof. Fix an interval $[t_1, t_2]$ and a port p. In each iteration ℓ, the "extra" load in this interval can be introduced only if two intervals overlap with the boundaries of $[t_1, t_2]$.

Consider a maximal set of contiguous intervals $I(\ell, a, p)$, $I(\ell, a+1, p)$, ..., $I(\ell, a+w, p)$ that contain $[t_1, t_2]$. Note that a is the smallest index such that $I(\ell, a, p)$ contains some b^ℓ_{et} with $t \in [t_1, t_2]$. Similarly, w is the largest number such that $I(\ell, a+w, p)$ contains some b^ℓ_{et} with $t \in [t_1, t_2]$. Since each interval is of size smaller than $5c_p$,

$$\sum_{b_{et} \in I(\ell, a, p)} b^\ell_{et} + \sum_{b_{et} \in I(\ell, a+w, p)} b^\ell_{et} < 10c_p. \tag{11}$$

Moreover,

$$\sum_{x=a+1}^{a+w-1} \sum_{b_{et} \in I(\ell, x, p)} b^\ell_{et} \leq \sum_{x=a+1}^{a+w-1} \mathrm{Size}(I(\ell, x, p))$$
$$= \sum_{x=a+1}^{a+w-1} \sum_{b_{et} \in I(\ell, x, p)} b^{\ell-1}_{et} \leq \sum_{e \in F_p \cap F(\ell)} \sum_{t \in [t_1, t_2]} b^{\ell-1}_{et},$$

where the first inequality follows from the port capacity constraints (3), and the second equality follows from the definition of Size$(*)$. Consequently, we have that

$$\sum_{e \in F_p \cap F(\ell)} \sum_{t \in [t_1, t_2]} b^\ell_{et} \leq \sum_{x=a}^{a+w} \sum_{b_{et} \in I(\ell, x, p)} b^\ell_{et}$$
$$< 10c_p + \sum_{e \in F_p \cap F(\ell)} \sum_{t \in [t_1, t_2]} b^{\ell-1}_{ft}$$
$$\leq 10c_p - |A(\ell-1, t_1, t_2, p)| + \sum_{e \in F_p \cap F(\ell-1)} \sum_{t \in [t_1, t_2]} b^{\ell-1}_{et},$$

where the last step uses the fact that $F(\ell) = F(\ell-1) \backslash A(\ell)$.

The LHS of (10) equals

$$\sum_{f \in F_p \cap F(\ell)} \sum_{t \in [t_1, t_2]} b^\ell_{ft} + |A(\ell-1, t_1, t_2, p)| \leq$$
$$\sum_{f \in F_p \cap F(\ell-1)} \sum_{t \in [t_1, t_2]} b^{\ell-1}_{ft} + 10c_p,$$

which equals the RHS of (10). □

We now establish a bound on the total "extra" load in any interval for the final assignment. Recall that b^* is the final, integral assignment derived from the iterative procedure above.

Lemma 3.6. *For any interval $[t_1, t_2]$ and port p,*

$$\sum_{e \in F_p} \sum_{t \in [t_1, t_2]} b^*_{et} \leq c_p(t_2 - t_1) + 10c_p \log n.$$

Proof. We fix the interval $[t_1, t_2]$ and port p. By construction of b^*, we need only to show that, for all ℓ

$$\mathrm{Vol}(p, \ell, t_1, t_2) \leq c_p(t_1 - t_2) + 10(\ell+1)c_p. \tag{12}$$

We prove inequality 12 by induction on ℓ. For $\ell = 0$, we have

$$\mathrm{Vol}(p, 0, t_1, t_2) = \sum_e \sum_{t \in [t_1, t_2]} b^0_{et} \leq c_p(t_1 - t_2) + 4c_p$$

by Constraint (6). So

$$\mathrm{Vol}(p, \ell+1, t_1, t_2) \leq \mathrm{Vol}(p, \ell, t_1, t_2) + 10c_p$$
$$\leq c_p(t_1 - t_2) + 10(\ell+2)c_p$$

by Lemma 3.5 and induction. □

We now have all the necessary ingredients to prove Lemma 3.2.

Proof of Lemma 3.2. In the final solution $\{b^*_{et}\}_{e,t}$, all flows are integrally assigned. Furthermore, the cost of the final solution is at most that of an optimal solution to the initial linear program (since each iteration, we are relaxing the previous linear program). Finally, by Lemma 3.6, for any time interval $[t_1, t_2]$ and port p, the total volume of assigned flows is at most $c_p(t_2 - t_1) + O(c_p \log n)$. □

3.2 Getting a valid schedule

What we obtain from Lemma 3.2 is, unfortunately, not a valid schedule but what could be called a *pseudo-schedule*; as noted in Remark 3.3, the total amount of flow passing through a port p during a time interval I could as much as $c_p O(\log n)$ more than $c_p|I|$, as allowed by the capacity of the port. In this section we show that we can convert the pseudo-schedule given by Lemma 3.2 into a valid schedule using *resource augmentation*, i.e., assuming the algorithm is allowed more port capacity than the optimal schedule. It is immediate from Lemma 3.2 that if we augment the capacity of every port by a factor of $1+O(\log n)$, then we obtain a valid resource-augmented schedule with optimal average response time. In the following, we show that we can achieve logarithmic-approximate average response time with a small constant blowup in port capacity, for the case of unit demand flows (and arbitrary port capacities).

Theorem 1. *For any positive integer c, there exists a polynomial-time algorithm that, given a set of n unit flows over a switch, computes a $(1 + \frac{O(\log n)}{c})$-approximation for average response time unit-size flows, while incurring a blowup in capacity by a factor of $(1+c)$.*

Proof. Given a set F of flows over a switch, by Lemma 3.2, there exists a pseudo-schedule which assigns flows to time slots such that the total response time is at most the cost of an optimal solution to the initial linear program and for any given time interval $[t_1, t_2]$, and for any port p, the total volume of flows assigned to p during the interval is at most $c_p(t_2 - t_1) + O(c_p \log n)$.

Due to space constraints, we give the proof of the desired claim for unit capacities. We extend the claim to arbitrary capacities in the full paper [32].The pseudo-schedule can be regarded as a sequence $\{G_t\}_t$ of bipartite $m \times m$ graphs such that in any given interval $[t_1, t_2]$, the degree of each vertex in the combined graph $\cup_{t_1}^{t_2} G_t$ is at most $(t_2 - t_1) + c' \log n$ for some $c' > 0$. Next, we convert this sequence $\{G_t\}_t$ into a sequence of bipartite matchings $\{M_t\}_t$. To this end, we divide the timeline into consecutive intervals $I_1, I_2, ...$, each of size $h = \lceil \frac{c \log n}{c} \rceil$. Now, starting from the beginning, we schedule flows in each interval before going to the next one. Consider an interval I_j, the degree of each vertex in the combined graph G_{I_j} is at most $d = \lceil c'(1 + \frac{1}{c}) \log n \rceil$. Applying the Birkhoff-von Neumann Theorem [13], G_{I_j} can be decomposed into at most d matchings in polynomial time. By increasing the capacity (bandwidth) of each port to $1 + c$, we can execute d matchings in the next available spots (with respect to release times) in at most h time steps. Since each flow is delayed by at most $h + d = \frac{O(\log n)}{c}$ steps, the total response time of this schedule is no more than

$$OPT \ + \ n \times \frac{O(\log n)}{c} \ \leq \ OPT \times (1 + \frac{O(\log n)}{c}), \qquad (13)$$

where the inequality follows from the fact that the number of flows is lower bound on the total response time. □

4 MAXIMUM RESPONSE TIME

In this section, we consider the problem of Flow Scheduling to Minimize Maximum Response Time (FS-MRT). More formally, for a given instance $S_{m,m'}$ of FS-MRT, our goal is to find the minimum ρ such that there exists a schedule of $S_{m,m'}$ with maximum response time ρ. Section 4.1 establishes that solving FS-MRT is **NP**-hard. Section 4.2 provides a tight approximation to FS-MRT via a linear programming relaxation and rounding of a more general problem.

4.1 Maximum Response Time Hardness

We establish the hardness of approximation for FS-MRT motivating our approximations in Section 4.2.

Theorem 2. *There is no polynomial time algorithm that solves Flow Scheduling to Minimize Maximum Response Time to within a factor of* 4/3 *of optimal, assuming* $P \neq NP$.

Our proof of Theorem 2 is via a reduction from the Restricted Time-table (RTT) problem, which is shown to be NP-hard in [20]. Due to space constraints, we defer the proof to the full paper [32].

4.2 Maximum Response Time Approximation

In this section, we give an approximation algorithm for Flow Scheduling to Minimize Maximum Response Time (FS-MRT). In fact, the algorithm solves a more general problem which we call *Time-Constrained Flow Scheduling*, for which there is also an easy reduction from FS-MRT. Time-Constrained Flow Scheduling is identical to FS-MRT except that flows do not have corresponding release times. Instead, each flow e, has a corresponding set of (possibly non-contiguous) *active* rounds $R(e)$ such that e can be scheduled in any round $t \in R(e)$. Observe that an instance of FS-MRT that is solvable with a maximum response time of ρ can be reduced to an instance of Time-Constrained Scheduling where $R(e) = \{t : r_e \leq t < r_e + \rho\}$ for all flows e. Therefore, the approximability of Time-Constrained Scheduling transfers directly to FS-MRT.

Linear Programming Relaxation. We provide a linear programming relaxation of Time-Constrained Flow Scheduling. Let $T = \{t \in R(e)\}_e$ be the set of rounds in which some edge can be scheduled.

$$\sum_{e \in F_p} d_e x_{e,t} \leq c_p \qquad \forall p \in P, t \in T \qquad (14)$$

$$\sum_{t \in R(e)} x_{e,t} = 1 \qquad \forall e \in F \qquad (15)$$

$$x_{e,t} \geq 0 \qquad \forall e \in F, t \in T \qquad (16)$$

The variable $x_{e,t}$ denotes the fraction of flow e scheduled in round t. Constraint (14) ensures that the total size of all edges adjacent to a port that are scheduled in a round is no more than the port's capacity. Constraint (15) ensures that all edges are scheduled.

Theorem 3. *Given an instance* $S_{m,m'}$ *of Time-Constrained Flow Scheduling, we can either determine that there is no schedule of* $S_{m,m'}$ *or produce a schedule in which the capacity of each port has been increased by* $2d_{\max} - 1$.

To prove Theorem 3, we invoke the following lemma which is proved in [36].

Lemma 4.1 (Theorem 3 in [36]). *Let* $\mathbf{A}$ *be a real-valued* $r \times s$ *matrix, let* $\mathbf{x}$ *be a real-valued* s*-vector, let* $\mathbf{b}$ *be a real-valued* r*-vector such that* $\mathbf{Ax} = \mathbf{b}$*, and let* Δ *be a positive real number such that in every column of* $\mathbf{A}$ *we have (a) the sum of the positive elements is at most* Δ *and (b) the sum of the negative elements is at least* $-\Delta$*. Then we can compute an integral* s*-vector* $\hat{\mathbf{x}}$ *such that (c) for all* i*,* $1 \leq i \leq s$*, either* $\hat{\mathbf{x}}_i = \lfloor \mathbf{x}_i \rfloor$ *or* $\hat{\mathbf{x}}_i = \lceil \mathbf{x}_i \rceil$*, and (d)* $\mathbf{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}$*, where* $\hat{\mathbf{b}}_i - \mathbf{b}_i < \Delta$ *for* $1 \leq i \leq r$*. In the case that all entries in* $\mathbf{A}$ *are integers, then a stronger bound applies:* $\hat{\mathbf{b}}_i - \lceil \mathbf{b}_i \rceil \leq \Delta - 1$.

Proof of Theorem 3. We first show that LP is a valid relaxation of Time-Constrained Flow Scheduling. We convert a schedule σ of an arbitrary instance of Time-Constrained Flow Scheduling into a feasible LP solution. For each flow e, if σ schedules e in round t, then we set $x_{e,t}$ to 1 and set it to 0 otherwise. By the port capacity restrictions on σ, we have that Constraint (14) is satisfied. Also, since all edges must be scheduled in some round of σ, we have that Constraint (15) is satisfied. Constraint (16) is trivially satisfied.

We now rewrite (14) - (16) in matrix form as $\mathbf{A}_{LP}\mathbf{x} = \mathbf{b}_{LP}$ with the use of slack variables. Then, for a given instance of Time-Constrained Flow Scheduling, we solve the program. This either outputs that there is no solution or produces a solution vector $\mathbf{x}^*$. In the former case, we use the fact that LP is a valid relaxation of Time-Constrained Flow Scheduling to determine that there is no feasible solution to the given instance. If the LP solver provides a solution vector $\mathbf{x}^*$, we rewrite $\mathbf{A}_{LP}$ and $\mathbf{b}_{LP}$ as follows. Let $d_{\max} = \max_{e \in F}\{d_e\}$. Let $\mathbf{A}$ and $\mathbf{b}$ be identical to $\mathbf{A}_{LP}$ and $\mathbf{b}_{LP}$ except that all rows corresponding to Constraint (16) have been removed,

and all values in rows corresponding to constraint (15) have been multiplied by $d_{\max}$ and made negative. Let $\Delta = d_{\max}$.

We show that $\mathbf{A}, \mathbf{x}, \mathbf{b}$, and Δ satisfy the conditions of Lemma 4.1. By construction we have $\mathbf{Ax} = \mathbf{b}$. Let the columns of $\mathbf{A}$ be indexed by $t \in [T]$. Let $pq = e \in F$ be a flow in the given problem instance. Constraint (14) entails that the coefficient d_{pq} would occur twice in a single column t: once for p and once for q. Similarly, Constraint (15) guarantees that $-2d_{\max}$ occurs once in each column t for e. So, conditions (a) and (b) are satisfied for $\mathbf{A}, \mathbf{b}, \mathbf{x}$, and Δ.

Lemma 4.1, therefore, entails the existence of matrices $\hat{\mathbf{b}}$, and $\hat{\mathbf{x}}$ that have properties (c) and (d). Property (c) entails that all values in $\hat{\mathbf{x}}$ are integral. Since all elements of $\mathbf{A}$ are integral, we have that all elements of $\hat{\mathbf{b}}$ are integral as well. Property (d) entails that the difference between the values in $\hat{\mathbf{b}}$ and $\mathbf{b}$ is strictly less than $2d_{\max}$ and so is at most $2d_{\max} - 1$.

Recall that all elements of $\mathbf{A}$ corresponding to Constraint (15) have been multiplied by $2d_{\max}$. Therefore, by dividing these values by $2d_{\max}$, we get a matrix $\mathbf{b}'$ such that the difference in values $\mathbf{b}'$ and $\mathbf{b}$ corresponding to Constraint (15) are strictly less than 1. Since all values are integral, this entail that the difference is 0. Therefore, the schedule given by $\hat{\mathbf{x}}$ satisfies Constraint (15), and all values corresponding to Constraint (14) are off by at most $2d_{\max} - 1$. Therefore, if we increase the capacity of each port by $2d_{\max} - 1$, we can feasibly schedule all edges in their active rounds. □

5 ONLINE FLOW SCHEDULING

We next consider a natural online version of flow scheduling, in which the sequence of flow requests is not available in advance; the scheduler learns about a request only at the request's release time. We use the standard framework of competitive analysis, and present some preliminary theoretical results in Section 5.1, and experimental results in Section 5.2.

5.1 Preliminary Theoretical Results

In this section, we establish several preliminary results for flow scheduling in the online setting. We first describe two lower bounds on the quality of any online approximation for both the average response time and maximum response time objectives. We then provide an online approximation for maximum response time that uses our offline algorithm, described above, as a subroutine. Due to space constraints, we defer all proofs to the full paper [32].

The following lemma shows that there is no online algorithm with a bounded competitive ratio for average response time.

Lemma 5.1 ([41]). *For any M, there is an instance I of flow scheduling such that the average response time of the schedule produced by any online algorithm on I is at least M times the average response time of the optimal schedule of I.*

The following lemma establishes a lower bound for maximum response time, using an argument similar to [33].

Lemma 5.2. *There is an instance I of flow scheduling such that the maximum response time of the schedule produced by any online algorithm on I is at least $3/2$ times the optimal for I.*

Lemma 5.3. *There is an online algorithm, which computes a schedule for any given instance I, with maximum response time at most double that of the optimal schedule of I, and where the capacity of each port p has been increased to $2(c_p + 2d_{\max} - 1)$ where $d_{\max} = \max_e\{d_e\}$.*

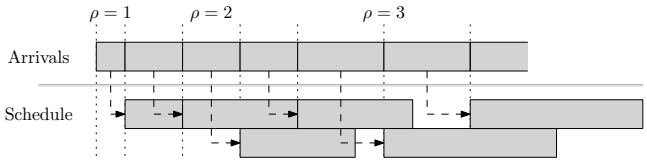

Figure 3: Gray boxes represent batches of flows. At each interval of the guessed max response time ρ, those flows in the previous batch are scheduled starting in the current round. Note that at most two boxes ever overlap.

Our online algorithm $\mathcal{A}_{MRT}$ is depicted in Figure 3. We informally define it here. In each round t, check if t is an integral value of the guessed maximum response time ρ. If so, use the offline algorithm to check if all flows which arrived in the previous ρ rounds can be scheduled with maximum response time ρ. If so, schedule them according to the offline algorithm starting in round t. Otherwise, increase the guessed ρ by one.

5.2 Experimental Results

In this section, we describe experiments conducted to evaluate the practical performance of several natural heuristics in on-line scheduling flows over a switch. In these experiments, we measure the average ($\frac{1}{n}\sum \rho_i$) and maximum ($\max \rho_i$) response times of flows. As noted before, the latter objective is predictive of the quality of service (QoS) as perceived by the user: by minimizing the maximum response time, we ensure that no job takes too long to complete. One must keep in mind, however, that optimizing for maximum response time may come at the cost of increased average response time which becomes more relevant when users submit batch jobs.

In the case of average response time, we compare the performance of these heuristics to the optimal value of the linear program (1)-(4) presented in §4.2. On the other hand, in the case of maximum response time, we compare the performance of these heuristics to the optimal value of the linear program (14)-(16) presented in §3.1. Since these LPs give lower bounds on the optimal values of any schedule, they provides us with bases for evaluating the heuristics.

5.2.1 Methodology. Packet-level simulators, such as ns2, are not suitable for flow simulation due to the large number of packets generated by each flow which makes the model infeasible. Hence, we have developed an in-house simulator for online flow scheduling of flows over a non-blocking switch.

Specifically, we use a 150×150 switch with unit port capacities. This switch models a 3000-machine cluster with 150 racks and a total bisection bandwidth of 300Gbps. Thus, each port has a capacity of 1Gbps or 128MBps. Moreover, by setting each time unit to be 1/128 second, each port has a capacity of 1MB per time unit.

Our simulator maintains a $(150, 150)$ bipartite graph G_t throughout the simulation, where t denotes the time step. The edges in G_t consist of those edges (flows) released at time t plus the ones remaining from previous steps. In other words $E(G_t)$ is the set of released edges waiting to be scheduled. Any heuristic can be

plugged in to extract a bipartite matching $M_t \subseteq E(G_t)$. Edges in M_t are assigned to run in time window t to $t+1$. Note that the edges waiting at a particular port form an open queue in the sense that any edge can be selected to run (as opposed to the edge at the front being the only available one).

In each instance of the experiment, flows are generated randomly controlled by two parameters M the average number of flows released per time unit, and T the number of steps during which the flows are generated. More precisely, for each time unit $t = 0, .., T-1$, a Poisson distribution of mean M is used to generate flows released at time t. For each such flow, an input port and an output port is selected uniformly at random. Note that $M = 150$ means that at each port, on average, there is one new flow per time step. Similarly, the average number of new flows per port is 2 and 4 for $M = 300$ and $M = 600$ respectively.

In our experiments, we compare the following three heuristics.

- **MaxCard**: at every step a matching of maximum cardinality is extracted from G_t. This heuristic is guaranteed to keep the largest number of ports busy during each step. We expect a good performance for $\frac{1}{n}\sum \rho_i$ since port utilization is kept at its max, but not for $\max \rho_i$ since it does not distinguish between edges.
- **MinRTime**: at every step t, each edge e gets assigned a weight equal to $t - r_e$, where r_e is the edge's (flow's) release time. Next, a matching of maximum weight is extracted from G_t, where the weight of an edge is the length of time since its release. We expect a good performance for $\max \rho_i$ since the longer an edge has been waiting the higher is its priority. On the other hand, $\frac{1}{n}\sum \rho_i$ may be high due to sub-optimal port utilization.
- **MaxWeight**: at every step, each edge gets assigned a weight equal to the sum of queue sizes at its two endpoints. In other words, the weight of an edge is the number of edges incident to its endpoints. Next, a matching of maximum weight is extracted from G_t. Note that the queue size at a port p is the number of released but unscheduled edges having p as an endpoint. We expect this heuristic to perform well for both objectives.

Simulations are performed for various values of M and T. Specifically, we fix $M \in \{50, 100, 150, 300, 600\}$ and run the simulator for $T \in \{10, 12, 14, 16, 18, 20, 40, 60, 80, 100\}$. Each result is the average of 10 tries. The linear programs are solved only for $T \in \{10, 12, 14, 16, 18, 20\}$ to avoid prohibitively long execution times: even for $M = 600$, and $T = 20$, each run takes more than 3 hours on an Intel Core-i7 6700HQ machine with 16GB of RAM.

5.2.2 Implementation. We implemented the simulator and its tools in C++. We use Lemon 1.3.1 library for various graph algorithms such as traversals and matchings. The `default_random_engine` was used for the distributions. The linear program is modelled and solved using Gurobi 8.1. In the case of maximum response time, we used a binary-search scheme with the linear program in (14)-(16) for finding the minimum feasible response time. The starting point of the binary search is set to the best of the three heuristics.

5.2.3 Performance. Figure 4, on page 9, shows our findings for average response time. The results are compared against the optimal value of the linear program (1)-(4) which provides a lower bound on the optimal average response time. As predicted, overall, MaxWeight and MinRTime are the best and the worst heuristic respectively. However, as the average number of incoming flows M (and hence the congestion) grows, they start to perform very similarly. Curiously, in every scenario, the performance of the the heuristics is within a factor 2 of the linear program. Moreover, the gap seems to close for larger values of M.

Figure 5, on page 10, shows the results for maximum response time. Again, the findings confirm our initial intuition. In particular, MinRTime has consistently the best performance (it almost matches the LP lower bound in some cases). On the other hand, MaxWeight is the worst of the three. Again, all heuristics are always within a factor 2.5 of the LP. Unlike the average case, the gap between the heuristics seems to grow with M.

Our conclusion is that MaxCard and MinRTime are good choices for minimizing average response time and minimizing maximum response time respectively. MaxWeight takes the middle ground and is thus the best choice (among the three) when it is desirable to keep both average and maximum response times low.

6 OPEN PROBLEMS

We have presented approximation algorithms for minimizing response time metrics in flow scheduling over a switch network. Our work offers a number of directions for future research.

Improved approximation ratios. For average response time, our algorithm achieves an $O(\log n/c)$-approximation while incurring a $1 + c$ augmentation in capacity, for any given positive integer c. While resource augmentation is necessary for any competitive algorithm in the online setting, does an offline approximation (with say a polylogarithmic approximation ratio) need resource augmentation? For maximum response time, our algorithm achieves the optimal objective while incurring an increase in capacity by the size of the maximum demand. An important open problem is to determine whether we need resource augmentation to obtain any reasonable approximation algorithm for maximum response time.

Competitive online algorithms. Our work on online algorithms is preliminary and provides some guidance on heuristics one can use for response-time related metrics. While we have given a constant-competitive algorithm for maximum response time with constant-factor resource augmentation, the situation with no resource augmentation is unclear. We plan to conduct a more thorough investigation of online algorithms – both theoretical and experimental.

Generalizations and beyond worst-case analysis. Our work has focused on scheduling flows on switch networks. We would like to extend our research to a broader class of datacenter networks (e.g., trees, fat-trees, more general networks) and more general types of flows (e.g., co-flows). We would also like to study the problems posed in a model that includes some information about the distribution of input instances that may be available from practical applications. This would be especially useful for the average response time objective, for which no non-trivial competitive ratio is achievable without resource augmentation.

ACKNOWLEDGMENTS

This work was partially supported by NSF grant CCF-1909363. We would like to thank Janardhan Kulkarni for the many discussions on online flow scheduling, and for generously allowing us to include his proof of Lemma 5.1.

A FIGURES SHOWING EXPERIMENTAL RESULTS

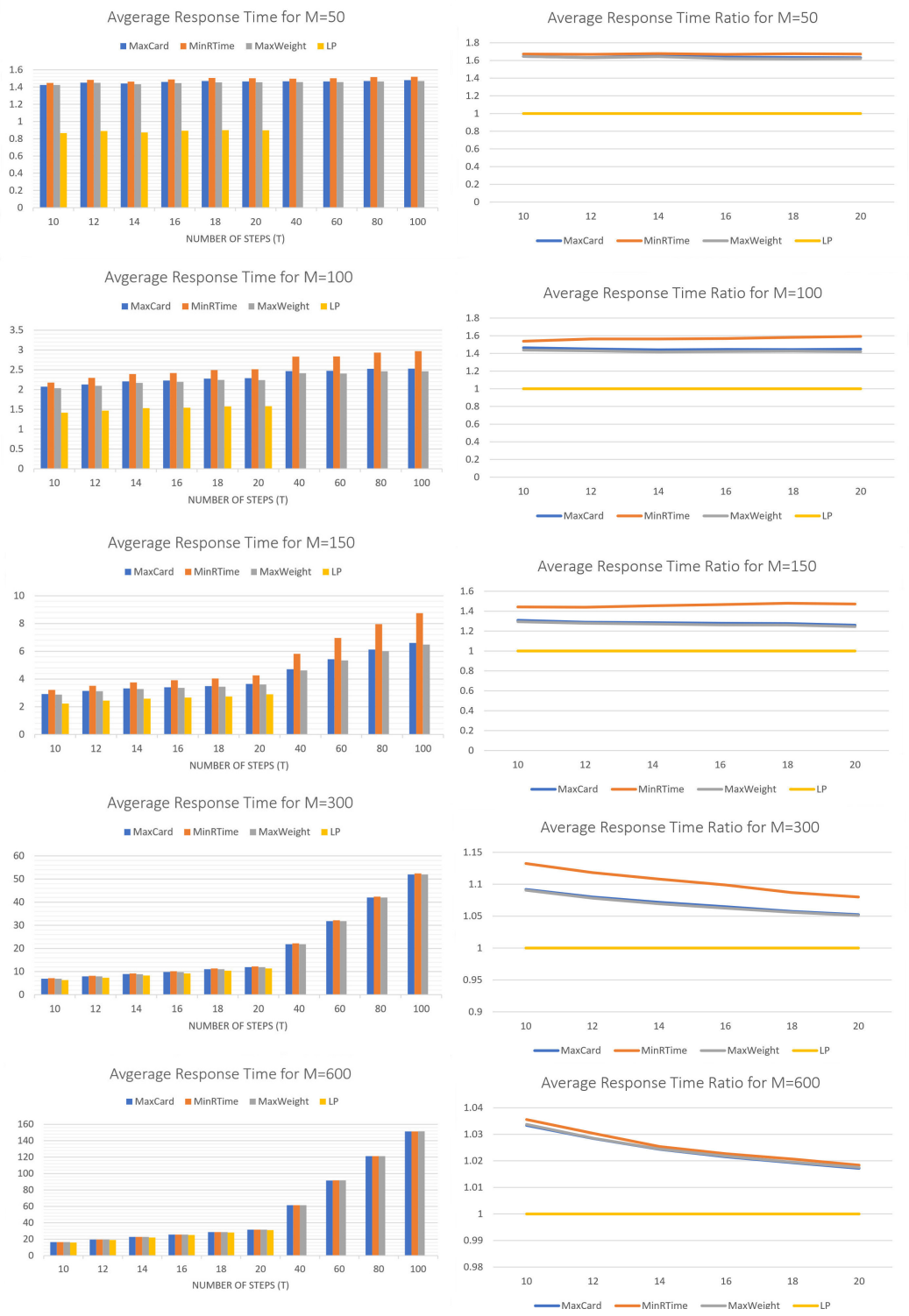

Figure 4: Average response time results.

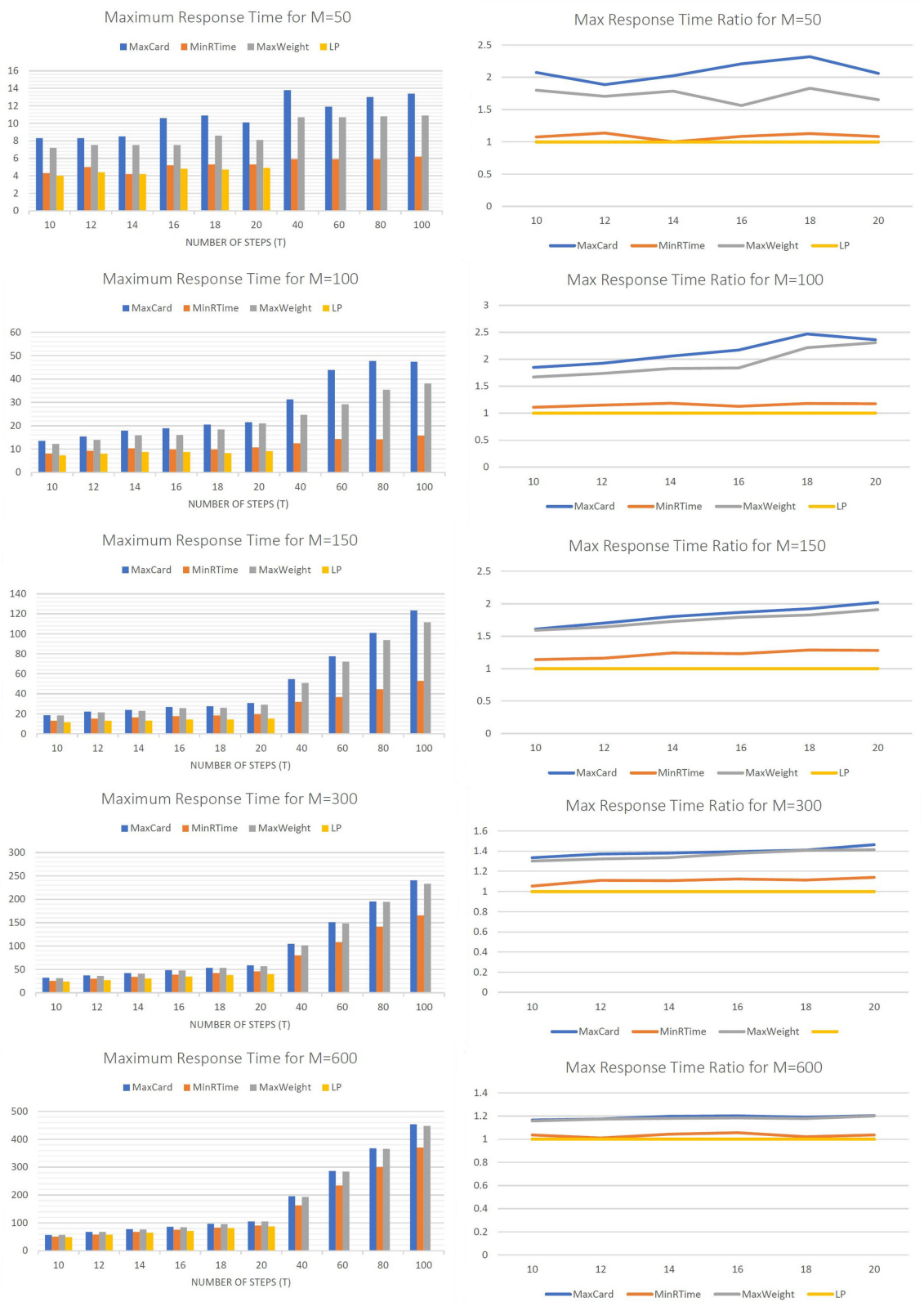

Figure 5: Maximum response time results.

REFERENCES

[1] S. Ahmadi, S. Khuller, M. Purohit, and S. Yang. On scheduling coflows. In *IPCO*, 2017.

[2] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. *SIGCOMM Comput. Commun. Rev.*, 43(4):435–446, August 2013.

[3] C. Ambühl and M. Mastrolilli. On-line scheduling to minimize max flow time: an optimal preemptive algorithm. *Operations Research Letters*, 33(6):597 – 602, 2005.

[4] K. R. Baker. *Introduction to Sequencing and Scheduling*. Wiley, New York, 1974.

[5] Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. Towards predictable datacenter networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):242–253, August 2011.

[6] N. Bansal. *Algorithms for Flow Time Scheduling*. PhD thesis, School of Computer Science, Carnegie Mellon University, December 2003.

[7] N. Bansal and H. Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1238–1244, 2009.

[8] N. Bansal and J. Kulkarni. Minimizing flow-time on unrelated machines. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 851–860, New York, NY, USA, 2015. ACM.

[9] J. Batra, N. Garg, and A. Kumar. Constant factor approximation algorithm for weighted flow time on a single machine in pseudo-polynomial time. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 778–789, Oct 2018.

[10] M. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 270–279, January 1998.

[11] K. Benzekki, A. El Fergougui, and Abdelbaki Elbelrhiti E. Software-defined networking (sdn): a survey. *Security and Communication Networks*, 9(18):5803–5833, 2016.

[12] E. W. Biersack, B. Schroeder, and G. Urvoy-Keller. Scheduling in practice. *SIGMETRICS Performance Evaluation Review*, 34(4):21–28, 2007.

[13] D. Birkhoff. Tres observaciones sobre el algebra lineal. *Universidad Nacional de Tucuman Revista , Serie A*, 5:147–151, 1946.

[14] C. Chekuri, S. Khanna, and A. Zhu. Algorithms for minimizing weighted flow time. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 84–93, 2001.

[15] M. Chowdhury, S. Khuller, M. Purohit, S. Yang, and J. You. Near optimal coflow scheduling in networks. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 123–134. ACM, 2019.

[16] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. *SIGCOMM, Comput. Commun. Rev.*, 44(4):443–454, August 2014.

[17] Michael Dinitz and Ben Moseley. Scheduling for weighted flow and completion times in reconfigurable networks. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020. Forthcoming.

[18] J. Du, J. Y.-T. Leung, and G. H. Young. Minimizing mean flow time with release time constraint. *Theoretical Computer Science*, 75:347–355, 1990.

[19] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *SIGCOMM Comput. Commun. Rev.*, 36(1):59–62, January 2006.

[20] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5:691–703, 12 1976.

[21] U. Feige, Janardhan Kulkarni, and Shi Li. A polynomial time constant approximation for minimizing total weighted flow-time. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1585–1595. SIAM, 2019.

[22] N. Garg and A. Kumar. Better algorithms for minimizing average flow-time on related machines. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 181–190, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[23] N. Garg and A. Kumar. Minimizing average flow-time: Upper and lower bounds. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07, pages 603–613, Washington, DC, USA, 2007. IEEE Computer Society.

[24] N. Garg, A. Kumar, and V. N. Muralidhara. Minimizing total flow-time: The unrelated case. In Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, editors, *Algorithms and Computation*, pages 424–435, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[25] P. Giaccone, B. Prabhakar, and D. Shah. Randomized scheduling algorithms for high-aggregate bandwidth switches. *IEEE Journal on Selected Areas in Communications*, 21(4):546–559, 2003.

[26] K. Giaro, M. Kubale, M. Malafiejski, and K. Piwakowski. Chromatic scheduling of dedicated 2-processor uet tasks to minimize mean flow time. In *1999 7th IEEE International Conference on Emerging Technologies and Factory Automation. Proceedings ETFA '99*, volume 1, pages 343–347 vol.1, Oct 1999.

[27] L. Gong, P. Tune, L. Liu, S. Yang, and J. Xu. Queue-proportional sampling: A better approach to crossbar scheduling for input-queued switches. In *Proceedings of the ACM on Measurement and Analysis of Computing Systems (SIGMETRICS)*, 2017.

[28] P. Goransson and C. Black. *Software Defined Networks: A Comprehensive Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2014.

[29] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[30] I. Grosof, Z. Scully, and M. Harchol-Balter. SRPT for multiserver systems. *Perform. Eval.*, 127-128:154–175, 2018.

[31] H. Jahanjou, E. Kantor, and R. Rajaraman. Asymptotically optimal approximation algorithms for coflow scheduling. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '17, pages 45–54, New York, NY, USA, 2017. ACM.

[32] H. Jahanjou, R. Rajaraman, and D. Stalfa. Scheduling flows on a switch to optimize response times. arXiv: `https://arxiv.org/abs/2005.09724`, May 2020.

[33] S. Jia, X. Jin, G. Ghasemiesfeh, J. Ding, and J. Gao. Competitive analysis for online scheduling in software-defined optical wan. In *Proc. of the IEEE INFOCOM Conference*, pages 1–9, 05 2017.

[34] B. Kalyanasundaram and K. Pruhs. Minimizing flow time nonclairvoyantly. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science*, pages 345–352, 1997.

[35] N. Kang, Z. Liu, J. Rexford, and D. Walker. Optimizing the "one big switch" abstraction in software-defined networks. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 13–24, New York, NY, USA, 2013. ACM.

[36] R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. V. Vazirani, and V. V. Vazirani. Global wire routing in two-dimensional arrays, 1987.

[37] H. Kellerer, T. Tautenhahn, and G. J. Woeginger. Approximability and nonapproximability results for minimizing total flow time on a single machine. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 418–426, May 1996.

[38] S. Khuller and M. Purohit. Improved approximation algorithms for scheduling co-flows. In *SPAA*, 2016. Brief Announcement.

[39] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[40] M. Kubale and H. Krawczyk. An approximation algorithm for diagnostic test scheduling in multicomputer systems. *IEEE Transactions on Computers*, 34:869–872, 09 1985.

[41] J. Kulkarni. Personal communication.

[42] E. L. Lawler and J. Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *J. ACM*, 25(4):612–619, October 1978.

[43] S. Leonardi and D. Raz. Approximating total flow time on parallel machines. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 110–119, May 1997.

[44] L. Luo, K. Foerster, H. Yu, and S. Schmid. Splitcast: Optimizing multicast flows in reconfigurable datacenter networks. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020. Forthcoming.

[45] M. Mastrolilli. *Scheduling to Minimize Max Flow Time: Offline and Online Algorithms*, pages 49–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[46] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 39–50, New York, NY, USA, 2009. ACM.

[47] Z. Qiu, C. Stein, and Y. Zhong. Minimizing the total weighted completion time of coflows in datacenter networks. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 294–303, New York, NY, USA, 2015. ACM.

[48] Z. Qiu, C. Stein, and Y. Zhong. Minimizing the total weighted completion time of coflows in datacenter networks. In *SPAA*, pages 294–303, 2015.

[49] Mehrnoosh Shafiee and Javad Ghaderi. An improved bound for minimizing the total weighted completion time of coflows in datacenters. *IEEE/ACM Trans. Netw.*, 26(4):1674–1687, 2018.

[50] D. Shah and J. Shin. Randomized scheduling algorithm for queueing networks. *The Annals of Applied Probability*, 22(1):128–171, 2012.

[51] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang. Rapier: Integrating routing and scheduling for coflow-aware data center networks. In *INFOCOM*, pages 424–432, 2015.