Competitive Analysis of Constrained Queueing Systems

Sungjin Im[*1], **Janardhan Kulkarni**[†2], **and Kamesh Munagala**[‡3]

1 **Electrical Engineering and Computer Science, University of California, Merced, CA, USA**
sim3@ucmerced.edu

2 **Microsoft Research, Redmond, WA, USA**
jakul@microsoft.com

3 **Department of Computer Science, Duke University, Durham, NC, USA**
kamesh@cs.duke.edu

Abstract

We consider the classical problem of constrained queueing (or switched networks): There is a set of N queues to which unit sized packets arrive. The queues are interdependent, so that at any time step, only a subset of the queues can be activated. One packet from each activated queue can be transmitted, and leaves the system. The set of feasible subsets that can be activated, denoted $\mathcal{S}$, is downward closed and is known in advance. The goal is to find a scheduling policy that minimizes average delay (or flow time) of the packets. The constrained queueing problem models several practical settings including packet transmission in wireless networks and scheduling cross-bar switches.

In this paper, we study this problem using the the competitive analysis: The packet arrivals can be adversarial and the scheduling policy only uses information about packets currently queued in the system. We present an online algorithm, that for any $\epsilon > 0$, has average flow time at most $O\left(\frac{R^2}{\epsilon^3}OPT + NR\right)$ when given $(1+\epsilon)$ speed, *i.e.*, the ability to schedule $(1+\epsilon)$ packets on average per time step. Here, R is the maximum number of queues that can be simultaneously scheduled, and OPT is the average flow time of the optimal policy. This asymptotic competitive ratio $O(\frac{R^2}{\epsilon^3})$ improves upon the previous $O(\frac{N}{\epsilon^2})$ which was obtained in the context of multi-dimensional scheduling [6]. In the full general model where N can be exponentially larger than R, this is an exponential improvement. The algorithm presented in this paper is based on Makespan estimates which is very different from that in [6], a variation of the Max-Weight algorithm. Further, our policy is myopic, meaning that scheduling decisions at any step are based only on the current composition of the queues. We finally show that speed augmentation is necessary to achieve any bounded competitive ratio.

1998 ACM Subject Classification C.2.1 Network Architecture and Design: Packet-switching networks, F.2.2 Nonnumerical Algorithms and Problems: Sequencing and scheduling

Keywords and phrases Online scheduling, Average flow time, Switch network, Adversarial

Digital Object Identifier 10.4230/LIPIcs.ICALP.2016.143

* S. Im was supported in part by NSF CCF-1409130.
† A part of this work was done while J. Kulkarni was at Duke University.
‡ K. Munagala was supported by NSF grants CCF-1408784, IIS-1447554, and CCF-1348696.

43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016).
Editors: Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi;
Article No. 143; pp. 143:1–143:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Stochastic processing networks [16] capture a wide range of resource allocation scenarios. In the general setting, there are a set of N queues, $Q_1, Q_2, \ldots, Q_N$. Packets of unit size arrive at these queues according to some arrival process. Time is discrete, and at each time step, at most one packet from each queue can be served (or scheduled), each packet requiring unit amount of service. The queues use shared resources for scheduling, leading to constraints on the subsets of queues that can be simultaneously scheduled. Let $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ denote the set of feasible subsets of queues that can be simultaneously scheduled at any time slot; we assume the system $\mathcal{S}$ is downward closed so that if $S \in \mathcal{S}$, then any subset of S also belongs to $\mathcal{S}$. [1] For such a system let $R = \max_{S \in \mathcal{S}} |S|$ denote the maximum number of packets that can be simultaneously be scheduled.

Such a general model was first formulated in the seminal work of Tassiulas and Ephremides [16]. We first discuss some applications. First consider a $n \times n$ packet switch with crossbar architecture [11, 12, 4]. Packets arrive at each of the n input ports, with each packet specifying an output port to which it must be scheduled. The crossbar architecture enforces the constraint that at most one packet can be scheduled per input or output port per time slot. There are therefore $N = n^2$ queues $\{Q_{ij}\}$, one per input-output pair (i, j). Queue Q_{ij} queues packets arriving at input i for output j. A feasible subset of queues in $\mathcal{S}$ is a matching of input to output ports, so that $\mathcal{S}$ is the set of all matchings between inputs and outputs. A generalization of this setting arises in wireless networking, where there is a queue Q_{ij} between every pair of locations i and j that are within communication radius; a feasible subset of queues is any set of pairs (i, j) that can simultaneously communicate without interference.

A similar problem arises in *multicast* switch scheduling [14], where packets arriving at an input port of a crossbar switch need to be simultaneously transmitted to multiple output ports.

We now present the formal model. Packets arrive into the queues according to an adversarially chosen process during a *finite* time interval $[0, T]$. We do not constrain the number of packets that can arrive into any queue at any time step. We assume each packet suffers a delay of at least one time slot. The main objective considered in this setting is the average flow time. Let n_t denote the number of packets awaiting service at time t; this is the queue size at time t. Let n denote the total (finite) number of packets arriving in the system. Let T' be a sufficiently large time, for example, $T' = T + n$, by which any 'non-idle' algorithm can complete all packets.

$$\text{Average Flow Time } = \frac{1}{n} \sum_{t=0}^{T'} n_t \,. \tag{1}$$

A scheduling policy is an algorithm that decides (at every time slot) the set of queues to schedule. In an *online* policy, this decision is based on the number of packets in each queue, but not on the knowledge of future arrivals. Policies whose decisions are based only on queue sizes (or current system state) are termed *Markovian* in queueing theory. In this paper, we will call such policies *myopic*. Our focus will be on designing myopic policies.

Integral vs. Fractional Schedules. We make a distinction between *integral* and *fractional* schedules. The definition above assumes time is slotted into unit length slots, and we execute

[1] Such a set system is often called an independence system, and a set $S \in \mathcal{S}$ is said to be independent.

one schedule $\sigma \in \mathcal{S}$ per time slot. Using the notation in [15], let

$$\langle \mathcal{S} \rangle = \left\{ \sum_{\sigma \in \mathcal{S}} \alpha_\sigma \sigma \;\; | \;\; \sum_{\sigma \in \mathcal{S}} \alpha_\sigma \leq 1, \;\; \alpha_\sigma \geq 0 \;\; \forall \sigma \in \mathcal{S} \right\} \tag{2}$$

For each $S \in \mathcal{S}$, we let $\sigma(S)$ denote a binary vector in $2^{[N]}$ that encodes the subset of queues activated in S, and for notational convenience, we let $\sigma \in \mathcal{S}$, omitting S. The set $\langle \mathcal{S} \rangle$ therefore represents collections of schedules, each executed to a fraction, so that the total fraction is one unit. A fractional schedule executes one schedule from $\langle \mathcal{S} \rangle$ per time slot. In other words, a fractional schedule assumes time is continuous, and that packets are divisible. When α amount of schedule σ is executed, α amount of the corresponding packets leave the system. Note that an integral schedule is a special case of a fractional schedule where $\alpha_\sigma \in \{0, 1\}$ for all $\sigma \in \mathcal{S}$.

In fractional schedule, an alternative but equivalent definition of average flow time will be useful: we assume that packets are processed in first-in-first-out manner in each queue, and a packet j's completion time C_j is the earliest time when the whole packet leaves the queue. An individual packet's flow time is defined as its completion time C_j minus its arrival time at the queue, and the average flow time is the total flow time of all packets divided by the total number of packets. It is easy to see this definition of average flow time coincides with the above definition (1).

Let OPT denote the average flow time of the fractional policy that makes optimal scheduling decisions with knowledge of all future arrivals. This is a valid relaxation since allowing the optimal scheduler/the adversary to be fractional can only decrease its average flow time. Our goal is to design an online *integral* scheduling policy whose average flow time, for all input sequences, is at most $c \times OPT$. We call c as the *competitive ratio* of the policy. We will achieve this by first developing a fractional online schedule and then converting it into an integral schedule online.

1.1 Resource Augmentation Analysis and Scalable Policies

A simple example shows that no online algorithm can have bounded competitive ratio. To see this, consider a simple 2×2 crossbar switch with two input ports and two output ports. For some large L, at time $t = 0$, L packets arrive at both inputs destined for output 1. From time $t = L$ to $t = 2L - 1$, where one packet arrives per time step destined for output 2; the algorithm is not told which input port these packets arrive at. (Suppose these arrive at input 1.) Beyond this time, there is one packet per time slot arriving at input 1 for output 1 and input 2 for output 2. The optimal algorithm which knows the future, spends the first L steps serving the L packets queued at input 1. From time $t = L$ to $t = 2L - 1$, it schedules the matching of input port 1 to output 2 and input port 2 to output 1. Beyond this time, it has no queued packets, and can schedule each incoming packet in the same time slot. In the limit as $T \to \infty$, $OPT = 1$. Any online algorithm has to guess the behavior of OPT between time $t = L$ and $t = 2L - 1$. Suppose it keeps $x \leq L$ packets on input port 1 at time $t = L$, then it has x packets queued for all time $t \geq L$, leading to average flow time of x. This shows an unbounded competitive ratio even for randomized strategies.

The above example is typical of several scheduling problems when the input is adversarial [8, 7, 1], and motivates the need for *resource augmentation analysis*, a framework introduced by Kalyanasundaram and Pruhs [8]. We say that an online algorithm is given $(1 + \epsilon)$ speed for any $\epsilon > 0$, if in the integral case, the algorithm is allowed to perform an extra round of scheduling every $\lfloor \frac{1}{\epsilon} \rfloor$ time steps; or is allowed to execute schedules at rate

$(1+\epsilon)$ in the fractional case. Equivalently, the online algorithm, given no extra speed, is compared against the optimal scheduler with $(1-\epsilon)$-speed. This view is more natural since we simply constrain the space of feasible schedules used by the optimal policy to have $(1-\epsilon)$ rate, *i.e.*, use $\sum_{\sigma\in\mathcal{S}} \alpha_\sigma \leq 1-\epsilon$ in (2), and give a speed 1 to the online algorithm. This also highlights the fact that resource augmentation is purely an analysis tool – the algorithms we design are oblivious to such resource augmentation, using only the current queue sizes to make scheduling decisions. We call an algorithm *scalable* if for every $\epsilon > 0$, the competitive ratio of the algorithm is $f\left(N, \frac{1}{\epsilon}\right)$ for some function f.

We note that even with resource augmentation, the design of scalable scheduling policies is non-trivial. Consider the *stochastic* setting where packets arrive in each queue according to a Bernoulli process with known rate. This arguably benign setting is widely studied in networks [16, 12, 13, 15]. Consider an 3×3 crossbar switch, and the natural policy that maximizes instantaneous throughput: Consider the pairs of inputs and outputs such that there is at least one packet queued for that input/output pair. Find a maximum size matching of these input/output pairs and schedule this matching, choosing a matching at random if there are multiple maximum matchings. It is known [12] that for Bernoulli traffic where the arrival rates for all inputs and outputs are strictly smaller than one packet per time step, the expected queue size (and hence expected flow time) of this policy can be unbounded. However, for such input, there are other policies with bounded expected queue size (see below for more details). This directly shows that there exists some $\epsilon > 0$ for which the maximum throughput policy, even with $(1+\epsilon)$-speed has unbounded competitive ratio, and is hence not scalable.

1.2 Our Results

Our main result is a myopic scheduling policy for *arbitrary* scheduling constraints $\mathcal{S}$ whose asymptotic competitive ratio only depends on R. Recall that $R = \max_{S\in\mathcal{S}} |S|$ is the maximum number of packets that can be feasibly scheduled any time slot. We first develop a fractional online scheduler, and then convert it into a feasible integral policy.

▶ **Theorem 1.** *There is an online fractional policy, which we name* Sampling Independent Set from Min-Makespan (SISM)*, that is myopic, and is $O(\frac{R^2}{\epsilon^3}+N)$-competitive when compared to the optimal scheduler with $(1-\epsilon)$-speed. More precisely, the online policy has an average flow time $O(\frac{R^2}{\epsilon^3}\mathrm{OPT}+N)$ where* OPT *is the optimal algorithm's average flow time. If the online algorithm does not have to be myopic, the competitive ratio can be improved to $O(\frac{R^2}{\epsilon^4})$.*

We note that if OPT is large, then the competitive ratio depends only on $R = \max_{S\in\mathcal{S}} |S|$, which could be much smaller than N, and is a more robust measure of the complexity of the scheduling constraints $\mathcal{S}$. On the other hand, N can be made larger by simply splitting one queue into several "virtual queues". (A related notion to R, called *rank*, is used in [15] as a measure of the complexity of $\mathcal{S}$.) In fact, N can be exponentially larger than R. As mentioned before, the best known competitive ratio prior to this work was $O(\frac{N}{\epsilon^2})$ [6] which was found in the context of multi-dimensional scheduling. Further, our algorithm is very different from [6] which can't avoid a linear dependency on N; this previous work will be discussed in detail in Section 1.4.

Fractional Scheduling Policy. Our scheduling policy SISM is quite natural. At time step t, let n_{it} denote the (possibly fractional) number of packets queued at queue Q_i. At time t,

assuming no more packets arrive in the future, solve the makespan minimization problem: Find a fractional cover $\{\lambda(S)\}_{S\in\mathcal{S}}$ such that if these schedules are executed (in arbitrary order), then all queues Q_i are emptied, i.e. $\sum_{S:i\in S}\lambda(S) = n_{it}$. We process each independence set S at a rate in proportion to its weight $\{\lambda(S)\}$. An equivalent view which explains the algorithm's name is that each cover (independent set) S exists in $\lambda(S)$ copies, and we sample a cover uniformly, and process it by an infinitesimal amount.

We first show that our algorithm is $O(R^2/\epsilon^3)$-competitive for the fractional average flow time. Roughly speaking, the algorithm incurs a cost equal to the total remaining size of packets for the fractional flow – the discrepancy between the total remaining size and the total number of packets is at most N, hence the N appears in the competitive ratio. Alternatively, by using a standard method of converting fractional flow to integral flow, we can get a competitive ratio with no dependency on N, but the resulting algorithm is no longer myopic.

We note that the SISM has running time poly($|\mathcal{S}|$). However, for several applications, $|\mathcal{S}|$ itself may be exponential in N, the number of queues, leading to an exponential time algorithm. This dependence is unavoidable given the generality of the problem statement: For myopic policies that can not store past scheduling decisions, it is easy to check that maximizing instantaneous throughput encodes computing independent set of a general graph, a problem whose solution cannot be approximated to any constant factor unless $P = NP$. Assuming $P \neq NP$, this implies there is no myopic policy with poly(N) computation per time slot, which has bounded competitive ratio even with constant speed. Nevertheless, for the case of scheduling crossbar switches, our policy has poly(N) computation time per step using the Birkhoff-von Neumann theorem (see for instance [13]). Furthermore, as mentioned above, for settings such as scheduling jobs in data centers where the number of resources in contention is constant, we can assume $\mathcal{S}$ to have constant size.

Integral Policy. As mentioned above, we convert our fractional policy SISM into an integral scheduler on the fly using an emulation technique from [15]. For comparison, best previous policies [4] required speed of at least 2, and were specific to crossbar switches.

▶ **Theorem 2.** *There is an online integral policy that is myopic, and is $O(\frac{R^2}{\epsilon^3} + NR)$-competitive when compared to the optimal scheduler with $(1-\epsilon)$-speed. More precisely, the online policy has an average flow time $O(\frac{R^2}{\epsilon^3}\mathrm{OPT} + NR)$ where OPT is the optimal scheduler's average flow time.*

1.3 Technical Contributions

Our analysis proceeds via establishing amortized local competitiveness [7], a framework first introduced in [2]. Potential functions are often very useful when local competitiveness analysis fails. As one can deduce from (1), we can perform a strictly local competitiveness analysis if we can upper bound the number of packets alive in SISM's queues at any time by the analogous number for the optimal scheduler. However, this approach fails since the optimal scheduler can cleverly group packets and complete them more quickly than our algorithm. Intuitively, in the presence of an arbitrary independence set system $\mathcal{S}$, the number of queued packets can change very dynamically. In such settings, potential functions allow us to compare the online algorithm to the optimal scheduler more robustly over time.

As illustrated in [7], the standard approach to designing a potential function is to establish a rough estimate of the algorithm's flow time assuming that no more packets arrive. We use the minimum makespan needed to complete all queued packets as our estimate. A standard

way of converting this quantity into a potential function is to now replace each packet's remaining size with the *lag* that measures how much the algorithm is behind the optimal scheduler in processing that packet. This conversion makes the potential resilient to discrete events such as new packets arriving, and allows us to focus on how the potential changes as packets are continuously processed. However, the analysis is still complicated since SISM's schedule can be very different from OPT's schedule, which means the algorithm's processing may not change the potential in the right direction when needed.

As mentioned above, the potential we use is based on the minimum makespan when each packet's remaining size replaced with its lag, which can be very different from the minimum makespan schedule constructed by SISM. We therefore need to relate these two schedules, which is not at all obvious. It becomes crucial that SISM sample an independent set uniformly at random from its min-makespan schedule. This ensures two nice properties. First, it decreases the makespan of SISM uniformly with time. More importantly, due to the uniform sampling nature, it processes packets in proportion to their respective remaining sizes. Using these two properties, we show that if the optimal scheduler has very few packets left (and these are exactly the times when the algorithm needs the potential), the min-makepsan schedule for the packet lags (*i.e.*, the potential) mostly consists of independent sets where all packet's size decrease almost uniformly. Hence the potential decreases in the desired direction.

We finally note that we do not know how to use popular linear programming approaches, such as dual fitting [5] for this problem. We do note that it is relatively straightforward to obtain a $O(1)$-competitive algorithm with R-speed using these methods, and the hard part is to obtain a scalable algorithm.

1.4 Related Work

Constrained queueing systems are among the most widely studied settings in scheduling theory. Existing work falls in the realm of queueing theory, and has mostly focused on the stochastic case where packet arrivals are *i.i.d.* according to some stochastic process (most commonly Bernoulli or Poisson). In this context, the key assumption is that the arrival rates into the queues are feasible (in a certain natural sense), and the focus is on designing policies that are *stable*, meaning that the expected flow time is finite. The seminal result of [16] shows that the *maximum weight* policy is stable: This policy defines the weight of a queue as the number of packets waiting at the queue, and at each time step, finds a feasible schedule in $\mathcal{S}$ such that the sum of the weights of the queues in the schedule is maximized. When specialized to a $n \times n$ crossbar switch, this policy finds a maximum weight matching between inputs and outputs [12], where the weight of an input/output pair is the number of packets queued at that input for that output. In this context, for Bernoulli arrivals, this policy has expected queue size (or average flow time) $O(n^2)$ [15].

Most theoretical work on constrained queueing has focused on improving delay bounds in the stochastic setting. The work of [13] constructs a policy with average flow time $O(n \log n)$ for an $n \times n$ crossbar switch. Their algorithm considers a batch of L packets per input and output, finds a makespan minimizing schedule over L time steps for this batch, and runs this schedule for the next L steps. Such a policy is clearly not myopic since the policy is computed in batches; in addition, they set L carefully based on the input arrival rates, making the policy specific to the stochastic setting. Our policy Min-Makespan can be thought of as an online, myopic analog of this policy. Finally, the work of [15] presents a policy with expected flow time $O(n)$, which can be generalized to arbitrary constraints $\mathcal{S}$; their policy is based on modeling the constrained system $\mathcal{S}$ as a network of queues, and using

a proportional fairness type queueing policy called Store and Forward Allocation (SFA) [3, 10] on this queueing network. The SFA policy is a fractional scheduling policy that is specific to stochastic arrivals, and we need new ideas for the adversarial setting. The main contribution of [15] is an emulation of fractional scheduling policies by integral policies, and we use this same technique for converting our fractional schedules to integral ones.

We emphasize that the hard part of our problem is to obtain a *scalable* algorithm, *i.e.*, algorithms that have bounded competitive ratio for any ϵ extra speed, in addition to being myopic. It is relatively simple to obtain competitive algorithms that use speed that depends on $\mathcal{S}$. In particular, for scheduling a crossbar switch, any greedy maximal matching algorithm is 1-competitive with speed 2. In fact, in this same setting, the work of Chuang *et al.* [4] shows something far stronger: With speedup 2, suitably designed stable marriage algorithms are 1-competitive on *any* QoS property of delays (such as weighted flow time, l_k-norms of flow time, etc). However, these algorithms are specific to switch scheduling; furthermore, they have unbounded competitive ratio with speed less than 2, and are hence not scalable.

The constrained queueing problem (fractional version) is a special case of the polytope scheduling problem (PSP) formulated in [5]. The authors show that the proportional fairness (PF) algorithm [9] can be adapted to derive a competitive scheduling algorithm for this general setting. The objective considered is the sum of completion times of the jobs. Using the KKT conditions combined with dual fitting, the authors show that the PF algorithm is constant competitive on this objective even when jobs have arbitrary lengths and weights. The flow time objective we consider corresponds to the difference between completion time and release date of a job, and is typically a more difficult objective to optimize even in simpler settings.

In [6], PSP was reformulated with queues. In PSP-Q, the polytope is defined over queues. This constrains how much each queue can be processed. Then, any two jobs arriving into the same queue are interchangeable in the sense that one job can be replaced with the other job, preserving how much the queue is processed. In [6], it was shown that the Normalized Max-Weight algorithm is $(1+\epsilon)$-speed $O(N/\epsilon^2)$-competitive – roughly speaking, the maximum weight independence set is scheduled assuming that each queue's weight is equal to the number of jobs in the queue. The work in [6] can handle weighted jobs, but the linear dependency on N was unavoidable due to the nature of the algorithm. On the other hand, this paper can only handle packets of the same size, but the competitive ratio only depends on R.

2 Fractional Scheduling Policy: SISM

In this section, we present our fractional scheduling policy Sampling Independent Set from Min-Makespan (SISM), and analyze its performance for the average flow time objective. For convenience, we abuse the notation slightly to let $Q_i(t)$ denote the (possibly fractional) number of packets (or workload) waiting at queue Q_i. See Figure 1 for the description of the algorithm.

To compute a desired fractional cover, we can simply solve a linear programming over the variables $\{\lambda(S)\}_{S\in\mathcal{S}}$. Then, the running time will be a polynomial in $|\mathcal{S}|$. Note that this computation needs to be done only when new packets arrive since otherwise the optimal cover remains the same. As mentioned earlier, for the special case of switch scheduling, such an optimal cover can be computed in polynomial time in N by using the Birkhoff-von Neumann theorem; we can represent the current workload as a square matrix; and obtain a doubly stochastic matrix by adding dummy quantities so that the entries in each row and

Sampling Independent Set from Min-Makespan (SISM)
At each time t,
Compute a fractional cover $\{\lambda(S)\}_{S\in\mathcal{S}}$ that completes the workload $\{Q_i(t)\}_{i\in[N]}$ with the minimum makespan $L_Q(t)$.
(i.e., $L_Q(t) = \min \sum_{S\in\mathcal{S}} \lambda(S)$ s.t. $\sum_{S\in\mathcal{S}:i\in S} \lambda(S) = Q_i(t)$).
Schedule each independent set S at a rate of $\lambda(S)/L_Q(t)$.

Figure 1 The SISM Procedure.

column add up to $\max_i Q_i(t)$ and normalizing the matrix; and decompose it into permutation matrices.

Our scheduling policy has the following nice property.

▶ **Proposition 3.** *The policy* SISM *decreases each $Q_i(t)$ at a rate of $\frac{Q_i(t)}{L_Q(t)}$ at all times when no packets arrive or are completed by* SISM *or the optimal scheduler.*

The proof easily follows by viewing the policy as sampling an independence set uniformly from a multi-set of independent sets where each independence set S exists in $\lambda(S)$ copies, and observing each queue Q_i appears in exactly $Q_i(t)$ independent sets in the multi-set.

2.1 Potential Function

Throughout the analysis, we assume that the optimal scheduler (OPT) is restricted to $(1-10\epsilon)$-speed for $\epsilon \le 1/10$. Our analysis is based on a potential function. To formally define the potential function, we need more notation. Let $Q_i^*(t)$ denote the workload waiting at Q_i at time t in the optimal schedule. Define $Z_i(t) := \max\{Q_i(t) - Q_i^*(t), 0\}$ to be the algorithm's lag on queue Q_i compared to the optimal scheduler. Define $L_Z(t)$ to be the minimum makespan achievable assuming that each queue Q_i has $Z_i(t)$ workload. More precisely,

$$L_Z(t) := \min_{\lambda_Z} \sum_{S\in\mathcal{S}} \lambda_Z(S) \quad \text{s.t.} \quad \sum_{S\in\mathcal{S}:i\in S} \lambda_Z(S) = Z_i(t) \quad \forall i \in [N] \tag{3}$$

The potential function $\Phi(t)$ is defined as follows.

$$\Phi(t) = \frac{R}{\epsilon}(L_Z(t))^2 \tag{4}$$

2.2 High Level Idea

We first give a high-level overview of the analysis. For notational convenience, let time ∞ refer to a sufficiently large time step by which all packets are completed by our policy and the optimal scheduler. This is well-defined since the total number of packets arriving into queues is finite. Our final goal is to show

$$\int_0^\infty V(t)dt \le \frac{2R^2}{\epsilon^3} \int_0^\infty V^*(t)dt, \tag{5}$$

where $V(t) := \sum_i Q_i(t)$ denotes the total workload waiting in our algorithm's queues. Likewise, $V^*(t)$ is analogously defined for the optimal scheduler. The left-hand-side quantity is the algorithm's total fractional flow time, and the right-hand-side quantity is the optimal scheduler's total fractional flow time, which is at most the optimal scheduler's total integral

flow time. Once we establish the bound (5), using the fact that the number of packets alive at each time t is at most $V(t) + N$, we will be able to complete the analysis.

To show (5), it suffices to show that the following standard conditions are satisfied.

1. Boundary condition: $\Phi(0) = \Phi(\infty) = 0$.
2. Discontinue changes: $\Phi(t)$ does not change when a packet arrives, or is completed by the online algorithm or the optimal scheduler.
3. Continuous changes: For every time t when no job arrives or completes, $V(t) + \frac{d}{dt}\Phi(t) \leq \frac{2R^2}{\epsilon^3} V^*(t)$.

The first condition is easy to check since the potential is clearly 0 when no jobs are waiting in the algorithm's or in the optimal scheduler's queues. When a new packet arrives into Q_i, $Q_i(t)$ and $Q_i^*(t)$ both increase by 1, hence $Z_i(t)$ and $L_Z(t)$ remain the same. Completion of packets does not affect the potential since $Z_i(t)$ and $L_Z(t)$ are defined in the continuous time domain. Hence the second condition follows. Note that there are only a finite number of time steps when discontinuous changes occur.

2.3 Competitive Analysis of Queue Size

In view of the above discussion, we will focus on proving the third condition concerning continuous changes. Throughout this section, we will consider an infinitesimal time interval $[t, t + \mathrm{dt}]$ during which no discontinuous changes occur. We will consider two cases. In the first case where the algorithm has a workload comparable to that of the optimal solution, we charge the algorithm's workload plus the possible potential increase to the optimal solution's workload. In the other case, the algorithm's workload in each queue Q_i is very similar to $Z_i(t)$ which is always non-negative and measures how much the algorithm is behind the optimal solution in terms of the workload in queue Q_i. This is the case where the potential helps the algorithm in need. In this case, the algorithm will effectively decrease the makespan $L_Z(t)$ based on lags $\{Z_i(t)\}$ in the potential function, which will cancel off the potential increase due to the optimal solution's processing and give enough credits to pay for the algorithm's workload due to the extra speed the algorithm is given.

We begin with a couple of definitions.

▶ **Definition 4.** A queue Q_i is said to be tight at time t if $Z_i(t) \geq (1 - \epsilon)Q_i(t)$, otherwise loose. An independent set $S \in \mathcal{S}$ is said to be tight at time t if *all* queues in S are tight. Otherwise, S is said to be loose.

The following simple observation will be used throughout the analysis.

▶ **Proposition 5.** $L_Q(t) \leq V(t) \leq R \cdot L_Q(t)$.

To study the continuous changes of $\Phi(t)$, we will first take a close look at the optimal scheduler's effect on $\Phi(t)$, freezing the algorithm's effect. Let $\frac{d}{dt}\Phi(t)\big|_{\mathrm{OPT}}$ denote the continuous change of $\Phi(t)$ due to the optimal scheduler's processing.

▶ **Lemma 6.** $\frac{d}{dt}\Phi(t)\big|_{\mathrm{OPT}} \leq 2(1 - 10\epsilon)\frac{R}{\epsilon} L_Z(t)$.

Proof. Fix a time t and consider an infinitesimal time interval $[t_1 = t, t_2]$ where *no discontinuous changes* occur. Let $L_{Z,1}$ denote the minimum makespan for queues with workload $\{Z_i(t_1)\}_{i\in[N]}$. Let $L'_{Z,2}$ the minimum makespan for queues with workload $\{Z'_i(t_2)\}$ where $Z'_i(t_2) := \max\{Q_i(t_1) - Q_i^*(t_2), 0\} \leq \max\{Q_i(t_1) - Q_i^*(t_1), 0\} + (Q_i^*(t_1) - Q_i^*(t_2))$. Note that $(Q_i^*(t_1) - Q_i^*(t_2))$ is non-negative, and refers to the amount of work that the optimal scheduler

does for queue Q_i during $[t_1, t_2]$. Observe that $L'_{Z,2} \le L_{Z,1} + (1-10\epsilon)(t_2 - t_1)$. This is because one can empty all queues with sizes $Z_i(t_1) + (Q^*_i(t_1) - Q^*_i(t_2))$ by following the schedule that achieves the makespan $L_{Z,1}$ for workload $\{Z_i(t_1)\}$, and the optimal schedule during $[t_1, t_2]$ with 1-speed; recall that the optimal scheduler has $(1-10\epsilon)$-speed. Such a schedule only does more work than the required workload, $\{Z'_i(t_2)\}_{i\in[N]}$. Hence due to downward closedness of $\mathcal{S}$, we have $\frac{d}{dt}L_Z(t)\big|_{\mathrm{OPT}} \le 1-10\epsilon$, which completes the proof. ◀

We now study the more interesting case of continuous changes of $\Phi(t)$ due to the algorithm's processing freezing the optimal scheduler's processing. We consider two cases depending on the magnitude of the volume of loose queues. Let $V_{loose}(t)$ denote the total sum of $Q_i(t)$ over all loose queues Q_i. $V_{tight}(t)$ is similarly defined for tight queues.

Case (i): $V_{loose}(t) \ge (\epsilon/R)V(t)$, i.e. $V_{tight}(t) \le (1-\epsilon/R)V(t)$. By definition, for any loose queue Q_i, we have $Z_i(t) := \max\{Q_i(t) - Q^*_i(t), 0\} \le (1-\epsilon)Q_i(t)$, hence $Q^*_i(t) \ge \epsilon Q_i(t)$. Thus $V^*(t) \ge \sum_{i:loose} Q^*_i(t) \ge \sum_{i:loose} \epsilon Q_i(t) = (\epsilon^2/R)V(t)$. We directly charge $V(t)$ to $V^*(t)$. We also charge $\frac{d}{dt}\Phi(t)\big|_{\mathrm{OPT}}$ to $V^*(t)$. Towards this end, we use Lemma 6 together with the following simple observation which immediately follows from $Z_i(t) \le Q_i(t)$ for all i, and downward closedness of $\mathcal{S}$.

▶ **Proposition 7.** *$L_Z(t) \le L_Q(t)$.*

Hence $\frac{d}{dt}\Phi(t)\big|_{\mathrm{OPT}} \le \frac{2R}{\epsilon}L_Z(t) \le \frac{2R}{\epsilon}L_Q(t) \le \frac{2R}{\epsilon}V(t) \le \frac{2R^2}{\epsilon^3}V^*(t)$.

The algorithm's processing can only decrease $\Phi(t)$, i.e. $\frac{d}{dt}\Phi(t)\big|_{algo} \le 0$, but we do not need it in this case; in the other case, we will need a stronger bound which is stated in Lemma 10. Combining these, the desired third condition easily follows.

Case (ii): Otherwise. This is the more interesting case where the potential plays a crucial role. We begin with showing the following lemma. Intuitively, since $Z_i(t)$ is very close to $Q_i(t)$ over most queues, the makespan for workload $\{Z_i(t)\}$ should be almost as large as the makespan for workload $\{Q_i(t)\}$.

▶ **Lemma 8.** *$L_Z(t) \ge (1-3\epsilon)L_Q(t)$.*

Proof. It is easy to see that the total weight of tight independent sets in $\{\lambda_Q(S)\}$ is at least $L_Q(t) - \frac{\epsilon}{R}V(t) \ge (1-\epsilon)L_Q(t)$, since one unit of loose workload in a queue can make at most one unit of independent sets loose. In other words, among independent sets of total weight $L_Q(t)$, tight independent sets have total weight at least $(1-\epsilon)L_Q(t)$, which lower bounds how much tight independent sets contribute to the makespan. Now for the sake of contradiction, suppose that $L_Z(t) < (1-3\epsilon)L_Q(t)$. This implies that there is a way of scheduling tight queues Q_i with workload $Z_i(t) \ge (1-\epsilon)Q_i(t)$ within $(1-3\epsilon)L_Q(t)$ time steps, so all tight queues Q_i with workload $Q_i(t)$ within $(1-3\epsilon)L_Q(t)/(1-\epsilon) < (1-\epsilon)L_Q(t)$ time steps. Hence we can complete all workload appearing in the tight independent sets more quickly than $(1-\epsilon)L_Q(t)$ time steps, which is a contraction to the minimality of $L_Q(t)$. ◀

For notational convenience, let $\{\lambda_Z(S)\}$ be an optimal fractional cover that achieves the minimum makespan for workload $\{Z_i(t)\}$ as illustrated in (3). We now take a close look at $\{\lambda_Z(S)\}$ focusing on tight independent sets.

▶ **Lemma 9.** *The total weight of tight independent sets in $\{\lambda_Z(S)\}$ is at least $(1-3\epsilon)L_Z(t)$.*

Proof. Recall that the total workload of loose queues is at most $(\epsilon/R)V(t) \le \epsilon L_Q(t)$. The total weight of tight independent sets in $\{\lambda_Z(S)\}$ is then at least $L_Z(t) - \epsilon L_Q(t) \ge L_Z(t) - \epsilon L_Z(t)/(1-3\epsilon) \ge (1-3\epsilon)L_Z(t)$. The first inequality is due to Lemma 8. ◀

Let $y_i(t)$ denote the total weight of tight independent sets in $\{\lambda_Z(S)\}$ that contain queue Q_i. Since our goal is to lower bound the potential's decrease due to A's processing, we can assume that Q_i is processed at a rate of $\frac{y_i(t)}{L_Q(t)} \leq \frac{Z_i(t)}{L_Q(t)} \leq \frac{Q_i(t)}{L_Q(t)}$ due to Proposition 3 and downward closeness of $\mathcal{S}$. For each tight independent set S, imagine that we process queue Q_i in S at a rate of $\frac{\lambda_Z(S)}{L_Q(t)}$ – here note that each Q_i is processed exactly at a rate of $\frac{y_i(t)}{L_Q(t)}$ in total. Hence the weight of tight indepedent set S decreases at a rate of $\frac{\lambda_Z(S)}{L_Q(t)}$, and the total weight of tight independent sets decreases at a rate of $\frac{(1-3\epsilon)L_Z(t)}{L_Q(t)} \geq (1-3\epsilon)^2 \geq 1-6\epsilon$ by summing over all tight independent sets, and subsequently by applying Lemma 9 and 8. Hence we derive the following lemma.

▶ **Lemma 10.** $\frac{d}{dt}L_Z(t)\Big|_{algo} \leq -(1-6\epsilon)$, *and* $\frac{d}{dt}\Phi(t)\Big|_{algo} \leq -2\frac{R}{\epsilon}(1-6\epsilon)L_Z(t)$.

We are now ready to complete the analysis. By Lemma 6, 10, 8, we have $\frac{d}{dt}\Phi(t) = -8R \cdot L_Z(t) \leq -4R \cdot L_Q(t) \leq -4V(t)$. In either of the two cases, we have shown the third condition on $\Phi(t)$ concerning the continuous changes.

2.4 From Queue Size to Flow Time

So far we have shown (5). By processing packets in each queue in first-in-first-out order, we have that $0 \leq A_i(t) - Q_i(t) < 1$ for all i, where $A_i(t)$ denotes the number of packets in queue Q_i at time t. Let $A(t) := \sum_i A_i(t)$. Let $\mathcal{T}$ be the collection of maximal intervals such that $V(t) > 0$ at all times t during T, for each time interval $T \in \mathcal{T}$. Note that $A(t) = 0$ if $V(t) = 0$. Hence the algorithm's total flow time is

$$\sum_{T\in\mathcal{T}}\sum_{t\in T} A(t) \leq \sum_{T\in\mathcal{T}}\sum_{t\in T}\int_{\tau=t}^{t+1}\Big(V(t)+N\Big)d\tau \leq O(\frac{R^2}{\epsilon^3})\int_0^\infty V^*(t)dt + N\sum_{T\in\mathcal{T}}|T|$$

Since our algorithm processes at least one unit of workload at each time if it exists, it is easy to observe that $\sum_{T\in\mathcal{T}}|T|$ is at least $\Omega(1)$ times the total number of packets. Knowing that each packet has flow time at least one, we can upper bound $N\sum_{T\in\mathcal{T}}|T|$ by N times the total number of packets. By dividing both sides of the inequality by the total number of packets, we derive the first part of Theorem 1.

The second part follows by a standard method of converting an algorithm good for fractional flow time into one for integral flow time. In fact, the quantity $\int_0^\infty V(t)dt$ in Equation (5) is the algorithm's total fractional flow time, where each packet incurs a penalty equal to its remaining size. In general, one can translate an online algorithm that is c-competitive with s speed for the average fractional flow time objective online into one that is $O(c/\epsilon)$-competitive with $s(1+\epsilon)$-speed for the average integral flow objective, for any $0 \leq \epsilon < 1$. For example, see [7]. However, after this conversion, the algorithm is no longer myopic.

3 Emulation by Integral Schedules

Once we have a fractional algorithm we can convert it into an integral algorithm with a small loss in the competitive algorithm. We remind the reader that an integral algorithm must schedule exactly one independent set S out of $\mathcal{S}$ at each time t. In contrast, a fractional algorithm schedules an independent set S for any infinitesimal time step dt, and reduces each queue in S by dt. Shah et al. [15] show that given an fractional schedule B, one can obtain an integral schedule on the fly that has an $O(RN)$ upper bound on the algorithm's total lag at each time as opposed to the optimal schedule.

Let $Q_i^B(t)$ be the number of packets waiting in queue Q_i at time t in the integral algorithm B's schedule. The quantity $Q_i^A(t)$ is analogously defined for a given fractional online algorithm A – however, $Q_i^A(t)$ does not have to be integral at an integer time point. Define B's lag as opposed to A as $\Delta_i(t) := \max\{Q_i^B(t) - Q_i^A(t), 0\}$. The integral algorithm B has the following description at any time t.

- Compute a (fractional) min-makespan schedule on $\{\Delta_i(t), i = 1, 2, \ldots, N\}$.
- In the above solution, if some independent set is scheduled to an amount at least 1, schedule this independent set.
- Otherwise, let $W(t) = \{i|\Delta_i(t) \geq 1\}$. Find an independent set that schedules the most number of packets from $W(t)$, and schedule this set.

The following lemma restates Lemma 5.7 and 5.8 in [15].

▶ **Lemma 11.** *For the algorithm B as described above, we have $\sum_i \Delta_i(t) \leq R(N+2)$.*

Note that the above lemma is only concerned with the difference between the queues of the two algorithms A and B. However, we can now use an idea similar to the one we used in Section 2.4. The crucial observation is that the discrepancy $\sum_i \Delta_i(t)$ can occur only at the times where the algorithm B has at least one packet at time t; the number of such times is at least the total number of packets arriving at queues. Theorem 2 easily follows from Theorem 1 and this observation.

4 Conclusions

In this paper, we studied competitive algorithms for constrained queueing systems, which have been extensively studied in stochastic queueing theory. In queueing theory, the goal is to obtain a stable algorithm when the load reaches the inherent "system threshold". A parameter is often used to measure the proximity of the system load to the threshold. For example, in the special case of switch scheduling, the expected load arriving at queues is assumed to have a makespan at most $1 - \epsilon$ for $\epsilon > 0$. The work of [15] shows an expected total queue size of $O(R/\epsilon + RN)$, and the additive term RN is ignored assuming that ϵ is arbitrarily small. In contrast, we give an $O(R^2/\epsilon^3 + RN)$-competitive algorithm for the average flow time objective when compared to the optimal scheduler with $(1 - \epsilon)$-speed. As illustrated in the seminal paper [8] that introduced the resource augmentation analysis model, a slightly weaker adversary/benchmark is also motivated by the system threshold. The goal is to design an online algorithm whose objective is comparable to the optimal offline solution with the least possible speed augmentation for all adversarial inputs. As mentioned before, such an algorithm is said to be scalable. Despite the seemingly similar goals and motivations as well as the seemingly similar analytic bounds, the relation between queueing theory (stability) and competitive analysis (scalability) remains unclear. Our work suggests the possibility of making a rigorous connection between the two concepts.

An open problem is to get tighter upper or lower bounds on the competitive ratio, even in the special case of switch scheduling. The lower bounds in [5, 6] do not hold since it requires packets having varying sizes. The only known negative resultis that the problem does not admit a (bounded) competitive algorithm without speed augmentation. However, we cannot even rule out the existence of $O(\text{poly}(1/\epsilon))$-competitive algorithms with $(1 + \epsilon)$-speed.

References

1. S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA*, pages 1228–1241, 2012. URL: `http://dl.acm.org/citation.cfm?id=2095213`.
2. Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *J. ACM*, 54(1):3:1–3:39, March 2007.
3. T. Bonald, L. Massoulié, A. Proutière, and J. Virtamo. A queueing analysis of max-min fairness, proportional fairness and balanced fairness. *Queueing Syst. Theory Appl.*, 53(1-2):65–84, June 2006. `doi:10.1007/s11134-006-7587-7`.
4. Shang-Tse Chuang, Ashish Goel, Nick McKeown, and Balaji Prabhakar. Matching output queueing with a combined input/output-queued switch. *IEEE Journal on Selected Areas in Communications*, 17(6):1030–1039, 1999.
5. Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints. In *STOC*, 2014.
6. Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive flow time algorithms for polyhedral scheduling. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 506–524, 2015. `doi:10.1109/FOCS.2015.38`.
7. Sungjin Im, Benjamin Moseley, and Kirk Pruhs. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News*, 42(2):83–97, 2011. `doi:10.1145/1998037.1998058`.
8. Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.
9. F. P. Kelly, A. K. Maulloo, and D. K. H. Tan. Rate control for communication networks: Shadow prices, proportional fairness and stability. *The Journal of the Operational Research Society*, 49(3):pp. 237–252, 1998.
10. F.P. Kelly, L. Massoulié, and N.S. Walton. Resource pooling in congested networks: proportional fairness and product form. *Queueing Systems*, 63(1-4):165–194, 2009. `doi:10.1007/s11134-009-9143-8`.
11. Nick McKeown. The islip scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 7(2):188–201, 1999.
12. Nick McKeown, Adisak Mekkittikul, Venkat Anantharam, and Jean Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Transactions on Communications*, 47(8):1260–1267, 1999.
13. Michael J. Neely, Eytan Modiano, and Yuan-Sheng Cheng. Logarithmic delay for $n \times n$ packet switches under the crossbar constraint. *IEEE/ACM Trans. Netw.*, 15(3):657–668, June 2007.
14. Balaji Prabhakar, Nick McKeown, and Ritesh Ahuja. Multicast scheduling for input-queued switches. *IEEE Journal on Selected Areas in Communications*, 15(5):855–866, 1997.
15. Devavrat Shah, Neil Walton, and Yuan Zhong. Optimal queue-size scaling in switched networks. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'12, pages 17–28, 2012.
16. Leandros Tassiulas and Anthony Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37(12):1936–1948, 1992.