

# Improved Bounds for Scheduling Flows under Endpoint Capacity Constraints

Searidang Pa\*

Rajmohan Rajaraman<sup>†</sup>

David Stalfa<sup>‡</sup>

## Abstract

We consider the problem of scheduling flows subject to endpoint capacity constraints. We are given a set of capacitated nodes and an online sequence of requests where each request has a release time and a demand that needs to be routed between two nodes. A schedule specifies, for each time step, the requests that are routed in that step under the constraint that the total demand routed on a node in any step is at most its capacity. A key performance metric in such a scheduling scenario is the response time (or flow time) of a request, which is the difference between the time the request is completed and its release time. Previous work has shown that it is impossible to achieve bounded competitive ratio for average response time without resource augmentation, and that a constant factor competitive ratio is achievable with augmentation exceeding two (Dinitz-Moseley Infocom 2020). For the maximum response time objective, the best known result is a 2-competitive algorithm with a resource augmentation at least 4 (Jahanjou et al SPAA 2020).

In this paper, we present improved bounds for the above flow scheduling problem under various response time objectives. Our first result is a lower bound showing that, without resource augmentation, the best competitive ratio for the maximum response time objective is  $\Omega(n)$ , where  $n$  is the number of nodes. The remaining results present simple, resource-augmented algorithms that are competitive for the maximum response time objective in their respective settings. Our first algorithm, Proportional Allocation, uses  $(1 + \varepsilon)$  resource augmentation to achieve a  $(1/\varepsilon)$ -competitive ratio for maximum response time in the setting with general demands, general capacities, and splittable jobs, for any  $\varepsilon > 0$ . Our second algorithm, Batch Decomposition, is 2-competitive (resp., matches optimum) for maximum response time using resource augmentation 2 (resp., 4) in the setting with unit demands, unit capacities, and unsplittable jobs. We also derive bounds for the simultaneous approximation of average and maximum response time metrics.

## 1 Introduction

We study the problem of scheduling an online sequence of flows subject to endpoint capacity constraints, which we call *Endpoint Capacitated Flow Scheduling* or ECFS. Such flow scheduling instances arise in diverse scenarios in networks and distributed systems, where requests for communication or data transfer need to be satisfied, while meeting constraints at the source and destination ports of the requests. For instance, allocating resources for packet flows in a crossbar switch is a classic example of this flow scheduling problem [4, 5, 6, 16]. More recently, variants of ECFS have been used to model the scheduling of dynamically reconfigurable topologies [10, 2].

We model ECFS by a set of capacitated nodes and an online sequence of requests where each request has a release time and a demand that needs to be routed between two nodes. An ECFS *schedule* specifies, for each time step, the requests that are allocated in that step under the constraint that the total demand allocated at any endpoint in any step is at most its capacity. For a given request, two measures of interest are its *completion time* and its *response time*, which is the difference between its completion time and its release time. In online scheduling scenarios with variable release times of flows, response time metrics are arguably of greater significance [2].

**1.1 Overview of Results & Techniques.** The focus of this paper is on designing and analyzing online algorithms for ECFS under response time metrics in the competitive analysis framework [1, 17]. Two natural variants of response time metrics are the *average response time*, which considers the overall performance of the schedule by giving equal weight to each job, and the *maximum response time*, which is motivated by fairness and aims to provide a guarantee to all jobs. We differentiate between the *general case*, where demands and capacities are arbitrary integers, and the *unit case*, where demands and capacities are unit. We further differentiate between *splittable* jobs, which can be fractionally scheduled over multiple time steps, and *unsplittable* jobs which need to be satisfied in a single step.

---

\*Northeastern University, [pa.s@northeastern.edu](mailto:pa.s@northeastern.edu)

<sup>†</sup>Northeastern University, [rajaraman.r@northeastern.edu](mailto:rajaraman.r@northeastern.edu)

<sup>‡</sup>Northeastern University, [stalfa.d@northeastern.edu](mailto:stalfa.d@northeastern.edu)

Algorithm	Resources	Competitive Ratio	Objective	Setting	Source
Proportional	$1 + \varepsilon$	$1/\varepsilon + 1/\text{OPT}$	maximum	splittable jobs general capacities/demands	*
LP Batch	4	2	maximum	unsplittable jobs general capacities/demands	[9]
Batch Decomposition	$2k$	$\max\{2/k, 1\}$	maximum	unsplittable jobs unit capacities/demands	*
FIFO	$2 + k$	$\max\{2/k, 1\}$	maximum	unsplittable jobs unit capacities/demands	*
FIFO	$2 + k$	$2(2 + k)/k$	average	unsplittable jobs unit capacities/demands	[2]
Shortest Job First	$2 + \varepsilon$	$2(2 + \varepsilon)/\varepsilon$	average	splittable jobs general capacities/demands	[2]
$\varepsilon$ is any positive real number, $k$ is any positive integer			* indicates a result proved in this paper		

Table 1: Comparison of Algorithms for Endpoint Capacitated Flow Scheduling

Previous work has shown that it is impossible to achieve a bounded competitive ratio for average response time without resource augmentation [2, 9] and that a constant factor competitive ratio is achievable for splittable jobs with augmentation exceeding two [2]. For the maximum response time objective, the best known result is a 2-competitive algorithm with a resource augmentation 4 [9]. In this paper, we present *improved bounds for ECFS under various response time objectives*.

**Lower bound for maximum response time.** Our first result is a lower bound on the best competitive ratio achievable for maximum response time (without resource augmentation).

**THEOREM 1.** *For any deterministic online ECFS algorithm Alg that is allowed to split jobs, there exists an instance with unit demands and capacities such that the maximum response time of Alg is  $\Omega(n)$  times that achieved by an optimal schedule that does not split jobs, where  $n$  is the number of nodes.*

The strong lower bound of Theorem 1 holds even for bipartite flows. The proof is an adaptation of an elegant construction in [7] which establishes a similar lower bound for *conflict scheduling*, a generalization of ECFS with unsplittable jobs. Together with previous lower bounds for average response time, Theorem 1 indicates that competitive ratio bounds for ECFS under response time metrics are more meaningful in settings that allow for resource augmentation. In the remainder of the paper, we focus on the tradeoff between resource augmentation and the competitive ratio achieved by ECFS schedules.

**Upper bounds for maximum response time.** Our next set of main results provides upper bounds on the competitiveness of two algorithms. The first, Proportional Allocation, applies in the setting with general demands, general capacities, and splittable jobs. The second, Batch Decomposition, applies in the setting with unit demands, unit capacities, and unsplittable jobs.

**THEOREM 2.** *Given an ECFS instance with splittable jobs, Proportional Allocation yields a  $(1/\varepsilon)$ -competitive schedule for maximum response time, assuming the capacity of each node is increased by a factor of  $1 + \varepsilon$ , for any  $\varepsilon > 0$ .*

**THEOREM 3.** *Given an ECFS instance with unsplittable jobs, unit demands, and unit capacities, Batch Decomposition yields a  $\max\{2/k, 1\}$ -competitive schedule for maximum response time, assuming the capacity of each node is increased by a factor of  $2k$ , for  $k \in \{1, 2\}$ .*

Our upper bound results for maximum response time and previous bounds for maximum and average response time are summarized in Table 1. We note that the LP Batch algorithm does not split jobs, and so assumes that all

requests have demands less than the capacities of the nodes on which they arrive. This presents a limitation for LP Batch in the most general setting. Proportional Allocation and Shortest Job First make no such assumption about the size of the requests. However, in the case where jobs are unsplittable, Proportional Allocation and Shortest Job First are not well defined. Both split jobs even when all jobs can be routed in a single round on both of their nodes.

**Simultaneous approximation of response time metrics.** The upper bound results suggest one should use the Shortest Job First algorithm to optimize average response time, and one should use Proportional Allocation to optimize for maximum response time. It is natural to ask whether there is an algorithm that is simultaneously competitive for both average and maximum response times. We find that Proportional Allocation has poor competitive ratio for average response time, even under arbitrary resource augmentation (Lemma 5.1), while Shortest Job First has poor competitive ratio for maximum response time under resource augmentation at most two (Lemma 5.2). However, the upper bounds for these two algorithms yield a new hybrid algorithm which provides guarantees for both average and maximum response time. In each round, route jobs according to Equal Priority with  $\varepsilon_1$  and also according to Shortest Job First with  $\varepsilon_2$ . This specifies an algorithm that is  $1/\varepsilon_1$  competitive for maximum response time and  $2(2 + \varepsilon_2)/\varepsilon_2$  competitive for all  $\ell_p$  norms ( $p \in [1, \infty)$ ), where the algorithm increases the capacity on each port by a factor of  $3 + \varepsilon_1 + \varepsilon_2$ .

For unsplittable jobs in the unit case, one can establish a better bound. We show that for any positive integer  $k$ , a simple FIFO algorithm with resource augmentation of  $2 + k$  achieves a competitive ratio of  $\max\{1, 2/k\}$  for maximum response time, while previous work of [2] established a bound of  $2(2 + k)/k$  for average response time.

**Techniques.** The algorithms studied in this paper explore simple techniques for scheduling flows: spreading the flows proportional to load (Proportional Allocation), assigning priorities to jobs based on size or release time (Shortest Job First and FIFO), and scheduling jobs in batches (Batch Decomposition). Our results indicate that these different approaches yield different tradeoffs for response time.

All of our upper bounds for maximum response time use the same lower bound on the optimum, which we refer to as the *interval lower bound*. Informally, we consider every interval of input rounds over the course of a given ECFS instance. Whenever the total load placed on any node during the interval exceeds the length of the interval, we obtain a non-trivial lower bound on the maximum response time, given by the difference between the load and the total capacity of the node over the length of the interval. The interval lower bound is local and applies equally well for the much simpler setting where jobs require the resources of a single node rather than a pair of nodes. In fact, as we prove in Lemma 4.2 there are instances for which the gap between the interval lower bound and the optimal max response time is quite large. Given the strong lower bound presented in Theorem 1, it is therefore surprising that algorithms with a small amount of resource augmentation are competitive with this weak lower bound, as our upper bounds show.

**1.2 Related work.** A closely related model studies the scheduling of flows on optical WAN, modelled as reconfigurable networks. Jia et al. [10] initiated the study of scheduling on optical WAN under the makespan and sum of completion times objective. Using the same model, Dinitz and Moseley [2] provide an algorithm that is  $O(1/\varepsilon^2)$ -competitive for the weighted sum of response time objective, assuming that the speed of each machine is  $2 + \varepsilon$  times that in an optimal solution. Recent work by Kulkarni et al. [12] extend these same bounds to a two-tiered reconfigurable network in which one tier of the network is fixed and the other tier is reconfigurable. The notion of resource augmentation in these papers is subtly different from ours. These models use *speed* augmentation, which allows for faster completion of a job, while we use *capacity* augmentation which allows more unit flows to be scheduled in each round. This entails that their upper bounds immediately extend to our model, though some of ours are not valid in theirs.

An identical model to ECFS with unsplittable flows is studied in Jahanjou et al. [9], which presents algorithms for the sum of response times and maximum response time objectives. In the setting where the jobs have unit demand and form a bipartite graph on the nodes, the paper provides an offline  $O(\log n)/c$ -approximation algorithm for the sum of response times objective with a  $1 + c$  factor increase on the capacity of each node. For the maximum response time objective, the paper provides an optimal schedule in the offline setting with  $2d_{max} - 1$  capacity augmentation, where  $d_{max}$  is the maximum demand of all requests. The paper also presents a 2-competitive online algorithm for max response time with capacity increase of 4, which uses the offline algorithm as a subroutine. Finally, Jahanjou et al. prove the NP-hardness of constructing an ECFS schedule which minimizes maximum response time.

The more general setting of conflict scheduling is studied in [7, 8, 3]. In this model, a node represents a class of jobs, and two nodes are adjacent if the corresponding two classes of jobs require the same resource. In each round, only jobs belonging to an independent set of nodes can be executed. ECFS can be reduced to conflict scheduling by considering the line graph of the requests. For the maximum response time objective, Irani and Leung [7] provide an  $O(n^3 \text{OPT}^2)$ -competitive algorithm for interval graphs and an  $O(n^2 \text{OPT}^2)$ -competitive algorithm for bipartite graphs, where  $n$  is the number of nodes in the conflict graph. We note that these results do not imply upper bounds in our model because ECFS instances may transform to conflict graphs that are neither bipartite nor interval. The paper also provides a strong  $\Omega(n)$  lower bound on the competitive ratio of any algorithm for max response time. To overcome this lower bound, Irani and Leung [8] consider the probabilistic version of model, where a job arrives on node  $i$  with probability  $p_i$  in each round.

In a related setting, Guez et al. [6] studies the problem of maximizing the throughput of packets on input-queued switches. Packets are queued at input ports, and are forwarded to output ports through a cross switch fabric. The main difference with ECFS is that their model allows a packet to be dropped, while ours requires that all requests be served. There is extensive literature on scheduling matchings over high-speed crossbar switches and these studies largely adopt a queuing theoretic framework (e.g., see [4, 5, 16]). Approximation algorithms for average completion time of co-flows, a collection of flows that share a common performance goal, on a non-blocking switch are given in [15] and [11].

## 2 Model

An instance of ECFS is composed of the following elements. We are given a set  $V$  of  $n$  nodes. Each node  $i \in V$  has a corresponding capacity  $c_i$ . There is also a set  $J$  of  $m$  requests or jobs. Each request  $j$  has an associated set of nodes  $\{i, i'\}$ , demand  $d_j \in \mathbb{R}^+$ , and release  $r_j \in \mathbb{N}^+$ . For a given node  $i$ , we say that  $i \in j$  if  $i$  is contained in the set of nodes associated with  $j$ . In each round, we may execute any released requests subject to the constraint that for any port  $i$ , the total demand of all executed requests  $j$  adjacent to  $i$  is no more than  $c_i$ . More formally, if  $x_{j,t}$  is the fraction of request  $j$  executed in round  $t$ , then,  $x_{j,t} > 0$  implies that  $r_j \leq t$  and, for every node  $i$ , we have that  $\sum_{j \ni i} d_j x_{j,t} \leq c_i$ . We also require that, over the course of the entire schedule, every job is completely executed, i.e. for all jobs  $j$ , we have that  $\sum_t x_{j,t} \geq 1$ . In any round  $t$  in the schedule, if there is a request  $j$  with  $r_j \leq t$  and  $j$  has not been completely scheduled, then we say that  $j$  is *pending*.

For a given instance, the goal is to construct a schedule that has good performance relative to some objective. Here, we consider two objectives: maximum response time and average response time. The response time of a job  $j$  is the difference between  $j$ 's release time  $r_j$  and its completion time  $C_j$ , which is the first round in which the job is completed, i.e. the first round after  $r_j$  in which  $j$  is no longer pending. Note that if a job is executed in the same round that it arrived, its response time is 1. For a given schedule, the maximum response time is the maximum amount of rounds between the release of a job and its completion:  $\max_j \{C_j - r_j\}$ . The average response time of a schedule is the sum of all response times over the number of jobs:  $(\sum_j C_j - r_j) / m$ .

We distinguish between splitting and nonsplitting schedules. A *splitting* schedule is permitted to schedule fractions of a job over multiple rounds, i.e. for any job  $j$ , the fraction of  $j$  routed in round  $t$  may be between 0 and 1. On the other hand, a *nonsplitting* schedule must execute the job entirely in a round or not at all, i.e. for any job  $j$ , the fraction of  $j$  routed in round  $t$  is either 0 or 1.

We consider the problem in the online setting. In this setting, at the start of each round  $t$ , an algorithm receives the set of jobs  $j$  such that  $r_j = t$ . The algorithm must then irrevocably schedule any of the pending jobs in that round before moving to the next. We analyze the performance of such algorithms using competitive analysis, where the objective value of the algorithm is compared against the offline optimum [17, 1]. Formally, if ALG is the algorithm's objective value and OPT is the offline optimum, then the *competitive ratio* is given by the ratio of ALG to OPT.

We also allow our algorithms to use some amount of *resource augmentation*. If an algorithm uses resource augmentation  $\gamma$ , then in any round the total load that can be scheduled on node  $i$  is  $\gamma \cdot c_i$ . Our competitive analysis then compares the objective value achieved by the algorithm against the offline optimum *without resource augmentation*.

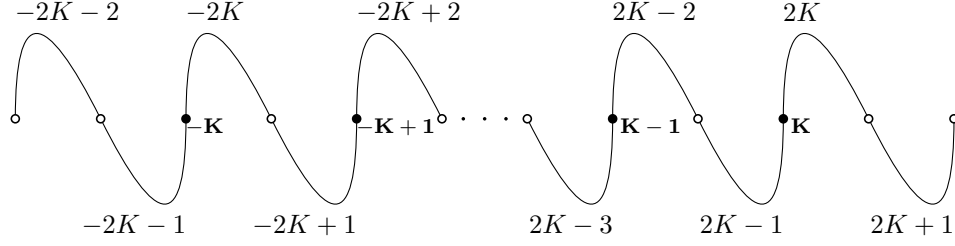


Figure 1: A chain of  $4K + 5$  nodes. Each edge is labelled from  $-2K - 2$  to  $2K + 1$ . Each black node is labelled from  $-K$  to  $K$ .

### 3 Lower Bound without Resource Augmentation

In this section we show that there is no algorithm for ECFS that achieves good competitive ratio for maximum response time without resource augmentation, even for the unit case when all demands and capacities are unit. This result (Theorem 1 restated below) motivates our use of resource augmentation in the remainder of the paper.

**THEOREM 1.** *For any deterministic online ECFS algorithm Alg that is allowed to split jobs, there exists an instance with unit demands and capacities such that the maximum response time of Alg is  $\Omega(n)$  times that achieved by an optimal schedule that does not split jobs, where  $n$  is the number of nodes.*

The construction used to prove Theorem 1 closely follows the one used in the elegant lower bound proof for the conflict scheduling problem in [7]. We note, however, that while the setting of [7] is more general, their result does not immediately extend to ECFS because they do not allow for splittable jobs.

We consider a chain with  $n = 4K + 5$  nodes for any  $K \geq 1$ . Starting from the left, label every edge in the path from  $-2K - 2$  to  $2K + 1$ . For each edge with even label  $\ell \geq 2K$ , the left node of the edge is labelled  $\ell/2$ . Figure 1 shows the construction. Fix an arbitrary integer  $C \geq 1$ . We define the adversary Adv using Input Sequences 1 and 2, the former being a subroutine used in the latter, which define the adversarial input. Input Sequence 1 has two parameters  $k \leq K$  and  $c \leq C$ . Figure 2 provides examples of input patterns generated by the subroutine.

In any round of a schedule, we say that an edge has *load*  $d$  if, at the start of the round, the sum of the fractional parts of pending requests on the edge is  $d$ . Similarly, we say that a node  $i$  has load  $d$  in a round if the sum of the loads of both edges adjacent to  $i$  in the round is  $d$ .

---

#### Input Sequence 1: Adversarial Subroutine

---

**Data:** parameters  $k \leq K$  and  $c \leq C$

```

1  $t \leftarrow 1$ 
2 repeat
3    $\ell \leftarrow \lceil t/C \rceil - 1$ 
4   the following requests arrive in a single round: one on each odd numbered edge adjacent to nodes
       $-k, -k+1, \dots, -(\ell+1)$ , one on each edge adjacent to nodes  $-\ell, -\ell+1, \dots, \ell$ , one on each even
      numbered edge adjacent to nodes  $\ell+1, \ell+2, \dots, k$ 
5    $i \leftarrow$  the unlabelled node with maximum excess load in Alg after scheduling the current round
6    $t \leftarrow t + 1$ 
7 until the load on  $i$  in Alg is at least  $(Ck + c + 1)/2$ 
8 return  $i$ 
```

---

The following two lemmas give the basic strategy the adversary employs. By introducing jobs as in the subroutine, the adversary guarantees that Alg accrues a certain amount of extra load on some node while Adv is able to keep that node empty. By carefully chaining successive runs of this subroutine together, one after the other, the adversary is able to build up a significant amount of excess load in Alg, all while keeping its own schedule Adv mostly empty. Note that Lemma 3.1 also implies that Input Sequence 1 always terminates since the condition at line 7 is always eventually met.

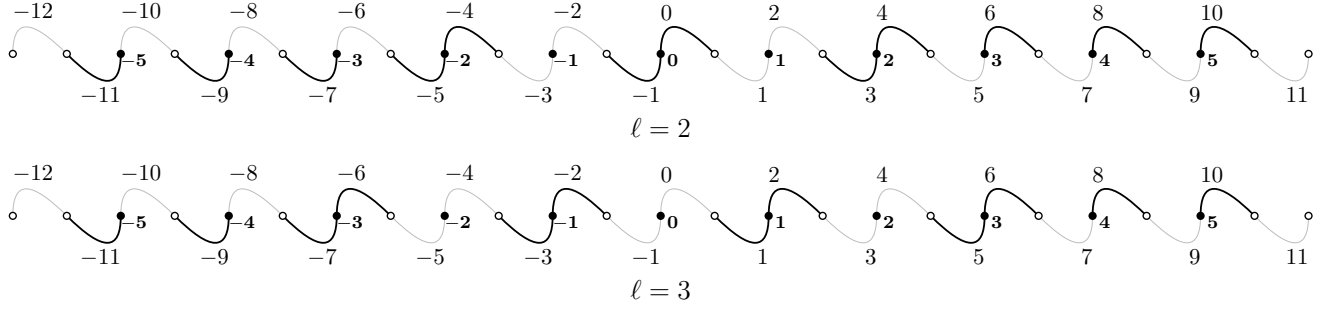


Figure 2: Two request arrival patterns for  $k = 5$ . Top shows arrival pattern for  $\ell = 2$ , bottom shows patten for  $\ell = 3$ . Requests arrive on the bold edges.

LEMMA 3.1. *Suppose that Alg has  $(Ck + c)/2$  load remaining on each odd edge in round  $T$ . Then after  $Ck + c + 1$  iterations of Input Sequence [1](#), there will be some unlabelled node  $i$  such that Alg has  $(Ck + c + 1)/2$  excess load on  $i$ .*

*Proof.* Suppose that Alg has pending load  $(Ck + c)/2$  on all odd labelled edges in round  $T$ . In each round, Input Sequence [1](#) introduces a new job on every unnumbered node (except the first and last) in the path from edge  $2k - 2$  to  $2k + 1$ . Note that no job ever arrives on edge  $-2k - 2$ . So, in order to avoid an excess load of  $(Ck + c + 1)/2$  on any node, the fractional components that Alg schedules on each odd edge in the path must sum to at least  $1/2$  in each round. Since a request arrives on edge  $2k$  in each round, after  $Ck + c + 1$  rounds there is at least  $(Ck + c + 1)/2$  load on edge  $2k$ .  $\square$

LEMMA 3.2. (IMMEDIATE COROLLARY TO LEMMA 3.3 IN [\[7\]](#)) *There exists a schedule  $\sigma$  over just the jobs introduced by Input Sequence [1](#) such that  $\sigma$  has maximum response time  $C$ . Further, if  $T$  is the last round that Input Sequence [1](#) produces input, then in round  $T + 1$  the returned node  $i$  has no load in  $\sigma$  and, in all rounds after  $T$ ,  $\sigma$  schedules jobs entirely on even edges or entirely on odd edges.*

We now define the adversarial input as Input Sequence [2](#), which is parametrized by integers  $K$  and  $C$ , and uses Input Sequence [1](#) as a subroutine. The sequence has  $KC$  iterations. In each iteration, the sequence creates an instance of Input Sequence [1](#) in order to build up load on one node  $i$  in Alg. It then allows Adv to empty any load it has remaining while maintaining the load on  $i$  in Alg. Finally, the sequence spreads the load in Alg to all the odd edges to prepare for the next iteration.

LEMMA 3.3. *On the input generated by Input Sequence [2](#), Alg has maximum response time  $C \cdot (n - 5)/4$ .*

*Proof.* We prove the lemma by showing that, for any given values of  $k \leq K$  and  $c \leq C$ , at the start of iteration  $k \cdot c$  of the line [2](#) for-loop, Alg has  $(Ck + c)/2$  pending load on all odd edges. The claim is proved by induction on  $k$  and  $c$ . The base case where  $k = c = 0$  is trivial.

Suppose that at the start of iteration  $k \cdot c$ , Alg has load  $(Ck + c)/2$  pending on all odd edges. After running Input Sequence [1](#) on input  $k, c$ , Alg has load  $(Ck + c + 1)/2$  on some node  $i$ , by Lemma [3.1](#). So, we must establish that the remainder of Input Sequence [2](#) spreads that load to all odd edges. We note that the line [9](#) for-loop does not affect the load on  $i$  in Alg: since one edge arrives on  $i$  in every round, at the end of this for-loop there remains  $(Ck + c + 1)/2$  load on  $i$  as long as Alg keeps  $i$  executing jobs at capacity in each round.

We show, by induction on  $t$ , that after  $t$  iterations of the line [9](#) for-loop, either Alg has maximum response time of  $\Omega(Cn)$  or there are at least  $t$  consecutive odd edges with load  $(Ck + c + 1)/2$  in Alg. The claim is trivial for  $t = 0$ , so assume the claim holds after  $t \geq 0$  iterations. In this case, when line [10](#) is executed, Alg can avoid having maximum response time  $Cn$  only by scheduling all jobs on the previously loaded odd edges. This places load  $(Ck + c + 1)/2$  load on at least  $t + 1$  consecutive even edges in Alg. Similar reasoning shows that after executing line [11](#), at least  $t + 1$  odd edges have load  $(Ck + c + 1)/2$  load in Alg after executing line [11](#).

The lemma then follows from the fact that there are  $2K + 2$  odd edges and setting  $K = (n - 5)/4$ .  $\square$



---

**Input Sequence 2: Adversarial Input**

---

**Data:** parameters  $K, C$

```
1 for  $k = 0$  to  $K$  do
2   for  $c = 0$  to  $C$  do
3      $i \leftarrow$  result of running Input Sequence 1 on  $k, c$ 
4     // Adv schedules the jobs from Input Sequence 1 according to Lemma 3.2
5     for  $C$  iterations do
6       if Adv schedules odd edges in the next round then
7         introduce a job on the odd edge adjacent to  $i$ 
8       else
9         introduce a job on the even edge adjacent to  $i$ 
10      // Adv executes the new job immediately
11    for  $2K + 2$  iterations do
12      introduce one request on every even numbered edge for  $Cn$  rounds
13      introduce one request on every odd numbered edge for  $Cn$  rounds
14      // Adv executes each new job as it arrives
```

---

LEMMA 3.4. (IMMEDIATE COROLLARY TO THEOREM 3.1 IN [7]) *Adv has maximum response time  $C$ .*

The proof of Lemma 3.4 (following Theorem 3.1 in [7]) uses properties of Adv that we have excluded for clarity. Essentially, each time Input Sequence 1 exits, Adv ensures that there is no load on Alg's overloaded node  $i$ . The for-loop in line 4 of Input Sequence 2 then allows Adv to schedule all remaining jobs while maintaining Alg's excess load on  $i$ . Adv can then easily handle the input introduced by the for-loop in line 8 of Input Sequence 2 because all these requests can be handled in the round in which they arrive.

Theorem 1 is entailed by Lemmas 3.3 and 3.4.

#### 4 Upper Bounds for Maximum Response Time

In this section, we present resource augmented algorithms for ECFS that are competitive for the maximum response time objective. Our use of resource augmentation is justified by the lower bound presented in Section 3. Since ECFS schedules are constrained by node capacities, we take resource augmentation to indicate increased node capacity.

Our upper bounds for maximum response time use the *interval lower bound*. Informally, we consider every interval of input rounds over the course of a given ECFS instance. If the load placed on any node during the interval is greater than the length of the interval, this provides a lower bound on the maximum response time for the entire instance. For example, if ten unit jobs arrive on a unit capacity port in a single round, then the best response time you can achieve is by executing one of these edges in the round it arrives, and the nine others in the immediate following rounds. Similarly, if a hundred jobs arrive on one node over thirty rounds, then the best response time one can hope for is seventy.

Formally, the interval lower bound  $L$  for a given ECFS instance is defined as follows.

$$(4.1) \quad L = \max_{i, t_1 \leq t_2} \left\{ \frac{1}{c_i} \cdot \sum_{t=t_1}^{t_2} \sum_{j \ni i: r_j=t} d_j - (t_2 - t_1 + 1) \right\} + 1$$

In general, we are interested only in ECFS instances for which  $L \geq 2$ . If  $L \leq 1$ , this implies that in every round  $t$  the load that arrives on  $i$  is no more than  $c_i$ . In this case, each job can be executed in the round in which it arrives and the problem is easily solvable.

Our first two lemmas establish fairly simple properties of the interval lower bound. The first shows that it lower bounds the optimal maximum response time. The second shows that for any interval of rounds, the input load on any port in that interval is no more than the length of the interval plus the interval lower bound.

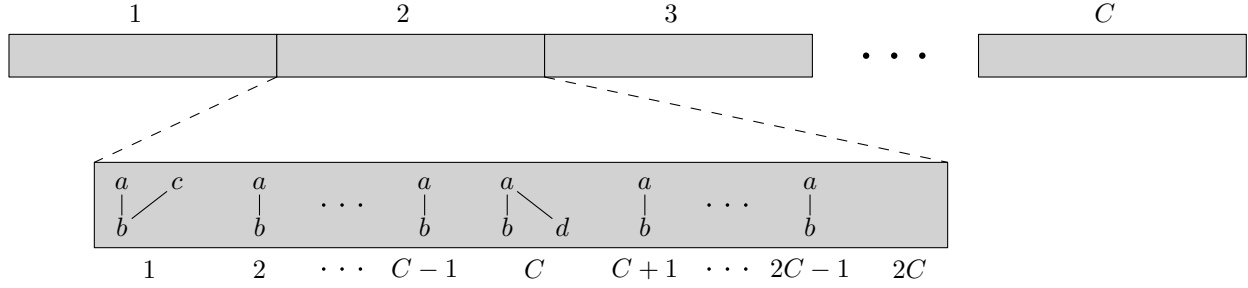


Figure 3: Input sequence for interval lower bound gap instance. The upper sequence shows the construction of the entire instance, consisting of  $C$  subinstances. The lower sequence shows the construction of a single subinstance, consisting of  $2C$  rounds.

LEMMA 4.1. *For a given instance of ECFS with interval lower bound  $L$ , the optimal maximum response time is at least  $L$ .*

*Proof.* Let  $\sigma$  be an arbitrary schedule and let  $L$  be the interval lower bound for the given ECFS instance. Fix a node  $i$  and rounds  $t_1 \leq t_2$ . The maximum load that can be executed in  $\sigma$  on node  $i$  in the interval  $[t_1, t_2]$  is  $c_i(t_2 - t_1 + 1)$ . Therefore, there is at least  $\ell = \sum_{t=t_1}^{t_2} \sum_{j \ni i: r_j=t} d_j - c_i(t_2 - t_1 + 1)$  load remaining on  $i$  after time  $t_2$ . So the last job  $j$  from the interval is executed no earlier than round  $t_2 + \ell/c_i$ . Since  $r_j \leq t_2$ , this yields a maximum response time of at least  $\ell/c_i + 1 \leq L$ .  $\square$

We note that the interval lower bound is local and applies equally well for the special case where jobs require the resources of a single node rather than a pair of nodes. A natural question arises about the strength of the interval lower bound. Our final result regarding the interval lower bound establishes that, in some cases, there is a large gap between it and the optimal maximum response time. This gap arises from the fact that the interval lower bound is merely counting the load that arrives on a given port in an interval, and does not account for other conflicts between different requests.

LEMMA 4.2. *There exists an instance with interval lower bound  $L$  for which the optimal maximum response time is  $\Omega(\sqrt{m} \cdot L)$ , where  $m$  is the number of requests.*

*Proof.* The gap instance is constructed over four unit capacity nodes  $a, b, c, d$  using unit demand jobs. Fix an arbitrary integer  $C \geq 2$ . A subinstance is composed of  $2C$  consecutive rounds. In the last round no jobs arrive. In all other rounds of the subinstance, a request arrives on nodes  $a$  and  $b$ . In the first round, an additional job arrives on  $b$  and  $c$ , and in the  $C^{\text{th}}$  round an additional job arrives on nodes  $a$  and  $d$ . The gap instance is constructed from  $C$  consecutive subinstances. Figure 3 shows the construction.

Considering the constructed instance, we prove the theorem via the following two claims: (1) the interval lower bound is 2 and (2) the optimal maximum response is at least  $C$ . Claims (1) and (2) are sufficient to prove the theorem since the total number of requests is  $2C^2 + C$ :  $2C + 1$  requests per subinstance and  $C$  subinstances.

We first prove (1). Nodes  $a$  and  $b$  are the only nodes that ever have more than one job arrive in a single round, so we need not consider nodes  $c$  and  $d$  when calculating the interval lower bound. We focus our attention on node  $a$  (the reasoning for  $b$  is similar). Note that, between any two rounds in which more than one job is introduced on  $a$ , there is at least one round in which no jobs are introduced on  $a$ . This entails that for any interval of rounds, the number of jobs that arrive on  $a$  is no more the length of the interval plus 1. This proves claim (1).

We now prove claim (2). Let  $\sigma$  be a schedule with maximum response time strictly less than  $C$ . For a fixed subinstance, there is exactly one job  $j_{a,d}$  on nodes  $a$  and  $d$  and exactly one job  $j_{b,c}$  on nodes  $b$  and  $c$ . In order for  $\sigma$  to avoid response time  $C$ , it must execute job  $j_{b,c}$  before the release of job  $j_{a,d}$ . Similarly,  $\sigma$  must execute job  $j_{a,d}$  before the final round of the subinstance where no jobs arrive. Note that, in any round where  $j_{b,c}$  or  $j_{a,d}$  is executed,  $\sigma$  cannot execute any job that requires both nodes  $a$  and  $b$ . This entails that, in each subinstance,  $\sigma$  can schedule no more than  $2C - 2$  of such jobs. Therefore, after  $C$  subinstances, there are at least  $C$  jobs pending that require both nodes  $a$  and  $b$ . The last of these jobs to be executed will have response time at least  $C$ , so we have derived a contradiction. This proves claim (2).  $\square$



**4.1 General Demands and Capacities.** We now turn to proving upper bounds on the competitive ratio for maximum response times with resource augmentation. In this section, we present an algorithm that allows for general capacities and demands, and uses only a small amount of augmentation. The algorithm is given as Algorithm 3. We define  $J(i, t)$  to be the set of pending jobs on  $i$  in round  $t$ .

---

**Algorithm 3:** Proportional Allocation

---

```

1 for each round  $t = 1, 2, \dots$  do
2   forall requests  $j$  pending in round  $t$  do
3      $f_t(j) = \min_{i \in j} \{c_i / \sum_{j' \in J(i, t)} d_{j'}\}$ 
4     execute a  $(1 + \varepsilon) \cdot f_t(j)$  fraction of  $j$ 

```

---

**THEOREM 2.** *Given an ECFS instance with splittable jobs, Proportional Allocation yields a  $(1/\varepsilon)$ -competitive schedule for maximum response time, assuming the capacity of each node is increased by a factor of  $1 + \varepsilon$ , for any  $\varepsilon > 0$ .*

*Proof.* We prove the theorem via two claims: (a) in every round there is no more than load  $c_i(1 + \varepsilon)$  on any node  $i$  and (b) for all rounds  $T$ , all requests released prior to round  $T$  are scheduled with response time at most  $\lceil L/\varepsilon \rceil$ .

The proof of (a) is straightforward. For a fixed  $i$  and  $t$ , the total scheduled load on node  $i$  in round  $t$  is at most

$$\sum_{j \in J(i, t)} \frac{c_i \cdot (1 + \varepsilon)}{\sum_{j' \in J(i, t)} d_{j'}} \cdot d_j = c_i \cdot (1 + \varepsilon).$$

We prove (b) by induction on  $T$ , letting  $\hat{L} = \lceil L/\varepsilon \rceil$ . The claim is trivial for  $T = 1$  since no requests are released prior to round 1. Consider a fixed job  $j^*$  released on nodes  $i_1$  and  $i_2$  in round  $T$ . We show that the total pending load on  $i_1$  in any round  $t \in \{T, T+1, \dots, T + \hat{L} - 1\}$  is at most  $c_{i_1}(\hat{L} + L)$ . For any  $\ell \in \{0, 1, \dots, \hat{L} - 1\}$ , the induction hypothesis entails that each request pending on  $i_1$  in round  $T + \ell$  is released in some round  $t \in \{(T + \ell) - (\hat{L} - 1), (T + \ell) - (\hat{L} - 2), \dots, T + \ell\}$ . By definition of the interval lower bound, the total load placed on  $i_1$  in the rounds from  $(T + \ell) - (\hat{L} - 1)$  to  $T + \ell$  is at most  $c_{i_1}$  times the length of the interval plus  $L$ , or  $c_{i_1}(\hat{L} + L)$ . Similar reasoning shows that the total load on  $i_2$  in any round  $t \in \{T, T+1, \dots, T + \hat{L} - 1\}$  is at most  $c_{i_2}(\hat{L} + L)$ . For  $t \in \{T, T+1, \dots, T + \hat{L} - 1\}$ , let  $i[t]$  be the node  $i \in \{i_1, i_2\}$  such that  $f_t(j) = c_i / \sum_{j \in J(i, t)} d_j$ . By definition of the algorithm we have that the total fraction of  $j$  scheduled in rounds  $T, T+1, \dots, T + \hat{L} - 1$  is

$$\sum_{t=T}^{T+\hat{L}-1} (1 + \varepsilon) f_t(j) \geq \sum_{t=T}^{T+\hat{L}-1} \frac{c_{i[t]}(1 + \varepsilon)}{c_{i[t]}(\hat{L} + L)} = \frac{(\hat{L}) \cdot (1 + \varepsilon)}{(\hat{L} + L)} \geq 1.$$

Therefore, all jobs released in time  $t$  are completed with response time  $\lceil L/\varepsilon \rceil$ . The theorem now follows from Lemma 4.1.  $\square$

We make two observations regarding this result. The first is that the Proportional Allocation algorithm makes full use of the resource augmentation afforded it. In other words, a schedule produced by Proportional Allocation would not be valid under a weaker notion of resource augmentation like, e.g. that used in [2]. Because that model uses a *speed* notion of resource rather than *capacity*, with  $\alpha$  resources their algorithms are able to schedule  $\alpha$  individual matchings. On the other hand, with the same resources, Proportional Allocation is able to schedule an  $\alpha$ -matching, which for general graphs affords our algorithm more power.

We also observe that neither the Proportional Allocation algorithm nor proofs of its guarantees rely on the fact that each job is associated with a pair of nodes, or edge, in the underlying graph. Indeed, even in the setting where each edge is specified as an arbitrary set of nodes, or an edge in the underlying hypergraph, Proportional Allocation provides exactly the same guarantees. This surprising result extends Theorem 2 to more general settings, including conflict scheduling of [7] which can be modelled as a series of node capacitated requests in a hypergraph.

**4.2 Unit Demands and Capacities.** In some settings we may prefer that jobs not be split over multiple rounds if it can be avoided, or we might demand that jobs never be split. In this respect, one issue with Proportional Allocation algorithm is that, except in uninteresting cases where at most one job arrives on any node in any round, the algorithm always splits jobs. In prior work [9], an LP based batching algorithm was given for ECFS which performs well for the maximum response time objective and does not split jobs, although the capacity increase needed for this algorithm is quite high relative to Proportional Allocation.

These considerations motivate the search for algorithms that avoid splitting jobs, are competitive with the optimal maximum response time, and require little resource augmentation. In this section, we present an algorithm which meets these criteria, although in order to do so we must restrict ourselves to the simplified setting where all job demands are unit and all node capacities are unit.

Our algorithm, Batch Decomposition, is specified as Algorithm 4. Batch Decomposition uses the notion of a 2-factor decomposition. A *2-factor* of a given multigraph is a spanning subgraph that has degree at most 2. A *2-factor decomposition* of a given multigraph is a set  $F$  of 2-factors such that, for every edge  $e$  there is some 2-factor in  $F$  that contains  $e$ . For the remainder of this subsection, we use the terms *graph* and *multigraph* interchangeably.

Batch Decomposition operates by collecting a set  $S$  of pending requests and representing them as a graph. The algorithm then constructs a 2-factor decomposition on this graph and executes  $k$  2-factors in each of the subsequent rounds until the set is empty, for  $k \in \{1, 2\}$ . The algorithm collects any jobs that arrive while the 2-factors are being scheduled into the next set  $S$  to be scheduled immediately after.

---

**Algorithm 4:** Batch Decomposition

---

**Data:** augmentation parameter  $k \in \{1, 2\}$

```

1  $P \leftarrow \emptyset$  // stores the set of pending jobs that are not currently being executed
2  $H \leftarrow \emptyset$  // stores the set of 2-factor subgraphs currently being executed
3 for rounds  $t = 1, 2, \dots$  do
4    $P \leftarrow P \cup \{j : r_j = t\}$ 
5   if  $H = \emptyset$  then
6      $H \leftarrow$  2-factor decomposition of  $P$ 
7      $P \leftarrow \emptyset$ 
8   choose up to  $k$  2-factors in  $H$ , schedule all of their requests, and remove them from  $H$ 

```

---

A useful lemma originally proved by Petersen [14, 13] shows that the minimum size of a graph's 2-factor decomposition is upper bound by the maximum degree of the graph.

LEMMA 4.3. (PETERSEN'S 2-FACTOR THEOREM) *For every positive integer  $D$ , every multigraph  $G$  with maximum degree  $2D$  can be decomposed into  $D$  spanning subgraphs  $G_1, \dots, G_D$  with maximum degree 2.*

We now use Lemma 4.3 to prove Theorem 3, which we restate below.

THEOREM 3. *Given an ECFS instance with unsplittable jobs, unit demands, and unit capacities, Batch Decomposition yields a  $\max\{2/k, 1\}$ -competitive schedule for maximum response time, assuming the capacity of each node is increased by a factor of  $2k$ , for  $k \in \{1, 2\}$ .*

*Proof.* We claim that if the number of rounds between two consecutive times the condition at line 5 is satisfied is upper bounded by  $B$ , then no job has response time greater than  $2B$ . Consider an arbitrary job  $j$  released in round  $r_j$  and completed in round  $C_j$ . Let  $T$  be the earliest round  $t \geq r_j$  for which the condition at line 5 is satisfied. By the claim and the definition of Algorithm 4, we have that  $T - r_j \leq B$  and that  $C_j - T \leq B$ . Therefore, the response time of  $j$  is no more than  $2B$ .

We now upper bound the number of rounds that pass between two consecutive times the condition at line 5 is satisfied in terms of the interval lower bound  $L$  (see Equation 4.1). More formally, we define  $T_\ell$  to be the round in which the condition is satisfied for the  $\ell^{\text{th}}$  time ( $T_0 = 0$ ). We argue by induction on  $\ell$  that  $T_\ell - T_{\ell-1} \leq L/k$ . The base case of  $\ell = 1$  is straightforward:  $T_1 - T_0 = 1 - 0 \leq L/k$  for nontrivial  $L \geq 2$ .

For the induction step, assume that  $T_\ell - T_{\ell-1} \leq L/k$  for arbitrary  $\ell \geq 1$ . By definition of the algorithm, the difference  $T_{\ell+1} - T_\ell$  is equal to  $(1/k)$  times the size of the 2-factor decomposition that is constructed in round  $T_\ell$ . By definition of the interval lower bound, the maximum load that arrived on any node in the interval  $\{T_\ell + 1, T_\ell + 2, \dots, T_{\ell+1}\}$  is at most  $L + (T_{\ell+1} - T_\ell) \leq L + (L/k)$  by the inductive hypothesis. This entails that the degree of any node the graph decomposed in round  $T_\ell$  is no more than  $L + (L/k)$ . By Lemma 4.3, we have that the size of the 2-factor decomposition of this graph is no more than  $(L + (L/k))/2$ . So, by definition of the algorithm, we have that

$$T_{\ell+1} - T_\ell \leq \frac{1}{k} \cdot \frac{1}{2} \cdot \left( L + \frac{L}{k} \right) = \frac{L}{2k} + \frac{L}{2k^2} \leq \frac{L}{k}$$

which completes the induction and proves the theorem.  $\square$

## 5 Simultaneous Approximation of Response Time Metrics

The algorithms presented in Section 4 seek to optimize a single objective: maximum response time. In some settings, it may be desirable to have an algorithm that optimizes several objectives simultaneously. In this section, we show that we can combine our results with prior work to achieve algorithms which are competitive for both maximum and average response time simultaneously. As above, we study two different settings for this problem: one with general demands splittable jobs, and the other with unit demands and unsplittable jobs.

**5.1 Splitting Schedules for General Demands.** An easy corollary of Theorem 2 and a result of Dinitz-Moseley is an existence of an algorithm with resource augmentation  $3 + \varepsilon$  that is  $(2/\varepsilon)$ -competitive for maximum response time and  $(1/\varepsilon^2)$ -competitive for average response time. The algorithm simply runs each individual algorithm in parallel and sums their resource augmentation.

**COROLLARY 5.1.** (COROLLARY TO THEOREM 2 AND THEOREM IV.14 IN [2]) *There exists an algorithm with resource augmentation  $3 + \varepsilon$  that is  $O(1/\varepsilon^2)$ -competitive for average response time, and is  $2/\varepsilon$ -competitive for maximum response time.*

*Proof.* In each round, use  $1 + \varepsilon/2$  resource to execute Proportional Allocation algorithm and use  $2 + \varepsilon/2$  resource to execute the Shortest Job First algorithm.  $\square$

We show, however, that by itself Proportional Allocation with any amount of resource augmentation performs poorly for average response time, while Shortest Job First performs poorly for maximum response time if resource augmentation is at most 2.

**LEMMA 5.1.** *Proportional Allocation with resource augmentation  $\alpha$  has competitive ratio  $\Omega(m/\alpha)$  for average response time, where  $m$  is the number of requests, for any  $\alpha > 0$ .*

*Proof.* Consider the following family of instances. At time 1,  $k$  unit requests and one request with demand  $M$  all arrive on a unit capacity node  $i$  (no other jobs arrive and the jobs' other endpoints are irrelevant). Consider the schedule which executes all unit requests in the first  $k$  rounds, and then schedules the size  $M$  request in the following  $M$  rounds. In this case, the sum of response times is  $1 + 2 + \dots + k + (M + k) \leq M + k^2$ .

On the other hand, Proportional Allocation with  $\alpha$  resource augmentation schedules a  $\alpha/(M + k)$  fraction of each request in each round until all requests are completed. In this case, all requests are completed in the same round no earlier than  $(M + k)/\alpha$ . Therefore, the sum of response times is at least  $(M + k)(k + 1)/\alpha \geq Mk/\alpha$ .

Setting  $M = k^2$ , the competitive ratio is at least  $\Omega(k/\alpha)$ . Since  $m = k + 1$ , this proves the result.  $\square$

**LEMMA 5.2.** *Shortest Job First with resource augmentation 2 has competitive ratio  $\Omega(m)$  for max response time, where  $m$  is the number of requests.*

*Proof.* Consider the following instance over four unit capacity nodes  $a, b, c, d$  as shown Figure 4. In all rounds  $t = 1, 2, \dots, T$ , if  $t$  is odd then two unit jobs arrive on nodes  $\{a, b\}$ , and if  $t$  is even then two unit jobs arrive on nodes  $\{c, d\}$ . Additionally, one request  $j^*$  arrives on nodes  $\{b, d\}$  with  $d_{j^*} = 2$  in round 1.

It is straightforward to give a schedule with maximum response time 4. First execute the job  $j^*$  over rounds 1 and 2, then schedule the remaining jobs in FIFO order. All jobs on  $\{a, b\}$  are independent of jobs on  $\{c, d\}$ , so

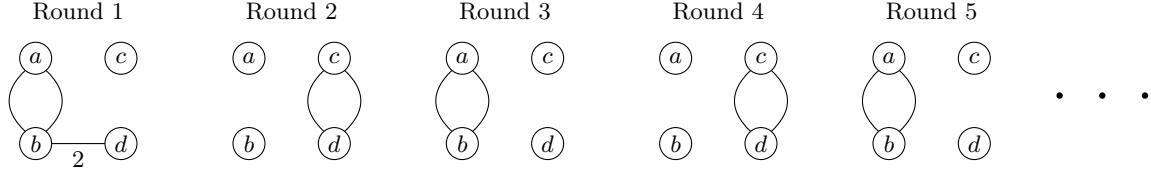


Figure 4: Job arrival sequence for lower bound on max response time for Shortest Job First with 2 resource augmentation. The request that arrives on nodes  $b$  and  $d$  in round 1 has demand 2. All other requests have unit demand. All nodes have unit capacity.

we can execute them in parallel. So, in every round  $t \geq 3$  we execute one job adjacent to all nodes and in any interval (not including round 1) of length 2 or greater the number of nodes that arrive on any port is at most the length of the interval. Therefore, all jobs  $j \neq j^*$  are completed in time  $r_j + 4$ .

On the other hand, even with a resource augmentation factor of 2, Shortest Job First never schedules the request  $j^*$  until round  $T + 1$ . With resource augmentation of 2 (or less), Shortest Job First always gives preference to the newer, unit size jobs that arrive in each round on either node  $b$  or node  $d$ , and scheduling these jobs uses all the (augmented) capacity of the node. Therefore, Shortest Job First has competitive ratio  $\Omega(T) = \Omega(m)$  for maximum response time with resource augmentation of 2 or less.  $\square$

The lemma implies that, given the same resources, Shortest Job First cannot match the competitiveness of Proportional Allocation with respect to maximum response time. However, it remains open whether Shortest Job First with  $2 + \varepsilon$  resources achieves any well-bounded competitive ratio for maximum response time.

**5.2 Nonsplitting Schedules for Unit Demands.** In the case of nonsplitting schedules for unit demands, we show that the simple FIFO algorithm given in Algorithm 5, with a resource augmentation factor of  $(2 + k)$  is simultaneously competitive for the average response time and maximum response time metrics, for any positive integer  $k$ . We begin by analyzing FIFO under the maximum response time metric.

---

**Algorithm 5:** FIFO Maximal Matching with a resource augmentation factor of  $(2 + k)$

---

**Data:** augmentation parameter  $k \in \{1, 2\}$

1  $P \leftarrow \emptyset$  // the set of pending requests

2 **for** each round  $t = 1, 2, 3, \dots$  **do**

3      $P \leftarrow P \cup \{j : r_j = t\}$

4     **for** each request  $j = \{i_1, i_2\}$  in  $P$  in release time order **do**

5         **if** the load on  $i_1$  is less than  $(2 + k)c_{i_1}$  and the load on  $i_2$  is less than  $(2 + k)c_{i_2}$  **then**

6             schedule  $j$  and  $P \leftarrow P \setminus \{j\}$

---

**LEMMA 5.3.** *For maximum response time metric, with a resource augmentation of a factor of  $(2 + k)$ , FIFO Maximal Matching is  $\max\{2/k, 1\}$ -competitive.*

*Proof.* We restrict our attention to  $k \in \{1, 2\}$  since the claim for  $k > 2$  is subsumed by the case  $k = 2$ . For ease of notation, let  $\gamma = 2/k$ . Since  $k \in \{1, 2\}$ , we have that  $\gamma \in \{1, 2\}$ .

We prove by induction on  $t$  that in any round  $t + \gamma L$  there is no pending request that arrives more than  $\gamma L$  rounds earlier, where  $L$  is the interval lower bound give in Equation 4.1. The base case when  $t = 0$  is trivially true. For the induction step, suppose that the claim holds for all  $t'$  such that  $0 \leq t' \leq t$ . Suppose, for the sake of contradiction, that there exists a request  $j^*$  on nodes  $i$  and  $i'$  such that  $j^*$  is pending in round  $t + \gamma L$  and  $r_{j^*} \leq t$ . By definition of the algorithm, it must be the case that in each round  $t \in \{t, t + 1, \dots, t + \gamma L\}$ , either  $i$  or  $i'$  is at full capacity. So we can infer that either  $i$  or  $i'$  is at full capacity for at least half these rounds. Without loss of generality, we assume it is  $i$ .

Given the resource augmentation, the total load executed on  $i$  in the interval  $\{t, t + 1, \dots, t + \gamma L\}$  is at least  $(2 + k) \cdot (\gamma L/2) = \gamma L + L$  by definition of  $\gamma$ . Consider the set of jobs  $J^*$  that comprise this load. Because FIFO

Maximal Matching orders requests by their arrival time, we have that  $r_j \leq r_{j^*} \leq t$  for all  $j \in J^*$ . By the inductive hypothesis, we infer that all  $j \in J^*$  complete within  $\gamma L$  of their release time. Since these jobs are pending at time  $t$  or later, we have that  $r_j \geq t - (\gamma L - 1)$  for all  $j \in J^* \cup \{j^*\}$ . Therefore, the number of jobs released on node  $i$  in rounds  $\{t - (\gamma L - 1), t - (\gamma L - 2), \dots, t\}$  is at least  $\gamma L + L + 1$ . However, this contradicts the definition of  $L$ , and so completes the induction.

Therefore, the maximum response time of FIFO Maximal Matching is  $\gamma L = (2/k)L$ . Since the optimal response time is at least  $L$  and no algorithm has maximum response time better than 1, FIFO Maximal Matching achieves a competitive ratio of  $\max\{2/k, 1\}$ .  $\square$

Moreover, we know from Theorem IV.5 of [2] that with resource augmentation  $(2 + k)$ , FIFO is  $2(2 + k)/k$ -competitive for average response time. Putting this result together with Lemma 5.3, we obtain that with a resource augmentation of  $(2 + k)$  for positive integer  $k$ , FIFO is simultaneously  $2(2 + k)/k$ -competitive under the average response time metric and  $2/k$ -competitive under the maximum response time metric.

## 6 Discussion and Open Problems

In this paper, we have shown a linear lower bound on the competitiveness of any ECFS schedule for the maximum response time metric, without resource augmentation. We then derived improved tradeoffs between resource augmentation and competitive ratio for maximum response time, for different problem settings (general vs unit case, splittable vs unsplittable jobs) using simple algorithmic approaches. Finally, we explored simultaneous approximation of average and response time metrics and proposed a hybrid algorithm based on Proportional Allocation and Shortest Job First for general demands and FIFO for unit case.

A number of open problems remain. First, we would like to find the best tradeoffs possible between resource augmentation and competitive ratio for response time metrics. Toward this end, we need to derive general lower bounds under resource augmentation. Of special interest is whether a bounded competitive ratio can be achieved for average response time with augmentation less than two. Second, deriving improved algorithms and bounds for simultaneous approximation of response time metrics is of practical interest. A natural direction in this regard is to consider  $\ell_p$  norms of response time metrics, which would provide a spectrum with average response time ( $p = 1$ ) and maximum response time ( $p = \infty$ ) at each end. We have been able to show that the results of [2] for Shortest Job First and FIFO qualitatively extend to  $\ell_p$  norms, for finite  $p$ , implying that the simultaneous approximations established in Section 5 generalize to  $\ell_p$  norms of response times.

## Acknowledgments

We thank Michael Dinitz for helpful discussions on the different models and problem formulations. This work was partially supported by NSF grant CCF-1909363.

## References

- [1] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
- [2] Michael Dinitz and Benjamin Moseley. Scheduling for weighted flow and completion times in reconfigurable networks. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 1043–1052, 2020.
- [3] G. Even, M. Halldórsson, L. Kaplan, and D. Ron. Scheduling with conflicts: online and offline algorithms. *Journal of Scheduling*, 12:199–224, 2009.
- [4] 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.
- [5] Long Gong, Paul Tune, Liang Liu, Sen Yang, and Jun (Jim) Xu. Queue-proportional sampling: A better approach to crossbar scheduling for input-queued switches. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(1), June 2017.
- [6] Dan Guez, Alex Kesselman, and Adi Rosén. Packet-mode policies for input-queued switches. In *Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '04, page 93–102, New York, NY, USA, 2004. Association for Computing Machinery.
- [7] S. Irani and V. Leung. Scheduling with conflicts on bipartite and interval graphs. *Journal of Scheduling*, 6:287–307, 2003.
- [8] Sandy Irani and Vitus Leung. Probabilistic analysis for scheduling with conflicts. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '97, page 286–295, USA, 1997. Society for Industrial and Applied Mathematics.
- [9] Hamidreza Jahanjou, Rajmohan Rajaraman, and David Stalf. Scheduling flows on a switch to optimize response times. In Christian Scheideler and Michael Spear, editors, *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 305–315. ACM, 2020.
- [10] Su Jia, Xin Jin, Golnaz Ghasemiefteh, Jiaxin Ding, and Jie Gao. Competitive analysis for online scheduling in software-defined optical wan. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pages 1–9, 2017.
- [11] Samir Khuller and Manish Purohit. Brief announcement: Improved approximation algorithms for scheduling co-flows. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '16, page 239–240, New York, NY, USA, 2016. Association for Computing Machinery.
- [12] Janardhan Kulkarni, Stefan Schmid, and Paweł Schmidt. Scheduling opportunistic links in two-tiered reconfigurable datacenters. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, page 318–327, New York, NY, USA, 2021. Association for Computing Machinery.
- [13] Henry Martyn Mulder. Julius Petersen’s theory of regular graphs. *Discrete Mathematics*, 100:157–175, 05 1992.
- [14] Julius Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 15:193–220, 1891.
- [15] 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.
- [16] Devavrat Shah and Jinwoo Shin. Randomized scheduling algorithm for queueing networks. *The Annals of Applied Probability*, 22(1):128–171, 2012.
- [17] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.