Maximizing Throughput in Multi-Queue Switches

Yossi Azar [*] Arik Litichevskey [†]

Abstract

We study a basic problem in Multi-Queue switches. A switch connects m input ports to a single output port. Each input port is equipped with an incoming FIFO queue with bounded capacity B. A switch serves its input queues by transmitting packets arriving at these queues, one packet per time unit. Since the arrival rate can be higher than the transmission rate and each queue has limited capacity, packet loss may occur as a result of insufficient queue space. The goal is to maximize the number of transmitted packets. This general scenario models most current networks (e.g., IP networks) which only support a "best effort" service in which all packet streams are treated equally. A 2-competitive algorithm for this problem was designed in [5] for arbitrary B. Recently, a $\frac{17}{9} \approx 1.89$-competitive algorithm was presented for $B > 1$ in [3]. Our main result in this paper shows that for B which is not too small our algorithm can do better than 1.89, and approach a competitive ratio of $\frac{e}{e-1} \approx 1.58$.

1 Introduction

Overview: Switches are a fundamental part of most networks. Networks use switches to route packets arriving at input ports to an appropriate output port, so that the packets will reach their desired destination. Since the arrival rate of packets can be much higher than the transmission rate, each input port is equipped with an incoming queue with bounded capacity. Considering that on the one hand, traffic in networks, which has increased steadily during recent years, tends to fluctuate, and on the other hand, incoming queues have limited space, packet loss is becoming more of a problem. As a result, over the recent years, considerable research has been carried out in the design and analysis of various queue management policies.

We model the problem of maximizing switch throughput as follows. A switch has m incoming FIFO queues and one output port. At each time unit, new packets may arrive at the queues, each packet belonging to a specific input queue. A packet can only be stored in its input queue if there is enough space. Since the m queues have bounded capacity, packet loss may occur. All packets have the same value, i.e. all packets are equally important. At each time unit, the switch can select a non-empty queue and transmit a packet from the head of that queue. The goal is to maximize the number of transmitted packets.

This model is worth studying, since the majority of current networks (most notably, IP networks) do not yet integrate full QoS capabilities. More specifically, the majority of current networks provide a "best effort" service, where packets belonging to different traffic streams are treated equally within intermediate switches. Traditionally, similar problems were analyzed while assuming

[*]`azar@tau.ac.il`. School of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel. Research supported in part by the Israeli Ministry of industry and trade and by the Israel Science Foundation.

[†]`litiche@tau.ac.il`. School of Computer Science, Tel-Aviv University, Tel-Aviv, 69978, Israel.

either some constant structure of the sequence of arriving packets, or a specific distribution of the arrival rates (see e.g. [8, 16]).

Recently, the above switching problem was studied while avoiding any a priori assumption on the input. In other words, at any point in time, packets arrive at an arbitrary subset of the input channels. We therefore use competitive analysis to evaluate the performance of online algorithms to the optimal off-line solution that knows the entire sequence in advance. Specifically, the competitive ratio of an online algorithm is the supermom, taken over all finite sequences, of the ratio between the online throughput and optimal throughput on the same input.

In [5] Azar and Richter showed that any deterministic work-conserving algorithm, i.e. transmits a packet in each unit of time if not all queues are empty, is 2-competitive. They also showed an $\frac{e}{e-1}$-competitive randomized algorithm. Recently, Albers and Schmidt [3] introduced a $\frac{17}{9} \approx 1.89$-competitive algorithm for $B > 1$. In this paper we show that for the case where B is larger than $\log m$ our algorithm approaches a competitive ratio of $\frac{e}{e-1} \approx 1.58$, which is significantly better than 1.89.

Our results:

- Our main contribution is an $\frac{e}{e-1} \approx 1.58$-competitive algorithm for the switch throughput problem, for any large enough B. Specifically, we show a *fractional* algorithm for the switch throughput problem, i.e. one that can insert *fractions* of packets into each queue if there is sufficient space and transmit a fraction from each queue of total of at most 1, with competitiveness of $\frac{e}{e-1}$. Then we transform our *fractional* algorithm into a discrete algorithm, i.e. one that can insert and transmit integral packets, with competitiveness of $\frac{e}{e-1}(1 + \frac{\lfloor H_m \rfloor + 1}{B})$, where $H_m = \sum_{i=1}^{m} \frac{1}{i} \approx \ln m$ is the mth harmonic number.
- We use two tools which are of their own interest :
  - We consider the online unweighted fractional matching problem in a bipartite graph. In this problem we have a bipartite graph $G = (R, S, E)$ where S and R are the disjoint vertex sets and E is the edge set. At step i, vertex $r_i \in R$ together with all of its incident edges, arrives online. The online algorithm can match fractions of r_i to various adjacent vertices $s_j \in S$. We prove an upper bound of $\frac{e}{e-1}$ on the competitive ratio of a natural online "water level" matching algorithm.
  - We present a generic technique to transform any *fractional* algorithm for the switch throughput problem into a discrete algorithm. Specifically, we show that given a c-competitive online fractional algorithm for the switch throughput problem, we can construct an online discrete algorithm with competitiveness of $c(1 + \frac{\lfloor H_m \rfloor + 1}{B})$, for the switch throughput problem.

Side results:

- We show a lower bound of $\frac{e}{e-1} - \Theta(\frac{1}{m})$ on the competitive ratio of any online unweighted fractional matching algorithm in a bipartite graph. Thus, we prove that our "water level" matching algorithm is optimal, up to an additive smaller than 2 for the size of the matching. In addition, this lower bound yields a lower bound of $\frac{e}{e-1} - \Theta(\frac{1}{m})$ on the competitive ratio of any randomized discrete algorithm for the online unweighted matching problem in a bipartite graph. Therefore, we slightly improve the asymptotic lower bound of Karp *et al.* [11] with a non-asymptotic lower bound.

- We consider the online b-matching problem. In the online b-matching problem we have a bipartite graph $G = (R, S, E)$, where $r_i \in R$ can be matched to a *single* adjacent vertex $s_j \in S$ such that every s_j can be matched up to b times. We introduce a more compact analysis for the upper bound of the competitive ratio obtained by algorithm *Balance* for the b-matching problem that was studied by Kalyanasundaram and Pruhs [10], using a similar technique to the one which was used in our analysis of the "water level" algorithm.

Our techniques: We start by studying the online fractional unweighted matching problem in a bipartite graph. We introduce a natural "water level" algorithm and obtain the competitiveness of $\frac{e}{e-1}$. We next construct an *online* reduction from the fractional switch throughput problem to the problem of finding a maximum fractional matching in a bipartite graph. Thus, we obtain a $\frac{e}{e-1}$-competitive algorithm, for the fractional switch problem, which emulates the fractional matching constructed in the bipartite graph. Next, we present a generic technique to transform any fractional algorithm for the switch throughput problem into a discrete algorithm. Specifically, we present an algorithm with queues larger than B that maintains a running simulation of the fractional algorithm in a relaxed fractional model and transmits from the queue with the largest overload with respect to the simulation. We then transform this algorithm into an algorithm with queues of size B.

Related results for the switch throughput problem with unit value: The online problem of maximizing switch throughput has been studied extensively during recent years. For the unit value scheduling, Azar and Richter [5] showed that any deterministic work-conserving algorithm, i.e. one that transmits a packet in each unit of time if not all queues are empty, is 2-competitive. In addition they showed a lower bound of $2 - \frac{1}{m}$ for the case where $B = 1$. For arbitrary B they showed a lower bound of $1.366 - \Theta(\frac{1}{m})$. They also considered randomized online algorithms and presented an $\frac{e}{e-1}$-competitive randomized algorithm. For $B = 1$, they showed a lower bound of $1.46 - \Theta(\frac{1}{m})$ on the performance of every randomized online algorithm. Recently, a $\frac{17}{9} \approx 1.89$-competitive deterministic algorithm was presented by Albers and Schmidt [3], for $B \geq 2$. For the case $B = 2$ they showed that their algorithm is optimal. They also showed that any greedy algorithm is at least $2 - \frac{1}{B}$-competitive for any B and large enough m. In addition, they showed a lower bound of $\frac{e}{e-1}$ on the competitive ratio of any deterministic online algorithm and a lower bound of 1.4659 on the performance of every randomized online algorithm, for any B and large enough m. We note that the lower bound obtained by Albers and Schmidt does not apply to $B > \log m$ considered in our paper.

Related results for the switch throughput problem with values: Additional results are known when packets have values and the goal is to maximize the total value of the transmitted packets; such a model corresponds to switches supporting QoS. Most of the previous work focused on a single queue problem, i.e. $m = 1$. Aiello *et al.* [2] initiated the study of different queuing policies for the 2-value non-preemptive model in which the switch has a single queue, preemption is not allowed and each packet has a value of either 1 or α. Recently, Andelman *et al.* [4] showed tight bounds for this case. The preemptive 2-value single-queue model was initially studied by Kesselman and Mansour [13], followed by Lotker and Patt-Shamir [15] who showed almost tight bounds. The general preemptive single-queue model, where packets can take arbitrary values, was investigated by Kesselman *et al.* [12], who proved that the natural greedy algorithm is 2-competitive (specifically $2\alpha/(1 + \alpha)$-competitive, where $\alpha \geq 1$ is the ratio between the largest value and the smallest one). Azar and Richter [5] presented a general technique which transforms any admission control c-competitive strategy for a single queue (both preemptive and non-preemptive) into a $2c$-competitive algorithm (preemptive or non-preemptive, respectively) for m queues. Thus, applying the result of Kesselman *et al.* [12], they derived a 4-competitive algorithm for the general preemptive model, and by applying the result of Andelman *et al.* [4], they derived a $(2e\lceil \ln \alpha \rceil)$-

competitive algorithm for the general non-preemptive model. Recently, a 3-competitive algorithm was given for the preemptive case in [6].

Related results for the cost version: Koga [14] and Bar-Noy *et al.* [7] investigated the online problem of minimizing the length of the longest queue in a switch. In their model, queues are unbounded in size, and hence packets are not lost. Koga [14] proved that the natural greedy algorithm that always empties the longest queue is $\Theta(\log m)$-competitive. Bar-Noy *et al.* [7] suggested a different algorithm that simulates the greedy algorithm in the continuous model and is also $\Theta(\log m)$-competitive. Chrobak *et al.* [9] studied the more general problem of minimizing the length of the longest queue in a continuous model where queues can be emptied subject to conflict constraints.

Related results for the online unweighted matching problem in a bipartite graph: In the online unweighted matching problem we have a bipartite graph $G = (R, S, E)$ where S and R are the disjoint vertex sets and E is the edge set. At step i, vertex $r_i \in R$ together with all of its incident edges, arrives online. The online algorithm can match vertex $r_i \in R$ to an adjacent vertex $s_i \in S$, and the goal is to maximize the number of matched vertices. Karp *et al.* [11] observed that in the integral unweighted matching problem in a bipartite graph any deterministic algorithm, which never refuses to match if possible, is 2-competitive and no deterministic algorithm can be better than 2-competitive. In addition, they introduced a randomized algorithm $RANKING$ with a competitive ratio of $\frac{e}{e-1} - o(1)$ and proved a lower bound of $\frac{e-1}{e} - o(1)$, thus obtaining the optimality of $RANKING$, up to lower order terms. We improve the lower order terms in their lower bound. The online unweighted matching problem was generalized by Kalyanasundaram and Pruhs in [10] to the b-matching problem. In this model each vertex $r_i \in R$ can be matched to a *single* adjacent vertex $s_j \in S$ such that every s_j can be matched up to b times. They showed an online algorithm *Balance* with competitiveness of $\frac{(1+\frac{1}{b})^b}{(1+\frac{1}{b})^b-1}$. They also proved that any deterministic algorithm is at least $\frac{(1+\frac{1}{b})^b}{(1+\frac{1}{b})^b-1}$-competitive, thus proving the optimality of *Balance* .[1]

Paper structure: Section 2 includes formal definitions and notation. In Section 3 we consider the online unweighted fractional matching and obtain an upper bound for a natural online algorithm. We address the fractional version of the switch throughput problem in Section 4 and obtain an upper bound for this problem. In Section 5 we present our general technique for the discretization of any fractional algorithm for the switch throughput problem and obtain an upper bound for the outcome algorithm. Open problems are presented in Section 6.

2 Problem Definition and Notations

We model the switch throughput maximization problem as follows. We are given a switch with m FIFO input queues, where queue i has *size* B_i, and one output port. Packets arrive online at the queues, every packet belonging to a specific input queue. All packets are of equal size and value. We denote the online finite packet sequence by σ. Initially, all m queues are empty. Each time unit is divided into two phases: in the *arrival* phase a set of packets arrives at specific input queues and may be inserted into the queues if there is sufficient place. Remaining packets must be discarded. In the *transmission* phase the switching algorithm may select one of the non-empty queues, if such exists, and transmit the packet at the head of that queue. The goal is to maximize the number

[1] In [10] and [11] the authors used, for defining the competitive ratio, the inverse ratio, which is at most 1. Specifically, they define the competitive ratio as the supremum over all finite sequences σ, of the ratio of the cardinality of the matching constructed by the online algorithm divided by the max cardinality of σ.

of transmitted packets. We consider the non-preemptive model, in which stored packets cannot be discarded from the queues. We note that in the unit value model the admission control question, i.e which packet to accept, is simple, since there is no reason to prefer one packet over the other. Therefore, it is enough to focus on the scheduling problem.

For a given time t, we use the term *load* of queue i to refer to the number of packets residing in that queue, at that time. For a given time t, the term *space* of queue i is used to refer to its size minus its load, i.e. how many additional packets can be assigned to queue i, at that time. Given an online switching algorithm A we denote by $A(\sigma)$ the value of A given the sequence σ. We denote the optimal (offline) algorithm by OPT, and use similar notation for it. A deterministic online algorithm A is *c-competitive* for all maximization problems described in this paper (the online unweighted fractional matching problem in a bipartite graph, online b-matching problem in a bipartite graph, and switch throughput maximization problem fractional version as well as discrete version) iff for every instance of the problem and every packet sequence σ we have: $OPT(\sigma) \leq c \cdot A(\sigma)$.

We prove our upper bound by assuming that all queues in the switch are of equal size B. We note that this is done for simplicity of notation only. The algorithms we present remain the same when queues have different sizes, where in our upper bound, B will be replaced by $\min_i\{B_i\}$.

3 Online Unweighted Fractional Matching in a Bipartite Graph

Our switching algorithm is based on an online fractional matching algorithm. Thus we start by considering the online unweighted matching problem in a bipartite graph, which is defined as follows. Consider an online version of the maximum bipartite matching on a graph $G = (R, S, E)$, where S and R are the disjoint vertex sets and E is the edge set. We refer to set R as the requests and refer to set S as the servers. The objective is to match a request to a server. At step i, vertex $r_i \in R$ together with all of its incident edges, arrives online. The response of A can be either to reject r_i, or to irreversibly match it to an unmatched vertex $s_j \in S$ adjacent to r_i. The goal of the online algorithm is to maximize the size of the matching.

The fractional version of the online unweighted matching is as follows: Each request r_i has a size x_i which is the amount of work needed to service request r_i. Algorithm A can match a fraction of size $k^i_j \in [0, x_i]$ to each vertex $s_j \in S$ adjacent to r_i. If request i is matched partially to some server j with weight k^i_j then $\sum_{j=1}^m k^i_j \leq x_i$ and we have to maintain that the load of each server j, which is $\sum_{i=1}^n k^i_j$ where n is the length of σ, is at most 1. We use the terms *match* and *assign* interchangeably. The goal of the online algorithm is to maximize the sum of fractions matched. More formally, the goal is to maximize $\sum_{i,j} k^i_j$.

Definition 3.1 *We use the term* load *of server j to refer to the sum of fractions that was matched to a server s_j and denote it by $l_j = \sum_{i=1}^n k^i_j$, where n is the length of σ.*

Definition 3.2 *The* size of the fractional matching *M, denoted by $|M|$, is the total sum of the fractions that were matched, i.e. $\sum_{j=1}^m l_i$.*

We show that a natural "water level" algorithm WL for the fractional matching problem is $\frac{e}{e-1} \approx 1.58$-competitive, even against a fractional OPT. Intuitively, given a request $r_i \in R$, algorithm WL flattens the load on the minimum loaded servers adjacent to r_i.

Algorithm WL**:** For each request r_i, match a fraction of size k^i_j for each adjacent s_j, where $k^i_j = (h - l_j)_+$ and $h \leq 1$ is the maximum number such that $\sum_{j=1}^{m} k^i_j \leq x_i$. By $(f)_+$ we mean $\max\{f, 0\}$.

Theorem 3.1 *Algorithm WL is $\frac{e}{e-1} \approx 1.58$-competitive.*

Proof: We start by defining the following terms:

1. For any given algorithm let $v(h)$ be the total load up to height h, i.e. $\sum_{j=1}^{m} \min\{h, l_j\}$.
2. Let L_{OPT} be the size of the fractional matching obtained by OPT.
3. Let S_{OPT} be the set of the requests fractions that were assigned by OPT.
4. For any given algorithm A let $X_h \subseteq S_{OPT}$ be the set of the requests fractions that were assigned by OPT, and assigned by A above height h.
5. For any given algorithm A let $Y_h \subseteq S_{OPT}$ be the set of the requests fractions that were assigned by OPT, and rejected by algorithm A.

Consider the total load assigned to the servers by WL. We divide the load assigned to the servers over some height h. Let k be the total size of fractions in $Y_h \cup X_h$, i.e. the total size of requests fractions that were assigned by OPT, and assigned by WL above height h or rejected. Clearly, $k \geq L_{OPT} - v(h)$. From volume preservation, this load was assigned to at least $\lceil k \rceil$ different servers by OPT. Hence, the size of the load assigned by WL to these k servers is at least h. Since OPT used these k servers, WL could also have used them, and thus part of the load assigned above height h could have been assigned below height h. Therefore we get :

$$\frac{dv}{dh} \geq L_{OPT} - v(h)$$

which is the same as

$$-\frac{dv}{L_{OPT} - v(h)} \leq -dh.$$

By integrating both sides

$$-\int_0^{h'} \frac{dv}{L_{OPT} - v(h)} \leq -\int_0^{h'} dh$$

which implies

$$\ln(L_{OPT} - v(h')) - \ln L_{OPT} = \ln(L_{OPT} - v(h))|_0^{h'} \leq -h|_0^{h'} = -h'.$$

Hence,

$$\frac{L_{OPT} - v(h')}{L_{OPT}} \leq e^{-h'}$$

and we get

$$v(h') \geq L_{OPT}(1 - e^{-h'}). \tag{1}$$

Since $L_{OPT} = OPT(\sigma)$ and $v(1) = WL(\sigma)$ we achieve $OPT(\sigma) \leq \frac{e}{e-1} \cdot A(\sigma)$. ■

We note that algorithm WL is optimally competitive, up to an additive smaller than 2 for the size of the matching. The proof is given in Appendix A.

Theorem 3.2 *Algorithm WL is $\frac{e^\alpha}{e^\alpha-1}$-competitive in a resource augmentation model, where the online algorithm can match α times more than the optimum algorithm on a single server.*

Proof: The proof follows immediately from the fact that Equation (1) holds in the resource augmentation model. ■

Remark 1 *Our technique can be extended for analyzing the b-matching problem that was studied in [10]. Specifically, we simplify the proof of the competitiveness of algorithm Balance for the b-matching problem which was studied by Kalyanasundaram and Pruhs [10]. See Appendix B for details.*

4 The Maximizing Switch Throughput Problem - The Fractional Version

In this section we consider a fractional model, which is a relaxation of the discrete model which was presented in Section 2. In the fractional model we allow the online algorithm to transmit fractional packets as well as to accept fractional packets into the queues. More formally, each time unit t is divided into two phases: in the arrival phase a set of packets arrives at the queues. *Fractions* of packets can be inserted into each queue if there is sufficient space. The remaining fractions of packets must be discarded. In the transmission phase of time t, the switching algorithm may select a set of non-empty queues, if such exists, and transmit fractional packets from the head of each queue s.t. the sum of transmitted fractions is at most 1. We assume that sequence σ consists of *integral* packets. However, this restriction is not obligatory for this section. The restriction is relevant for the transformation of a fractional algorithm for the switch throughput problem into a discrete scheduling algorithm. We focus on algorithms that accept packets or fractional packets if sufficient place exists; since all packets have unit values there is no reason to prefer one packet over another. We refer to such algorithms as greedy admission control algorithms, and therefore focus on the scheduling problem alone. We show that a fractional algorithm for the scheduling problem is $\frac{e}{e-1} \approx 1.58$-competitive, even against a fractional adversary. We begin by introducing a translation of our problem (the fractional model) into the problem of online unweighted fractional matching in a bipartite graph.[2]

Given a sequence σ, we translate it into the bipartite graph $G^\sigma = (R, S, E)$, which is defined as follows:

- Let T denote the latest time unit in σ in which a packet arrives. We define the set of *time nodes* as $R = \{r_1, \ldots, r_{T+mB}\}$. Each $r_i \in R$ has unit size, i.e. $x_i = 1$ for each $1 \leq i \leq T+mB$
- Let P be the total number of packets specified in σ. We define the set of *packet nodes* as $S = \{s_1, \ldots, s_P\}$.
- Let P_i^t denote the set of the last B packets that arrive at queue q_i until time t (inclusive). Define $P^t = \bigcup_{i=1}^m P_i^t$. We define the set of edges in G^σ as follows: $E = \{(r_t, s_p) | p \in P^t\}$.

[2] A similar translation was presented in [5] for integral scheduling.

Note that in G^σ, time and packets nodes arrive in an online fashion. Thus the problem is how to divide the time among the packets, where edges connect each of the time nodes to the packet nodes which correspond to packets that can be resident in some queue in time t. For consistency with Section 3 we denote the *time nodes* by **requests** and the *packet nodes* by **servers**.

Remark 2 *We note that in contrast to the matching model that was presented in Section 3, here the servers as well as the requests arrive online. We emphasize that this does not change the results obtained in Section 3 since in G^σ the servers which arrive at time t only connect to request $r_{t' \geq t}$ and thus can be viewed as servers which were present but not yet adjacent to requests $r_{t'<t}$.*

Before we proceed we introduce some new definitions.

Definition 4.1 *A* schedule *SC for a sequence of arriving packets σ, for the switch throughput problem, is a set of triplets of the form (t, q_i, k_i^t) for each $1 \leq i \leq m$, where queue q_i is scheduled for the transmission of a fraction of size $k_i^t \geq 0$ at time t. The* size of the schedule*, denoted by $|SC|$, is the total amount of fractions scheduled for transmission, i.e. $\sum_{t,i} k_i^t$.*

Definition 4.2 *A schedule SC for a sequence σ is called* legal *if for every triplet (t, q_i, k_i^t), queue q_i has a load of at least k_i^t at time t and $\sum_{i=1}^{m} k_i^t \leq 1$.*

The following lemmas connect bipartite fractional matching to our problem.

Lemma 4.1 *Every legal fractional schedule SC for the sequence σ can be mapped, in an online fashion, to a fractional matching M in G^σ such that $|SC| = |M|$.*

Proof: Let SC be a legal schedule for σ. We construct the desired matching M incrementally while moving ahead in time. Given a time t, for each $1 \leq i \leq m$ we have $(t, q_i, k_i^t) \in SC$. We match a fraction of size $\min\{k_i^t, 1 - l_j\}$ from request r_t to server s_j, where $j = \min\{1 \leq k \leq P \,|\, s_k \in P_i^t, l_k < 1\}$ and l_j is the load on server j. If k_i^t is not matched completely on s_j we match the rest of k_i^t to a new server s'_j, where $j' = \min\{1 \leq k \leq P \,|\, s_k \in P_i^t, l_k < 1\}$. Note that $j' \neq j$ since $l_j = 1$. Simple induction proves that for each time t and queue q_i the sum of unmatched fractions of servers in P_i^t is equal to the total load of queue q_i at time t according to SC. Hence, every fraction transmitted can be mapped to a fraction matched in M. Clearly, by the construction, $|SC| = |M|$. ■

Lemma 4.2 *Every matching M in G^σ can be translated, in an online fashion in polynomial time, to a legal schedule SC for σ such that $|SC| = |M|$, while performing greedy admission control.*

Proof: Let M be a matching in G^σ. We construct a legal schedule SC for σ incrementally, while going over the requests in R, starting from r_1. Let request r_t be fractionally matched in M to subset S_t of servers belonging to P^t. We define a set $I = \{i | P_i^t \cap S_t \neq \phi\}$. For each $i \in I$ we define k_i^t to be the total sum of fractions matched from request r_t to $s_j \in P_i^t$, and for each $1 \leq i \leq m$ such that $i \notin I$ we define $k_i^t = 0$. We add the triplets (t, q_i, k_i^t) to the schedule SC. Simple induction shows that for every i and t, the total load of servers in P_i^t is at most the load of queue i at time t according to SC. Therefore, we can always translate matched fractions in M to a fraction transmission in SC, and our obtained schedule is legal. Clearly, this translation takes polynomial time and, by the construction, $|SC| = |M|$. ■

The following corollaries directly result from Lemmas 4.1 and 4.2.

Corollary 4.3 *For any sequence σ, an optimal fractional schedule can be found (offline) in polynomial time.*

Corollary 4.4 *For any sequence σ, the size of the optimal fractional schedule for σ is equal to the size of a maximum fractional matching in G^{σ}.*

Remark 3 *Actually, maximum fractional matching is equal to integral maximum matching, since matching corresponds to a totally unimodular matrix.*

Now, we present a fractional scheduling algorithm EP for maximizing switch throughput in the relaxed model. Since the bipartite unweighted fractional matching problem is connected to the maximizing switch throughput problem, algorithm EP intuitively bases its scheduling decisions on the matching constructed by algorithm WL, which was presented in Section 3, in the online constructed graph G^{σ}.

Algorithm EP:

- Maintain a running simulation of WL in the online constructed graph G^{σ}.
- **Admission control**: perform greedy admission control.
- **Scheduling**: transmit the total size of the fractions that were matched from r_t to servers in P_i^t from the head of queue i.

Remark 4 *We note that algorithm EP actually constructs a legal schedule for σ incrementally as shown by Lemma 4.2.*

Theorem 4.5 *For every sequence σ, $OPT(\sigma) \leq \frac{e}{e-1} EP(\sigma)$.*

Proof: The theorem follows immediately from Corollary 4.4, Remark 4 and Theorem 3.1. ∎

5 The Maximizing Switch Throughput Problem

In this section we consider the maximizing switch throughput problem. Given a sequence σ which consists of *integral* packets, we present a generic technique to transform any c-competitive online fractional algorithm for the switch throughput problem into a discrete algorithm with a competitive ratio of $c(1 + \frac{\lfloor H_m \rfloor + 1}{B})$, where $H_m = \sum_{i=1}^{m} \frac{1}{i} \approx \ln m$ is the mth harmonic number. In particular, we take the fractional scheduling algorithm EP, which was presented in Section 4, with a competitive ratio of $\frac{e}{e-1}$ and transform it into an online discrete scheduling algorithm with a competitive ratio of $\frac{e}{e-1}(1 + \frac{\lfloor H_m \rfloor + 1}{B})$. Thus, the competitive ratio of the discrete algorithm asymptotically approaches $\frac{e}{e-1} \approx 1.58$ for large size queues B. We start by defining the cost scheduling problem in the next subsection.

5.1 The cost scheduling problem

In this subsection we introduce a general technique for the discretization of the scheduling of any fractional algorithm A, for the *cost* version. We will use this technique later for the discretization of the scheduling of any fractional algorithm A, for the *throughput* problem. Specifically, we will use this technique for the discretization of algorithm EP, which was presented in Section 4.

Consider the switch throughput problem, but assume that queues are unbounded. Hence all packets are accepted into a queue. Evidently, the maximum throughput problem is degenerate in this scenario, since no packet loss may occur. In this case, one can consider the minimization of the maximum queue size. More formally, we are given a switch with m FIFO queues, where queues have unbounded size, and one output port. Packets arrive online; each packet is destined for one of the queues. Initially, all m queues are empty. We denote the online finite packet sequence by σ. Each time unit is divided into two phases: in the arrival phase a set of packets arrives at the queues. Packets **must** be inserted by the switching algorithm into each queue. In the transmission phase, the switching algorithm may select one of the non-empty queues, if it exists, and transmit the packet at the head of that queue. We define the cost of the online algorithm A given a finite sequence of packets σ, as the maximum length of a queue taken over all queues and times. The goal is to minimize the maximum cost, i.e. minimize the maximum queue size over time.

We now turn to consider a relaxation of the model and allow an online algorithm to transmit fractional packets waiting at the head of different non-empty queues, if they exist, provided that the total size of the transmitted fractions in one unit of time does not exceed 1. We denote by A the online fractional algorithm. We now introduce a general technique for the discretization of the scheduling of algorithm A given a finite sequence of *integral* packets σ while adding an additive factor of at most H_m to the cost of A. We start with the following definition and lemmas.

Definition 5.1 *An online fractional algorithm A is* work-conserving *if in each unit of time it transmits a total size of* 1 *if it exists, and transmits the total load if not.*

Observation 5.1 *Clearly, given a finite sequence of* integral *packets σ, any online work-conserving algorithm will transmit a total size of* 1 *in each unit of time if not all queues are empty.*

Lemma 5.1 *Every algorithm A can be transformed into a work-conserving algorithm A' while not worsening the performance of any sequence (in particular, $c_{A'} \leq c_A$, where $c_{A'}$, c_A are the competitive ratios of A' and A, respectively).*

Proof: Consider the following transformation of algorithm A into A'.

Algorithm A': Maintain a running simulation of algorithm A. At each time unit t for each queue $1 \leq i \leq m$ transmit a fraction of size k_i^t as A if possible. If the total size of the fractions transmitted by A', at time unit t, is smaller than 1, i.e. $\sum_{i=1}^{m} k_i^t < 1$, choose some subset of queues with a total load of at least $k = 1 - \sum_{i=1}^{m} k_i^t$ and transmit total size of k in some arbitrary fashion.

By a simple induction on time, it is straightforward to prove the following claim:

Claim 5.2 *Given a time unit t for each $1 \leq i \leq m$, let l_i^A and $l_i^{A'}$ be the loads of queue i as seen by algorithms A and A' at that time, respectively. Then for each time unit t, $l_i^A \geq l_i^{A'}$.*

The proof of Lemma 5.1 follows directly from Claim 5.2 ■

Henceforth we will deal exclusively with work-conserving algorithms, even when we neglect to say so explicitly. Now, we define algorithm M which gets an algorithm A as a parameter and denote it by M^A. At each time unit, in order to decide which queue to serve, M^A computes the load seen by A and uses this information to make its decision. Given a time unit t, let l_i^A and l_i be the simulated load of A and the actual load of M^A on queue i at that time, respectively. The *residual load* of queue i at time unit t is defined as $l_i^{res} = l_i - l_i^A$. Intuitively, algorithm M^A transmits at each time unit from a queue with maximum residual load. Before we present the exact definition of algorithm M^A we define the following:

Definition 5.2 *The* mid-state *of each time unit is the state of the queues after algorithm A transmits its fractional packets from some queues, and just before algorithm M^A transmits its packet from some queue.*

Algorithm M^A: Maintain a running simulation of algorithm A. At each time unit transmit a packet from a queue which has the maximum residual load at the mid-state (breaking ties arbitrarily), unless all queues are empty.

Theorem 5.3 *The cost of M^A is at most the cost of A plus H_m.*

Before we prove Theorem 5.3 we start with some definitions, observations and lemmas.

Observation 5.2 *After the arrival phase as well as after the transmission phase,* $\sum_{i=1}^m l_i^{res} = 0$.

Proof: The proof follows immediately from the fact that σ consists of integral packets, and thus both simulation A and algorithm M^A transmit the same total volume and receive the same amount of packets. ■

Lemma 5.4 *At the beginning of each time unit, $l_i^{res} > -1$ for each queue.*

Proof: The proof is by induction on time t. For $t = 0$ the claim is trivial. Suppose the claim is true for $t-1$ and consider t. From Observation 5.2 after the arrival phase, $\sum_{i=1}^m l_i^{res} = 0$. Since A is work-conserving, from Observation 5.1 at the mid-state $\sum_{i=1}^m l_i^{res} = 1$. Thus, there is a queue with a positive residual load. Hence, M^A will transmit from some queue with positive residual load. The fact that the residual load may decrease by at most 1 for each queue, implies that $l_i^{res} > -1$ for each queue. ■

Corollary 5.5 *At the mid-state there is always a queue with a positive residual load, provided that not all queues are empty.*

Remark 5 *Note that we actually proved Lemma 5.4 for any algorithm which transmits packets from any queue with a positive residual load. From Corollary 5.5 such a queue always exists, unless all queues are empty.*

We first show a weaker bound of Theorem 5.3.

Lemma 5.6 *The cost of M^A is at most the cost of A plus $m-1$.*

Proof: Let t denote the time unit by which the maximum additive factor is achieved. Let j be the queue which is maximizing $l_i - l_i^A$. Consider the arrival phase of t. As noted by Observation 5.2

$$0 = l_j - l_j^A + \sum_{i \neq j}(l_i - l_i^A).$$

Since by Lemma 5.4, $l_i^{res} > -1$, we conclude that $l_j - l_j^A \leq m - 1$. ■

We now want to tighten the bound on the cost of M^A and prove Theorem 5.3. This is done by using a theorem for the leaky-bucket problem which turns out to be the complement of our cost problem. We address the leaky-bucket problem as presented in [1]. The setting is a collection of m unbounded buckets initially filled with the same amount of material. By unbounded we mean that there is neither a finite bottom nor a finite end. During each time unit, an amount $k_i \in \mathbb{R}$ of material leaks from bucket i. The total volume leaking at each time unit is p. This amount of material is replaced by adding p refills of unit size into the buckets. More formally, let $B_1, B_2 \ldots B_m$ be a set of m buckets, having associated variables $L_1, L_2 \ldots L_m$, where $L_i \in \mathbb{R}$ denotes the amount of material held in B_i. Initially all $L_i = Z$. At each time unit an amount of $k_i \geq 0$ is reduced from L_i. The total amount of depletion is p, i.e. $\sum_{i=1}^{n} k_i = p$. The online algorithm can try to restore the amount of material of the buckets by adding p refills of size 1. From now on we consider the case $p = 1$.

In [1] Adler *et al.* have considered the natural scheduler S_0 which refills the emptiest bucket.

Scheduler S_0: Fill the emptiest bucket with an amount of 1.

We now present some of the results as presented in [1] for the case where $p = 1$.

Theorem 5.7 *Scheduler S_0 achieves an upper bound of $Z + 1$ on the maximum load of any bucket and an outcome of at least $Z - H_m$, against every adversary.*

We next want to relate the problem of the leaky bucket to the cost scheduling minimization problem. To do so, we define the term *negative-residual load.* The negative-residual load of queue i is defined as $Nl_i^{res} = -l_i^{res}$. Clearly, unless all queues are empty, algorithm M^A transmits a packet from the queue with the minimum negative residual load at the mid-state (breaking ties arbitrarily). We now return to the proof of Theorem 5.3.

Proof: We show a translation of our problem to the leaky-bucket problem. We put $Z = 0$ and allow bucket loads to become negative. We note that in [1] Adler *et al.* considered the case $Z = 0$ in the proof of the lower and upper bound. When algorithm A transmits a fraction of size k_i from queue i, the negative residual load decreases, and we interpret this as a depletion of k_i. When M^A transmits a packet from queue j, the residual load increases, and we interpret this as filling bucket j with one unit size. Note that the arrival of packets do not change the residual load, since both A and M^A accept the same load, as σ consists of integral packets. Thus, if we restrict our attention to the negative residual loads, clearly algorithm M^A acts on them exactly as scheduler S_0. From Theorem 5.7, scheduler S_0 achieves an outcome of at least $Z - H_m$ against every adversary. Thus, the negative residual load is at least $-H_m$. Since $Nl_{res} = -l_{res}$, the residual load of M^A is at most H_m. ■

Remark 6 *An alternative proof of a slightly weaker bound of $\lfloor \log_2 m \rfloor$ plus the cost of A can be obtained using [7]. Specifically, Bar-Noy* et al. *[7] showed that for a fractional version of algorithm* Longest Queue First*(LQF), denoted by ContinuousLQF, the cost of $M^{ContinuousLQF}$ is at most the cost of ContinuousLQF plus $\lfloor \log_2 m \rfloor$. A close examination of their proof indicates that the proof holds for every work-conserving algorithm.*

5.2 Discretization of the fractional scheduling

Given a finite sequence of *integral* packets σ we present a general technique to transform any c-competitive *fractional* algorithm for the switch throughput problem into a discrete algorithm with a competitive ratio of $c(1+\frac{\lfloor H_m \rfloor+1}{B})$. In particular, we will apply this technique for the discretization of algorithm EP, which was presented in Section 4. It can easily be shown that any algorithm A can be transformed into a greedy admission control algorithm A' while not worsening the performance of any sequence (in particular, $c_{A'} \leq c_A$, where $c_{A'}$, c_A are the competitive ratios of A' and A, respectively); we focus on greedy admission control algorithms.

For our discretization process we rely on the results of the cost problem, which were presented in Subsection 5.1. We want to address the packets which were accepted by EP as the input sequence σ for the cost problem studied in Subsection 5.1, in a manner which is yet to be seen. Recall that algorithm EP is a greedy admission control algorithm. Thus, EP might accept a fractional packets due to insufficient queue space. Since in the model of the cost problem we study the case where σ consists of integral packets we want EP to only accept packets integrally. Hence, we continue by considering the following problem: assume we are given an online c-competitive algorithm A with greedy admission control. We want to produce a competitive algorithm $\hat{A}$ which assigns only integral packets. We start by assuming that algorithm $\hat{A}$ has queues of size $B+1$ (algorithm A maintains queues of size B); we shall get rid of this assumption later on. Intuitively, $\hat{A}$ emulates the scheduling of A and accepts only integral packets. We start with a definition and an observation before we define algorithm $\hat{A}$ formally.

Definition 5.3 *We denote algorithms that accept integral packets if there is sufficient space, as* discrete greedy admission control *algorithms.*

Observation 5.3 *Every greedy admission control algorithm assigns fractions on a queue only when the load on that queue is strictly above $B-1$.*

We now define the transformation of a given greedy admission control algorithm A with queues of size B into algorithm $\hat{A}$ with queues of size $B+1$ which only accepts integral packets.

Algorithm $\hat{A}$:

- Maintain a running simulation of A. Let k_i^t be the fraction size which was transmitted by algorithm A at time t from queue i.
- **Admission control:** Perform discrete greedy admission control.
- **Scheduling:** For each queue i transmit a fraction of size k_i^t.

Let l_i^t and $\hat{l}_i^t$ be the loads of queue i in a given time unit t in A and $\hat{A}$, respectively. For the algorithm to be well defined, i.e. $\hat{l}_i^t \geq k_i^t$ after the arrival phase, we must prove the next lemma.

Lemma 5.8 *For each queue i and time unit t, after the arrival phase $\hat{l}_i^t \geq l_i^t$.*

Proof: The proof is by induction on time unit t for each queue i. For $t=0$ the claim is trivial. Suppose the claim is true for t, and consider $t+1$. We consider the two phases of t. At the arrival phase of t no transmission occurs and packets may arrive. Therefore, we consider any integral packet or fractional packet p arriving in time t at some queue i. If p is an integral packet which A accepts, if $\hat{A}$ does not accept p then $\hat{l}_i^t > B \geq l_i^t$ (since $\hat{A}$ is a discrete greedy admission control algorithm). If p is a fractional packet which A accepts, if $\hat{A}$ does not accept the integral packet which the fraction belongs to, we obtain that $\hat{l}_i^t > B = l_i^t$ (where the equality derives from Observation 5.3). If both algorithms accept p (if p is a fractional packet $\hat{A}$ may accept the integral packet which the fraction belongs to) then $\hat{l}_i^t \geq l_i^t$ by the induction hypothesis. Since in any case, after the arrival phase $\hat{l}_i^t \geq l_i^t$ both algorithms transmit k_i^t, and the inequality holds. ■

Corollary 5.9 *Algorithm $\hat{A}$ can always transmit k_i^t as defined.*

Theorem 5.10 *For a given algorithm A with queues of size B, algorithm $\hat{A}$ with queues of size $B+1$ has the same throughput given the same sequence σ.*

Proof: From Corollary 5.9, at each time unit t, $\hat{A}$ transmits the same total size as algorithm A. ■

Recall that EP is not a work-conserving algorithm. Therefore $\hat{EP}$ which emulates the scheduling of EP is also not a work-conserving algorithm. Since we want to use some of the results from the cost problem in Subsection 5.1, we first need to transform algorithm $\hat{EP}$ into a work-conserving algorithm $\hat{EP}'$. We use the transformation presented by Lemma 5.1 in Subsection 5.1.

Lemma 5.11 *Every non work-conserving algorithm A can be transformed into a work-conserving algorithm A' which accepts only packets which are accepted by A, while not worsening the performance of any sequence (in particular, $c_{A'} \leq c_A$, where $c_{A'}$, c_A are the competitive ratios of A' and A, respectively).*

Proof: Clearly, Claim 5.2 which states that $l_i^A \geq l_i^{A'}$, can easily be modified for the maximizing switch throughput problem when A' accepts only packets which are accepted by A. Thus it is not possible for algorithm A' to reject some packet p which was accepted by A on some queue i. ■

Corollary 5.12 *Algorithm $\hat{EP}$ can be transformed into $\hat{EP}'$ which is a work-conserving algorithm, and maintain its competitiveness.*

Now, we consider Algorithm M which was presented in Section 5.1. We assume that M has queues of size $B + 1 + \lfloor H_m \rfloor$. Recall that the precondition of M was that the parameter algorithm is work-conserving. Hence, we apply algorithm M on algorithm $\hat{EP}'$ and denote the resulting algorithm as $M^{\hat{EP}'}$. The sequence of packets which is given to $M^{\hat{EP}'}$ will consist **solely** of the packets which were accepted by $\hat{EP}'$. Note that this is a sequence which consists exclusively of integral packets. We emphasize that $M^{\hat{EP}'}$ is a *discrete* algorithm, since it only transmits integral packets. We now present the following lemma:

Lemma 5.13 *Algorithm $M^{\hat{EP}'}$ with queues of size $B + 1 + \lfloor H_m \rfloor$ has the same throughput as algorithm $\hat{EP}'$ with queues of size $B + 1$, given the same sequence σ.*

Proof: Consider the sequence of packets which was accepted by $\hat{EP}'$ as a sequence for the cost scheduling problem. In addition, recall that $\hat{EP}'$ is a work-conserving algorithm. Since the sequence of packets which is given to $M^{\hat{EP}'}$ consists **solely** of the packets which were accepted by $\hat{EP}'$, from Theorem 5.3, it follows that if the queues were of size $B + H_m + 1$ no packet loss may occur for $M^{\hat{EP}'}$. Since $M^{\hat{EP}'}$ transmits only integral packets and assigns only integral packets, queues of size $B + \lfloor H_m \rfloor + 1$ for $M^{\hat{EP}'}$ will suffice. ■

We now return to our original model where queues are of size B. By Lemma 5.13, if $M^{\hat{EP'}}$ had queues of size $B + \lfloor H_m \rfloor + 1$, then algorithms $M^{\hat{EP'}}$ and algorithm $\hat{EP'}$ would have had equal throughput. Unfortunately, this is not the case, so we continue by emulating an algorithm with large queues with an algorithm of small queues. Specifically, assume we are given an online competitive discrete algorithm A with queues of size y and we want to produce a competitive algorithm E^A with queues $y' < y$. Intuitively, E^A tries to emulate the schedule of algorithm A and accept only packets which A accepts as long as the queue is not full. We emphasize that if algorithm A was a fractional algorithm this problem could easily be solved by scaling the accepted volume and the transmitted volume of A by y'/y. Thus we continue by considering only discrete algorithms.

Algorithm E^A:

- Maintain a running simulation of algorithm A.
- **Admission control**: accept a packet if A accepts it and the queue is not full.
- **Scheduling**: transmit packets as A if the queue is not empty.

We show that the competitiveness of E^A is the competitive ratio of A times the ratio $y/y' > 1$. We start with the following remarks, definitions and lemmas.

Remark 7 *Let OPT and OPT' be the offline optimal algorithm with large queues and small queues, respectively. Note that the competitiveness of algorithm A is against OPT with large queues. Clearly, for each input sequence σ, $OPT(\sigma) \geq OPT'(\sigma)$. Thus, E^A-competitiveness is even against OPT.*

Definition 5.4 *We define the term* non-empty period *for a queue i as the maximum number of consequent units of time for which queue i is not empty. If a non-empty period for a queue i consists of time units t up to t', then the non-empty period starts at the arrival phase of time unit t and ends after the transmission phase of unit unit t'.*

Remark 8 *For each queue i, time is partitioned into disjoint non-empty periods and periods where queue i is empty.*

Lemma 5.14 *Let algorithm A maintain queues of size y and let E^A maintain queues of size y' such that $y' < y$. Let X be the number of packets which are accepted by A during a non-empty period for queue i of algorithm E^A. For each non-empty period for queue i of algorithm E^A, E^A rejects at most* $\min\{(X - y')_+, y - y'\}$*, where $(f)_+$ is* $\max\{0, f\}$.

Proof: We consider some queue i. For every non-empty period of E^A where $X < y$ the claim is easy, and follows from the fact that at least y' packets can be accepted sequentially, since the load on queue i at the beginning of every non-empty period is equal to 0. Assume that $X > y$ and assume by contradiction that E^A rejected more than $y - y'$ packets (this is the minimum). Consider the first packet p among those packets. When p is rejected by queue i, queue i holds exactly y' packets. Observe that algorithm E^A imitates the schedule of A, and that queue i is not empty during a non-empty period of queue i. Hence, during this non-empty period E^A transmits the same number of packets as algorithm A does. Therefore, from volume preservation, when p arrives the load of queue i in the scenario of algorithm A is at least $y' + y - y' = y$. Thus, A cannot accept packet p, which contradicts our assumption. ■

Theorem 5.15 *Let A maintain queues of size y and let E^A maintain queues of size y' such that $y' < y$. Then $A(\sigma) \leq \frac{y}{y'} E^A(\sigma)$.*

Proof: The proof follows immediately from Lemma 5.14, by summing up the number of packets accepted in all of the non-empty periods for each queue i of E^A (all accepted packets will eventually be transmitted). We emphasize that when we are not in a non-empty period no packets arrive. ■

We prove the main result of this paper with the next theorem.

Theorem 5.16 *Algorithm $E^{M^{\hat{EP'}}}$ for the maximizing switch throughput problem is $\frac{e}{e-1}(1 + \frac{\lfloor H_m \rfloor + 1}{B})$-competitive.*

Proof: Let A_X denote an online algorithm A which works on queues of size X, where $X \geq 0$. Then:

$$\begin{aligned} OPT_B(\sigma) &\leq \frac{e}{e-1} EP_B(\sigma) \\ &\leq \frac{e}{e-1} \hat{EP}_{B+1}(\sigma) \\ &\leq \frac{e}{e-1} \hat{EP'}_{B+1}(\sigma) \\ &\leq \frac{e}{e-1} M^{\hat{EP'}}_{B+1+\lfloor H_m \rfloor}(\sigma) \\ &\leq \frac{e}{e-1}(1 + \frac{\lfloor H_m \rfloor + 1}{B}) E_B^{M^{\hat{EP'}}} \end{aligned}$$

where the first inequality is obtained from Theorem 4.5, the second from Theorem 5.10, the third from Lemma 5.11, the fourth from Lemma 5.13, and the last inequality from Theorem 5.15. Note that $E_B^{M^{\hat{EP'}}}$ is a discrete algorithm. ■

Corollary 5.17 *For $B \gg H_m$ the competitive ratio of $E^{M^{\hat{EP'}}}$ approaches $\frac{e}{e-1} \approx 1.58$.*

6 Discussion and open problems

- Albers and Schmidt [3] show a lower bound of $\frac{e}{e-1}$, for $m \gg B$. They also show an algorithm with competitiveness $\frac{17}{9} \approx 1.89$ for every $B > 1$. Our main result in this paper shows that for the case were $B > \log m$ our algorithm can do much better than 1.89. Our algorithm approaches a competitive ratio of $\frac{e}{e-1}$, which interestingly is the lower bound for $m \gg B$. Azar and Richter [5] show a lower bound of $1.366 - \Theta(\frac{1}{m})$ for any B and m, which is smaller than the lower bound of $\frac{e}{e-1}$. Therefore, there is still work to be done to determine the optimal competitive ratio for arbitrary B and m.
- It is an interesting question whether greedy algorithms such as LQF (transmit from the longest queue first) has a competitive ratio smaller than 2, for $B \gg m$. For $m \gg B$ this is not true as is seen in [3].
- For $B \gg \log m$ our general technique shows that the competitive ratio of the discrete version algorithm approaches the performance of a fractional algorithm. Hence, it is interesting to determine the best fractional algorithm for large sized queues.

References

[1] M. Adler, P. Berenbrink, T. Friedetzky, L. Goldberg, and M. Paterson. A proportionate fair scheduling rule with good worst-case performance. In *Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 101–108, 2003.

[2] W. A. Aiello, Y. Mansour, S. Rajagopolan, and A. Rosen. Competitive queue policies for differentiated services. In *Proceedings of the IEEE INFOCOM '2000*, pages 431–440, 2000.

[3] S. Albers and M. Schmidt. On the performance of greedy algorithms in packet buffering. In *Proc. 36th ACM Symp. on Theory of Computing*, 2004.

[4] N. Andelman, Y. Mansour, and A. Zhu. Competitive queueing policies for QoS switches. In *Proc. 13rd ACM-SIAM Symp. on Discrete Algorithms*, pages 761–770, 2003.

[5] Y. Azar and Y. Richter. Management of multi-queue switches in QoS networks. In *Proc. 35th ACM Symp. on Theory of Computing*, pages 82–89, 2003.

[6] Y. Azar and Y. Richter. The zero-one principle for switching networks. In *Proc. 36th ACM Symp. on Theory of Computing*, 2004.

[7] A. Bar-Noy, A. Freund, S. Landa, and J. Naor. Competitive on-line switching policies. In *Proc. 13rd ACM-SIAM Symp. on Discrete Algorithms*, pages 525–534, 2002.

[8] A. Birman, H. R. Gail, S. L. Hantler, Z. Rosberg, and M. Sidi. An optimal service policy for buffer systems. *Journal of the Association Computing Machinery (JACM)*, 42(3):641–657, 1995.

[9] M. Chrobak, J. Csirik, C. Imreh, J. Noga, J. Sgall, and G. J. Woeginger. The buffer minimization problem for multiprocessor scheduling with conflicts. In *Proc. 28th International Colloquium on Automata, Languages, and Programming*, pages 862–874, 2001.

[10] B. Kalyanasundaram and K. R. Pruhs. An optimal deterministic algorithm for online b-matching. *Theoretical Computer Science*, 233:319–325, 2000.

[11] R. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of 22nd Annual ACM Symposium on Theory of Computing*, pages 352–358, may 1990.

[12] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 520–529, 2001.

[13] A. Kesselman and Y. Mansour. Loss-bounded analysis for differentiated services. In *Proc. 12nd ACM-SIAM Symp. on Discrete Algorithms*, pages 591–600, 2001.

[14] H. Koga. Balanced scheduling toward loss-free packet queuing and delay fairness. In *Proc. 12th Annual International Symposium on Algorithms and Computation*, pages 61–73, 2001.

[15] Z. Lotker and B. Patt-Shamir. Nearly optimal fifo buffer management for diffserv. In *Proc. 21st ACM Symp. on Principles of Distrib. Computing*, pages 134–143, 2002.

[16] M. May, J. C. Bolot, A. Jean-Marie, and C. Diot. Simple performance models of differentiated services for the internet. In *Proceedings of the IEEE INFOCOM '1999*, pages 1385–1394, 1999.

A Appendix A

Lower bound for the fractional version of the online unweighted matching. In this appendix we show that the matching algorithm WL, which was presented in Section 3, has an optimal competitive ratio of up to an additive smaller than 2 for the size of the matching. We show this by obtaining that the size of the match constructed by any fractional algorithm on a specific sequence σ is $< m(1-\frac{1}{e})+2$. Note that it can be easily shown that any algorithm A can be transformed into a work-conserving algorithm A', i.e. a fractional algorithm which matches as much as possible, while not worsening the performance of any sequence (in particular, $c_{A'} \leq c_A$, where $c_{A'}$, c_A are the competitive ratios of A' and A, respectively). Thus, we focus only on work-conserving fractional algorithms.

Theorem A.1 *The competitive ratio of every online deterministic fractional algorithm for the fractional unweighted matching problem in a bipartite graph is at least $\frac{e}{e-1} - \Theta(\frac{1}{m})$, where m is the number of servers in the bipartite graph.*

Remark 9 *Note that this result slightly improves the lower bound of $\frac{e}{e-1} - o(1)$ of Karp* et al.*[11] for randomized discrete algorithms, since a fractional algorithm can match fractions equal to the probabilities of the any randomized discrete algorithm and obtain its expectancy deterministically. We also note that contrary to the proof of Karp* et al. *in [11], our proof is not asymptotic.*

Proof: We prove the theorem by showing that for every fractional algorithm there exists an input sequence σ with optimal offline match of size m and online match of size at most $\lfloor m(1-\frac{1}{e})\rfloor + 2 - \frac{1}{e}$. Consider a bipartite graph with m servers and fix any online algorithm A. The construction of the sequence by the adversary is as follows:
Let I_j be the set of $m-j$ online servers with maximum load (breaking ties arbitrarily) before the arrival of request r_j. Then for $j = 0, 1, 2 \ldots m-1$ generate request r_j such that it is adjacent only to servers $s \in I_j$.

Note that in each step j, OPT will match the entire request on the server which will be excluded from I_j in the next iteration and hence $OPT(\sigma) = m$. We start the analysis of algorithm A on σ by defining the term $L_j(X)$ for a given set of servers X. Let $L_j(X)$ be the average online load of the servers $s \in X$ before the arrival of request r_j. Now, we prove some claims and lemmas.

Claim A.2 *For all $0 \leq j \leq m-1$ as long as algorithm A matches the entire request we have $L_j(I_j) \geq H_m - H_{m-j}$.*

Proof: The proof is by induction on j. For $j = 0$, the claim is clearly true. Suppose the claim is true for j, i.e. $L_j(I_j) \geq H_m - H_{m-j}$, and consider $j+1$. When request r_j arrives it can be matched solely on servers $s \in I_j$; thus we obtain that

$$L_{j+1}(I_j) \geq H_m - H_{m-j} + \frac{1}{m-j} = H_m - H_{m-(j+1)}.$$

Since there must be a server in I_j with load of at most $H_m - H_{m-(j+1)}$, then

$$L_{j+1}(I_{j+1}) \geq L_{j+1}(I_j) = H_m - H_{m-(j+1)}.$$

■

Observation A.1 *Algorithm A can match a request partially only once (assuming is a work-conserving algorithm).*

Lemma A.3 *The size of the fractional matching constructed by A is at most $\lfloor m(1-\frac{1}{e})\rfloor + 2 - \frac{1}{e}$.*

Proof: Algorithm A cannot match a request r_j on I_j if $L_j(I_j)$ is equal to 1 (since in that case all servers in I_j have a load equal to 1). By Claim A.2, $L_j(I_j) \geq H_m - H_{m-j}$. Hence, the number of requests which A can match, prior to the last matched request r_j, is obtained by solving the following inequality:

$$\sum_{i=m-j+1}^{m} \frac{1}{i} = H_m - H_{m-j} \leq 1.$$

Since

$$\sum_{i=m-j+1}^{m} \frac{1}{i} \geq \int_{m-j+1}^{m+1} \frac{1}{i} = \ln(m+1) - \ln(m-j+1),$$

it suffices to find the maximum j:

$$\ln(m+1) - \ln(m-j+1) \leq 1.$$

Hence,

$$j \leq \left\lfloor (m+1)(1-\frac{1}{e}) \right\rfloor.$$

Consequently, when request j arrives, A can match a total size smaller than 1. Therefore, by Observation A.1, the online fractional matching size is at most $\lfloor m(1-\frac{1}{e})\rfloor + 2 - \frac{1}{e}$. ■

Corollary A.4 *Note that our proof yields that the size of the matching constructed by any randomized discrete algorithm is at most $m(1-\frac{1}{e}) + 2 - \frac{1}{e}$. This slightly improves the current result which was presented by Karp* et al. *[11].*

Remark 10 *A tighter analysis shows that the size of the fractional matching constructed by A is at most $m(1-\frac{1}{e}) + 1 - \frac{1}{e}$.*

Theorem A.1 follows immediately from Lemma A.3. ■

Corollary A.5 *Algorithm WL is optimally competitive, up to an additive smaller than 2 for the size of the matching.*

B Appendix B

The online b-matching problem. We show a simplified proof for the online b-matching problem that was studied in [10]. We start by defining an online version to the unweighted b-matching problem. In the online b-matching problem we have a bipartite graph $G = (R, S, E)$, where S and R are the disjoint vertex sets and E is the edge set. We refer to set R as the requests and to set S as the servers. The objective is to match a request to a server. At step i vertex $r_i \in R$, together with all of its incident edges, arrives online. The response of algorithm A can be either to reject request r_i, or to irreversibly match the request to **one** adjacent vertex $s_j \in S$ such that the number of the requests that were matched to s_j is at most b. Thus, every $s_j \in S$ can be used **up to** b times by OPT and by A. The b-matching problem can be viewed as a restriction of the fractional model presented in Section 3 in which:

1. All the request sizes x_i are equal to $1/b$.
2. A request r_i is assigned integrally on an adjacent server s_j, i.e. $\forall_{i,j} k_j^i = 1/b$.

Under these constraints algorithm WL which was presented in Section 3 is in fact algorithm *Balance* that was studied in [10]. In [10] it was proved that algorithm *Balance* has a competitive ratio of $\frac{(1+\frac{1}{b})^b}{(1+\frac{1}{b})^b-1}$. We now show a more compact proof for the competitive ratio of *Balance* under constraints 1-2.

Algorithm *Balance*: Match each request r_i to the least loaded adjacent server s_j, unless each adjacent server was matched b times.

Theorem B.1 *Algorithm Balance is* $\frac{(1+\frac{1}{b})^b}{(1+\frac{1}{b})^b-1}$*-competitive in the restricted model.*

Proof: We use the same terms that were suggested in Theorem 3.1. Consider the total load assigned to the servers by *Balance* above some height h, where $h = i/b$ such that $i \in \mathbb{N}$ and $0 \leq i \leq b$. Let k be the total size of fraction in $Y_h \cup X_h$, i.e. the total size of fraction that were assigned by OPT and assigned by WL above height h or rejected. Clearly, $k \geq L_{OPT} - v(h)$. From volume preservation this load was assigned to at least $\lceil k \rceil$ different servers by OPT. Hence, the amount of load assigned by *Balance* to those $\lceil k \rceil$ servers is at least h. Since OPT used those $\lceil k \rceil$ servers WL could have used them as well. Thus some of the load assigned above height h could have been assigned below height h. Therefore for height $h = i/b$ we obtain:

$$v(h) - v(h - 1/b) \geq \frac{L_{OPT} - v(h)}{b}$$

which is the same as

$$v(h) \geq \frac{L_{OPT}}{b+1} + \frac{b}{b+1} v(h - \frac{1}{b}).$$

Thus by developing the right side of the equation recursively,

$$\begin{aligned} v(i/b) &\geq \frac{L_{OPT}}{b+1}(1 + \frac{b}{b+1} + \cdots + (\frac{b}{b+1})^{i-1}) \\ &= \frac{L_{OPT}}{b+1} \frac{(\frac{b}{b+1})^i - 1}{\frac{b}{b+1} - 1} \\ &= L_{OPT}(1 - (\frac{b}{b+1})^i). \end{aligned}$$

Hence,

$$L_{OPT} \leq v(i/b) \frac{(1 + \frac{1}{b})^i}{(1 + \frac{1}{b})^i - 1}. \quad (2)$$

Since $L_{OPT} = OPT(\sigma)$ and $v(b) = Balance(\sigma)$, algorithm *Balance* is $\frac{(1+\frac{1}{b})^b}{(1+\frac{1}{b})^b-1}$-competitive. ■

Theorem B.2 *Algorithm Balance is* $\frac{(1+\frac{1}{b})^\alpha}{(1+\frac{1}{b})^\alpha-1}$*-competitive in the restricted model with resource augmentation, where the online algorithm can match* α *requests on a single server and* OPT *can match* b *requests on a single server (all requests are of size* $\frac{1}{b}$*).*

Proof: The proof follows immediately from the fact that Equation (2) holds in the restricted model with resource augmentation. ■