Algorithmica (2013) 67:498–515
DOI 10.1007/s00453-012-9700-0

A Universal Randomized Packet Scheduling Algorithm

Łukasz Jeż

Received: 31 January 2012 / Accepted: 15 October 2012 / Published online: 24 October 2012

Abstract We give a memoryless scale-invariant randomized algorithm REMIX for *Packet Scheduling* that is $e/(e-1)$-competitive against an adaptive adversary. REMIX unifies most of previously known randomized algorithms, and its general analysis yields improved performance guarantees for several restricted variants, including the s-bounded instances. In particular, REMIX attains the optimum competitive ratio of $4/3$ on 2-bounded instances.

Our results are applicable to a more general problem, called *Item Collection*, in which only the relative order between packets' deadlines is known. REMIX is the optimal *memoryless* randomized algorithm against adaptive adversary for that problem.

Keywords Online algorithms · Competitive analysis · Adaptive adversary · Packet scheduling · Buffer management with bounded delay

1 Introduction

In this paper, we consider the problem of *Packet Scheduling*, introduced by Kesselman et al. [14] (under the name *Buffer Management with Bounded Delay*). This problem models the behavior of a single network switch responsible for scheduling packet transmissions along an outgoing link as follows. We assume that time is divided into

A preliminary version of this article appeared in Proceedings of the 19th European Symposium on Algorithms (ESA) under different title; the algorithm's name was also slightly different.

Ł. Jeż (✉)
Institute of Computer Science, University of Wrocław, ul. Joliot-Curie 15, 50-383 Wrocław, Poland
e-mail: lje@cs.uni.wroc.pl

Ł. Jeż
Institute of Mathematics, Academy of Sciences of the Czech Republic, Žitná 25, 115 67 Praha 1, Czech Republic

unit-length steps. At the beginning of a time step, any number of packets may arrive at a switch and be stored in its *buffer*. Each packet has a positive weight, corresponding to the packet's priority, and a deadline, which specifies the latest time when the packet can be transmitted. Only one packet from the buffer can be transmitted in a single step. A packet is removed from the buffer upon transmission or expiration, i.e., reaching its deadline. The goal is to maximize the *gain*, defined as the total weight of the packets transmitted.

As many others, we prefer to call the problem *Packet Scheduling* since it is equivalent to a single machine scheduling of unit-length jobs, with given weights, release times and deadlines; the last two restricted to integer values. In this setting, the goal is to maximize the total weight of jobs completed before their respective deadlines.

As the process of managing a packet queue is inherently a real-time task, we model it as an *online problem*. This means that the algorithm, when deciding which packets to transmit, has to base its decision solely on the packets which have already arrived at a switch, without the knowledge of the future.

1.1 Competitive Analysis

To measure the performance of an online algorithm, we use the standard notion of *competitive analysis* [5], which, roughly speaking, compares the gain of the algorithm to the gain of the *optimal solution* on the same instance. For any algorithm ALG, we denote its gain on instance I by $\mathcal{G}_{\mathrm{ALG}}(I)$. The optimal offline algorithm is denoted by OPT. We say that a deterministic algorithm ALG is $\mathcal{R}$-competitive if on any instance I it holds that $\mathcal{G}_{\mathrm{ALG}}(I) \geq \frac{1}{\mathcal{R}} \cdot \mathcal{G}_{\mathrm{OPT}}(I)$.

When analyzing the performance of an online algorithm ALG, we view the process as a game between ALG and an *adversary*. The adversary creates the instance on the one hand, i.e., controls what packets are injected into the buffer, and solves the instance optimally on the other, i.e., chooses its packets for transmission. The goal is then to show that the adversary's gain is at most $\mathcal{R}$ times ALG's gain.

If the algorithm is randomized, we consider its expected gain, $\mathbb{E}[\mathcal{G}_{\mathrm{ALG}}(I)]$, where the expectation is taken over all possible random choices made by ALG. However, in the randomized case, the power of the adversary has to be further specified. Following Ben-David et al. [2], we distinguish between an *oblivious* and an *adaptive-online* adversary, called *adaptive* for shortness from now on. An oblivious adversary has to construct the whole instance in advance. This instance may depend on ALG but not on the random bits used by ALG during the computation. The expected gain of ALG is compared to the gain of the optimal offline solution on I. In contrast, in case of an adaptive adversary, the choice of packets to be injected into the buffer may depend on the algorithm's behavior up to the given time step. This adversary must also provide an answering entity ADV, which creates a solution in parallel to ALG. This solution may not be changed afterwards. We say that ALG is $\mathcal{R}$-competitive against an adaptive adversary if for any adaptively created instance I and any answering algorithm ADV, it holds that $\mathbb{E}[\mathcal{G}_{\mathrm{ALG}}(I)] \geq \frac{1}{\mathcal{R}} \cdot \mathbb{E}[\mathcal{G}_{\mathrm{ADV}}(I)]$. We note that ADV is (wlog) deterministic, but as ALG is randomized, so is the instance I.

In the literature on online algorithms [5], the definition of the competitive ratio sometimes allows an additive constant, i.e., a deterministic algorithm is then called

$\mathcal{R}$-competitive if there exists a constant $\alpha \geq 0$ such that for any instance I it holds that $\mathcal{G}_{\mathrm{ALG}}(I) \geq \frac{1}{\mathcal{R}} \cdot \mathcal{G}_{\mathrm{OPT}}(I) - \alpha$. An analogous definition applies to the randomized case. For our algorithm REMIX the bound holds for $\alpha = 0$, which is the best possible.

1.2 Basic Definitions

For a packet j, we denote its: *weight* by w_j, *relative deadline* by d_j, *absolute deadline* by D_j, and *release time* by r_j. The absolute deadline is the exact point in time at which the packet expires, whereas the relative deadline is, *at any time*, the number of further steps after which the packet expires. To avoid nested subscripts we will use slightly different notation for sequences of packets: when considering a sequence of packets $j_1, j_2, \ldots, j_n$, instead of x_{j_i} (where $x \in \{d, D, r, w\}$), we simply write x_i. We also denote a packet of weight w and relative deadline d by (w, d). A packet that is in the buffer, i.e., has already been released and has neither expired nor been transmitted by an algorithm, is called *pending* for the algorithm. The *lifespan* of a packet is its relative deadline value upon injection, or in other words the difference between its absolute deadline and release time.

The goal is to maximize the weighted throughput, i.e., the total weight of transmitted packets. We assume that time is slotted in the following way. We distinguish between points in time and time intervals, called *steps*. In step t, corresponding to the interval $(t, t+1)$, ADV and the algorithm choose, independently, a packet from their buffers and transmit it. The packet transmitted by the algorithm (ADV) is immediately removed from its buffer and no longer pending. Afterwards, at time $t+1$, the relative deadlines of all remaining packets are decremented by 1, and the packets whose relative deadlines reach 0 expire and are removed from both ADV's and the algorithm's buffers. Next, the adversary injects any set of packets. At this point, we proceed to step $t+1$.

All known algorithms are *scale-invariant*, which means that they make the same decisions if all the weights of packets in an instance are scaled by a positive constant. A class of further restricted algorithms is of special interest for their simplicity: an algorithm is *memoryless* if in every step its decision depends only on the set of packets pending at that step.

1.3 Previous and Related Work, Restricted Variants

The currently best, 1.828-competitive, deterministic algorithm for general instances was given by Englert and Westermann [9]. Their algorithm is scale-invariant, but it is *not* memoryless. However, the same article provides a 1.893-competitive variant that is memoryless scale-invariant. The best known randomized algorithm is the 1.582-competitive memoryless scale-invariant RMIX, proposed by Chin et al. [6]. For reasons explained in Sect. 2.1 the original analysis by Chin et al. is only applicable to the oblivious adversary model. However, a refined analysis shows that RMIX remains 1.582-competitive in the adaptive adversary model [4].

The greedy algorithm (which always transmits the heaviest pending packet) has competitive ratio *exactly* 2 [12, 14], i.e., this memoryless scale-invariant algorithm is 2-competitive and no better than that. For a few years, no better deterministic algorithm for the general case was known, which led to a study of many restricted

Table 1 Comparison of known and new results. All the upper bounds in *the middle column* (randomized algorithms for adaptive adversary model) follow from this article, as does (cf. Sect. 3) the one for similarly ordered instances in oblivious adversary model; a reference next to one of these entries means that this particular bound was already known. The results without citations are implied by other entries of the table. *An asterisk* denotes that the algorithm attaining the bound is memoryless scale-invariant

		deterministic	(rand.) adaptive	(rand.) oblivious
general	upper	1.828 [9], 1.893* [9]	1.582* [4]	1.582* [6]
	lower	1.618	1.333	1.25
similarly ordered	upper	1.618* [16]	1.582*	1.333* [13]
	lower	1.618	1.333	1.25
s-bounded	upper	$2-\frac{2}{s}+o(\frac{1}{s})$* [6]	$1/(1-(1-\frac{1}{s})^s)$*	$1/(1-(1-\frac{1}{s})^s)$*
	lower	1.618	1.333	1.25
2-bounded	upper	1.618* [14]	1.333* [4]	1.25* [6]
	lower	1.618 [7, 12, 18]	1.333 [4]	1.25 [7]

variants. Below we present some of them, together with known results. The most relevant bounds known are summarized in Table 1. Note that the majority of algorithms are memoryless scale-invariant.

For a general overview of techniques and results on buffer management, see the surveys by Azar [1], Epstein and Van Stee [10] and Goldwasser [11].

Uniform Sequences An instance is *s-uniform* if the lifespan of each packet is exactly s. Such instances have been considered for two reasons. Firstly, there is a connection between them and the *FIFO model* of buffer management, also considered by Kesselmann et al. [14] (the connection itself stated in Corollary 5.5 therein). Secondly, the 2-uniform instances are among the most elementary restrictions that do not render the problem trivial. However, analyzing these sequences is not easy: while a simple deterministic $\sqrt{2}$-competitive algorithm for 2-uniform instances [18] is optimal among *memoryless* scale-invariant algorithms [6], for unrestricted algorithms, a sophisticated analysis shows the optimum competitive ratio is 1.377 [8].

Bounded Sequences An instance is *s-bounded* if the lifespan of each packet is at most s; therefore every s-uniform instances is also s-bounded. The 2-bounded instances give rise to the strongest currently known lower bounds. These are $\phi \approx 1.618$ for deterministic algorithms [7, 12, 18], 1.25 for randomized algorithms in the oblivious adversary model [7], and 4/3 in the adaptive adversary model [4]. For 2-bounded instances, algorithms matching these bounds are known [4, 6, 14]. A family of deterministic algorithms EDF_β is $2-\frac{2}{s}+o(\frac{1}{s})$-competitive for s-bounded instances [6]; β is a parameter suitably chosen as a function of s. Detailed analysis reveals that the ϕ-competitive deterministic algorithm for 2-bounded instances, itself a member of the EDF_β family, remains 3-competitive also on 3-bounded instances [6]. However, these algorithms' competitive ratio guarantees present an improvement over Englert and Westermann's algorithm only for small values of s.

Similarly Ordered Sequences An instance is *similarly ordered* or has *agreeable deadlines* if for every two packets i and j their spanning intervals are not properly contained in one another, i.e., if $r_i < r_j$ implies $D_i \leq D_j$. Note that every 2-bounded instance is similarly ordered, as is every s-uniform instance, for any s. An optimal deterministic ϕ-competitive algorithm [16] and a randomized 4/3-competitive algorithm for the oblivious adversary model (as well as an alternative optimal deterministic algorithm) [13] are known for similarly ordered instances.

Other Restrictions There are many other meaningful restricted variants of the problem, for which better algorithms are known. Interestingly, one of the optimal deterministic algorithms for similarly ordered instances [16] performs well in several restricted instance classes [15] (the performance of the other algorithm [13] was not studied). Let us mention one class of instances, studied by Kesselmann et al. in their seminal paper [14], for which our algorithm provides additional bounds: as various transmission protocols usually specify only several priorities for packets, it is sensible to bound the number of different packet weights.

Generalization: Collecting Weighted Items from a Dynamic Queue The following generalization of *Packet Scheduling* has been studied [3], and called *Item Collection* by its authors. In *Item Collection* the algorithm is to collect weighted items, one per step, from a dynamic queue S, which is in fact an ordered list. The content of S varies over time, as it is updated between any two consecutive time steps. Such an update is performed by an external agent (the adversary), and consists of deleting any number of items at the front of S (a prefix of S) and inserting new items into arbitrary locations in S. Note that the collection is the algorithm's action, not a queue update. However, the algorithm is allowed to collect an item only once, and only while it is in the queue. The objective is to maximize the total weight of the collected items.

Item Collection is easily seen to generalize *Packet Scheduling*: let the items in the queue correspond to packets, their locations in S determined by non-decreasing order of their deadlines, with ties broken arbitrarily. These items have the same weights as their corresponding packets, are inserted upon their release, and deleted upon expiration—ordering by deadlines ensures that a prefix of S is deleted.

Note that in *Item Collection* the algorithm does not know how many steps will lapse between deletions of two successive items in its queue—this is why ties between packets' deadlines are broken arbitrarily in the reduction.

Some of the algorithms for *Packet Scheduling* extend to *Item Collection*, for example the greedy algorithm, which ignores the deadlines altogether. In general, any algorithm for *Packet Scheduling* that only compares packets' deadlines, without resorting to their exact values, and can thus be presented with a list of packets ordered by their deadlines, extends to *Item Collection*. Among such algorithms are the EDF_β algorithms, RMix, and our algorithm ReMix—reader is advised to keep this in mind, because we present it for the more popular setting of *Packet Scheduling*.

While the paper of Bienkowski's et al. [3] focuses on deterministic algorithms for *Item Collection*, it does provide a lower bound for memoryless randomized algorithms, matched by ReMix, cf. Sect. 5.3.

1.4 Our Contribution

We consider randomized algorithms against an adaptive adversary, motivated by the following observation. In reality, traffic through a switch is not at all independent of the packet scheduling algorithm. For example, lost packets are typically resent, and throughput through a node affects the choice of routes for data streams in a network. These phenomena can be captured by the adaptive adversary model but not by the oblivious one. The adaptive adversary model is also of its own theoretical interest and has been studied in other settings [5].

The main contribution of this paper is a simple memoryless scale-invariant algorithm REMIX, which unifies most of previously known randomized algorithms for *Packet Scheduling*. It also applies to the generalized *Item Collection* problem, for which it is optimal among *memoryless* randomized algorithms, cf. Sect. 5.3. As its name reflects, REMIX is very similar to RMIX, proposed by Chin et al. [6]. The only, yet crucial difference between these two algorithms is the transmission's probability distribution over pending packets. While RMIX is essentially the EDF_β algorithm with β chosen in each step according to a *fixed* (same in every step) probability distribution, it is no longer so in REMIX.

This subtle change allows a refined analysis of REMIX, which yields an upper bound on the algorithm's competitive ratio that depends on the maximum number of packets, over all steps, that have positive probability of transmission in the step. Specifically, if we denote that number by N, our upper bound on the competitive ratio of REMIX is $f(n) = 1/(1-(1-\frac{1}{N})^N)$. Note that $f(N)$ tends to $e/(e-1)$ from below. The number N can be bounded a priori in certain restricted variants of *Packet Scheduling*, thus giving better bounds for them. Specifically, it is easily observed that $N \leq s$ in instances with at most s different packet weights as well as in s-bounded instances. The particular upper bound of $4/3$ we obtain for 2-bounded instances (known before) is optimal in the adaptive adversary model [4]; the bounds for s-bounded instances and $s > 2$ (in both oblivious and adaptive adversary model) are new. Certain structural properties of optimum schedules for similarly ordered instances imply that $N \leq 2$ for that class of instances as well, at least in the oblivious adversary model, yielding an upper bound of $4/3$ (also known before [13]). While details are given in Sect. 3, most of the implications are summarized in Table 1.

2 General Upper Bound

2.1 Analysis Technique

In our analysis, we follow the paradigm of modifying ADV's buffer, introduced by Li et al. [16] (the development of this technique is covered in Sect. 4). Namely, we assume that in each step REMIX and ADV have precisely the same packets in their buffers. Once they both transmit a packet, we modify ADV's buffer judiciously to make it identical with that of REMIX. This leads to a streamlined and intuitive proof.

When modifying the buffer, we may have to let ADV transmit another packet, inject an extra packet to his buffer, or upgrade one of the packets in its buffer by

increasing its weight or deadline. We ensure that these changes are *advantageous to the adversary* in the following sense: for any adversary strategy ADV, starting with the current step and buffer content, there is an adversary strategy $\overline{\text{ADV}}$ that continues computation with the modified buffer, such that the total gain of $\overline{\text{ADV}}$ in this and the following steps, on any instance, is at least as large as that of ADV.

To prove R-competitiveness, we show that in each step the expected *amortized gain* of ADV is at most R times the expected gain of REMIX, where the former is the total weight of the packets that ADV eventually transmitted in this step. Both expected values are taken over random choices of REMIX.

We are going to assume that ADV never transmits a packet a if there is another pending packet b such that ADV is always better off by transmitting b. Formally, we introduce a dominance relation among the pending packets and assume that ADV never transmits a dominated packet.

We say that a packet $a = (w_a, d_a)$ is *dominated* by a packet $b = (w_b, d_b)$ at time t if at time t both a and b are pending, $w_a \leq w_b$ and $d_a \geq d_b$. If one of these inequalities is strict, we say that a is *strictly dominated* by b. We also say that packet a is (strictly) dominated at time t if at that time there is a pending packet b that (strictly) dominates it. When a packet a is not strictly dominated, we call it a *non-dominated* packet (observe that a dominates itself). In *Item Collection* setting, a is dominated by b if b precedes a in the queue and $w_b \geq w_a$; there is no distinction between dominance and strict dominance in *Item Collection*, since there are no ties between deadlines there.

Fact 1 *For any adversary strategy* ADV, *there is a strategy* $\overline{\text{ADV}}$ *such that*:

(i) *the gain of* $\overline{\text{ADV}}$ *on every sequence is at least the gain of* ADV,
(ii) $\overline{\text{ADV}}$ *transmits a non-dominated packet in every step*.

Proof ADV can be transformed into $\overline{\text{ADV}}$ by applying an exchange argument iteratively: take the minimum t_0 such that ADV first violates (ii) in step t_0, and transform ADV into an algorithm ADV$'$ with gain no smaller than that of ADV, which satisfies (ii) up to step t_0, possibly violating it in further steps.

Let t_0 be the first step in which (ii) is violated. Let $y = (w, d)$ be the packet transmitted by ADV and $x = (w', d')$ be a packet that strictly dominates y; then $w' \geq w$ and $d' \leq d$. Let ADV$'$ transmit the same packets as ADV up to step $t_0 - 1$, but in step t_0 let it transmit x, and in the remaining steps let it try to transmit the same packets as ADV. It is impossible in one case only: when ADV transmits x in some step t. But then $d \geq d' > t$, so let ADV$'$ transmit y, still pending at t. Clearly, (i) is preserved. □

The proof trivially extends to *Item Collection*.

We stress that Fact 1 holds for both oblivious and adaptive adversary model. Now we give an example of another simplifying assumption, common in the oblivious adversary model, which *seems* to break down in the adaptive adversary model.

In the oblivious adversary model the instance is fixed in advance by the adversary, so ADV may precompute the optimum schedule to the instance and follow it. Moreover, by standard exchange argument for the *fixed* set of packets to be transmitted, ADV may always send the packet with the smallest deadline from that set—this is usually called the *earliest deadline first* (EDF) property or order. This assumption not

only simplifies analyses of algorithms but is often crucial for them to yield desired bounds [6, 8, 13, 16].

In the adaptive adversary model, however, the following phenomenon occurs: as the instance I is randomized, ADV does not know for sure which packets it will transmit in the future. Consequently, deprived of that knowledge, it cannot ensure any specific order of packet transmissions. For example, it might be the case that in one step ADV decides to transmit a packet x, and, depending on the outcome of the algorithm's random choice, in the next step ADV might transmit either y or z, both pending for ADV already in the previous step, such that the deadlines of the three packets satisfy $D_y < D_x < D_z$. This example is elusive: while we are unable to preclude adversarial strategies of this kind, we are equally unable to come up with a specific instance where they would not be dominated by other adversarial strategies.

2.2 Algorithm and Its Analysis

Our algorithm roughly works as follows. In each step it finds a maximum chain of non-dominated pending packets $h_1, h_2, \ldots, h_m$, in decreasing order of weights, such that together they dominate all pending packets. By Fact 1, we may assume that ADV transmits one of those packets. REMIX chooses one of them at random, according to an appropriate probability distribution. The distribution is devised in such a way that the expected gain of REMIX in a step is on the one hand close to the maximum possible gain of ADV in that step, i.e., $w(h_1)$, but on the other hand it is spread over the whole chain to obfuscate the adversary. Details are presented in Algorithm 1.

We introduce the packet h_0 to shorten REMIX's pseudocode. The packet itself is chosen in such a way that $p_0 = 0$, to make it clear that it is not considered for transmission (unless $h_0 = h_1$). The while loop itself could be terminated as soon as $r = 0$, because afterwards REMIX does not assign positive probability to any packet. However, letting it construct the whole sequence $h_1, h_2, \ldots h_m$ such that $H_m = \emptyset$ streamlines our analysis. Before proceeding, we note that these easily follow from the algorithm.

Fact 2 *The sequence of packets $h_0, h_1, \ldots, h_m$ selected by* REMIX *satisfies*

$$w_0 = w_1 > w_2 > \cdots > w_m, \tag{1}$$

$$d_0 \geq d_1 > d_2 > \cdots > d_m. \tag{2}$$

Furthermore, every pending packet is dominated by one of $h_1, \ldots, h_m$.

Fact 3 *The numbers $p_1, p_2, \ldots, p_m$ form a probability distribution such that*

$$p_i \leq 1 - \frac{w_{i+1}}{w_i} \quad \text{for all } i < m. \tag{3}$$

Furthermore, the bound is tight for $i < n$, while $p_i = 0$ for $i > n$, i.e.,

$$p_i = \begin{cases} 1 - \frac{w_{i+1}}{w_i}, & \text{for } i < n \\ 0, & \text{for } i > n \end{cases} \tag{4}$$

Algorithm 1 REMIX (single step)

if there are no pending packets **then**
  do nothing and proceed to the next step
end if
$m \leftarrow 0$ ▷ counts non-dominated packets
$n \leftarrow 0$ ▷ counts packets with positive probability assigned
$r \leftarrow 1$ ▷ unassigned probability
$H_0 \leftarrow$ pending packets
$h_0 = (w_0, d_0) \leftarrow$ heaviest packet from H_0
while $H_m \neq \emptyset$ **do**
  $m \leftarrow m + 1$
  $h_m = (w_m, d_m) \leftarrow$ heaviest non-dominated packet from H_{m-1}
  $p_{m-1} \leftarrow \min\{1 - \frac{w_m}{w_{m-1}},\ r\}$
  $r \leftarrow r - p_{m-1}$
  if $r > 0$ **then**
    $n \leftarrow n + 1$
  end if
  $H_m \leftarrow \{x \in H_{m-1} \mid x \text{ is not dominated by } h_m\}$
end while
$p_m \leftarrow r$
transmit h chosen from $h_1, \ldots, h_n$ with probability distribution $p_1, \ldots, p_n$
proceed to the next step

With these, we can prove our main result.

Theorem 1 REMIX *is* $1/(1-(1-\frac{1}{N})^N)$*-competitive against an adaptive adversary, where* N *is the maximum number of packets, over all steps, that are assigned positive probability in a step.*

Proof To prove the theorem, we will describe, for any given step, the changes to ADV's scheduling decisions and modifications to its buffer that make it the same as REMIX's buffer. These, as explained in Sect. 2.1, affect the adversary's amortized gain. Having specified those, we will show that the expectations of the adversary's amortized gain and the algorithm's gain satisfy

$$\mathbb{E}[\mathcal{G}_{\text{ADV}}] \leq w_1, \tag{5}$$

$$\mathbb{E}[\mathcal{G}_{\text{REMIX}}] \geq w_1\left(1 - \left(1 - \frac{1}{n}\right)^n\right), \tag{6}$$

where n is the number of packets assigned positive probability in the step. The theorem follows by summation over all steps.

We begin by describing modifications to ADV's buffer. To this end, recall that, by Fact 1, ADV (wlog) sends a packet that is non-dominated. By Fact 2, the packets $h_1, h_2, \ldots h_m$ dominate all pending packets, so the one sent by ADV, say p, is (wlog) one of $h_1, h_2, \ldots h_m$: if p is dominated by h_i, but not strictly dominated, then p has

the same weight and deadline as h_i. The modifications to ADV's buffer depend on the relation between the two non-dominated packets that ADV and REMIX transmit in the step. Suppose that ADV transmits $h_z = (w_z, d_z)$ whereas REMIX transmits (the randomly chosen) $h_f = (w_f, d_f)$, respectively. Then there are two cases.

Case 1: $d_f \leq d_z$. Note that in this case $w_f \leq w_z$, since h_z is non-dominated. After both ADV and REMIX transmit their packets, we replace h_f in the buffer of ADV by a copy of h_z. This way their buffers remain the same afterwards, and the change is advantageous to ADV: this is essentially an upgrade of the packet h_f in its buffer, as both $d_f \leq d_z$ and $w_f \leq w_z$ hold.

Case 2: $d_f > d_z$. After both ADV and REMIX transmit their packets, we let ADV additionally transmit h_f, and we inject a copy of h_z into its buffer, both of which are clearly advantageous to ADV. This makes the buffers of ADV and REMIX identical afterwards.

Now that we have specified the modifications, we can bound the adversary's expected amortized gain, i.e., prove (5), For convenience, denote the amortized gain of ADV when it transmits h_z by $\mathcal{G}^{(z)}_{\text{ADV}}$. Clearly, $\mathcal{G}^{(z)}_{\text{ADV}}$ is a random variable that equals $w_z + w_f$ if $d_z < d_f$ (equivalently, $z > f$), and w_z otherwise. Thus, when ADV transmits h_z, its expected amortized gain is

$$\mathbb{E}\big[\mathcal{G}^{(z)}_{\text{ADV}}\big] = w_z + \sum_{i<z} p_i w_i. \tag{7}$$

As the adversary's expected amortized gain satisfies

$$\mathbb{E}[\mathcal{G}_{\text{ADV}}] \leq \max_{1\leq j\leq m} \big\{\mathbb{E}\big[\mathcal{G}^{(j)}_{\text{ADV}}\big]\big\}, \tag{8}$$

to establish (5), we will prove that

$$\max_{1\leq j\leq m} \big\{\mathbb{E}\big[\mathcal{G}^{(j)}_{\text{ADV}}\big]\big\} \leq \mathcal{G}^{(1)}_{\text{ADV}} = w_1. \tag{9}$$

The equality in (9) follows trivially from (7). To see that the inequality in (9) holds as well, observe that, by (7), for all $j < m$,

$$\mathbb{E}\big[\mathcal{G}^{(j)}_{\text{ADV}}\big] - \mathbb{E}\big[\mathcal{G}^{(j+1)}_{\text{ADV}}\big] = w_j - w_{j+1} - p_j w_j \geq 0, \tag{10}$$

where the inequality follows from (3). This end the proof of (5).

Now we turn to (6), the bound on the expected gain of REMIX in a single step. Obviously,

$$\mathbb{E}[\mathcal{G}_{\text{REMIX}}] = \sum_{i=1}^{n} p_i w_i. \tag{11}$$

By (4), $p_i w_i = w_i - w_{i+1}$ for all $i < n$. Also, $p_n = 1 - \sum_{i<n} p_i$, by Fact 3. Making corresponding substitutions in (11) yields

$$\mathbb{E}[\mathcal{G}_{\text{REMIX}}] = \left(\sum_{i=1}^{n-1}(w_i - w_{i+1})\right) + \left(1 - \sum_{i=1}^{n-1} p_i\right) w_n$$

$$= w_1 - w_n \sum_{i=1}^{n-1} p_i. \tag{12}$$

As (4) implies $w_i = w_{i-1}(1 - p_{i-1})$ for all $i \leq n$, we can express w_n as

$$w_n = w_1 \prod_{i=1}^{n-1} (1 - p_i). \tag{13}$$

Substituting (13) for w_n in (12), we obtain

$$\mathbb{E}[\mathcal{G}_{\text{REMIX}}] = w_1 \left(1 - \prod_{i=1}^{n-1} (1 - p_i) \sum_{i=1}^{n-1} p_i \right). \tag{14}$$

Let $x_i = 1 - p_i$ for $1 \leq i < n$ and let $x_n = \sum_{i=1}^{n-1} p_i$; note that $\sum_{i=1}^{n} x_i = n - 1$. The inequality between arithmetic and geometric means for $x_1, \ldots, x_n$ yields

$$\prod_{i=1}^{n-1} (1 - p_i) \sum_{i=1}^{n-1} p_i = \prod_{i=1}^{n} x_i \leq \left(\frac{\sum_{i=1}^{n} x_i}{n} \right)^n = \left(1 - \frac{1}{n} \right)^n. \tag{15}$$

Plugging (15) into (14) yields

$$\mathbb{E}[\mathcal{G}_{\text{REMIX}}] \geq w_1 \left(1 - \left(1 - \frac{1}{n} \right)^n \right), \tag{16}$$

which proves (6), and together with (5), the theorem. □

3 Application to Similarly Ordered and Other Restricted Instances

We have already mentioned that for s-bounded instances or those with at most s different packet weights, $N \leq m \leq s$ in Theorem 1, which trivially follows from Fact 2. Thus for either kind of instances REMIX is $1/(1 - (1 - \frac{1}{s})^s)$-competitive. In particular, on 2-bounded instances REMIX coincides with the previously known optimal 4/3-competitive algorithm RAND [4] for the adaptive adversary model.

While in a similarly ordered instance the number of non-dominated pending packets in any step can be arbitrarily large, surprisingly enough, in every step an *oblivious* adversary (wlog) transmits a packet from a set $\mathcal{H}$ of at most two packets that can be identified by the algorithm, assuming (as we did throughout the analysis) that the same packets are pending for the algorithm and the adversary. Hence, on such instances REMIX is 4/3-competitive against oblivious adversary, provided that H_0 is initialized to $\mathcal{H}$ in every step.

This surprising fact follows from Lemma 2, which we state and prove at the end of this section. As that lemma describes a property of optimum (offline) schedules, it applies to an oblivious adversary's schedule only. The end result, i.e., a 4/3-competitive algorithm was known before [13], as was in fact Lemma 2 itself. The lemma's and the analysis technique's origins and development are surveyed in Sect. 4.

Before we proceed, we need to inspect some basic properties of schedules. Formally, a schedule is an injective function that assigns packets to time steps within their lifespans. It can thus be thought of as a matching in a bipartite graph whose partitions are the packets and the steps respectively, and edges connect packets to the steps in their lifespans. The edges incident to a packet are assigned the packet's weight, in a sense making the graph vertex- rather than edge-weighted. Naturally, an optimal schedule for a set of packets corresponds to a maximum weight matching in the corresponding graph; for a set of packets X we denote that graph by G_X. Structural properties of optimum schedules and algorithms to find them thus follow from those of and for matchings.

Firstly, we note a relation between optimum schedules for sets of packets Q and Q' such that $Q \subseteq Q'$. While the lemma seems obvious and its proof is a simple reasoning about matchings, we prove it for completeness.

Lemma 1 *Let Q and Q' be two sets of packets such that $Q \subseteq Q'$. Then for every optimum schedule S for Q there exists an optimum schedule S' for Q' such that the set inclusion $S' \cap Q \subseteq S$ holds*

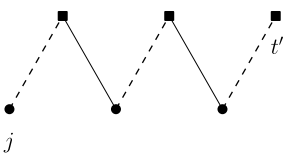
Proof Let S and S'' be any given optimum schedules for Q and Q' respectively. We demonstrate how to obtain an optimum schedule S' for Q' that satisfies $S' \cap Q \subseteq S$. To this end view both S and S'' as matchings in $G_{Q'}$ and consider their symmetric difference $S \oplus S''$, which is a disjoint sum of alternating paths and cycles. Unless there is some packet $j \in S'' \cap Q \setminus S$, we are done by letting $S' = S''$. Note that such $j \in S'' \cap Q \setminus S$ is an endpoint of some alternating path P in $S \oplus S''$. In the following we will consider augmenting either S or S'' along P—such operation results in a matching representing some schedule. Note that all packets in P come from Q, as they are scheduled in S.

First we prove that P has even length, i.e., it ends in a vertex corresponding to a packet from Q'. Assume for contradiction that P's length is odd, in which case P ends in a vertex corresponding to a time step $t' \in T$, i.e., no packet is assigned to t' in S. Then the matching $S \oplus P$ yields a schedule for Q that satisfies the set inclusion $S \cup \{j\} \subseteq S \oplus P$, contradicting optimality of S. See Fig. 1(a) for an illustration.

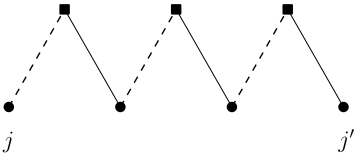
Hence P has even length and thus ends with a vertex corresponding to a packet $j' \in S \setminus S''$. Both S and S'' can be augmented using P, and in terms of packets scheduled this operation simply swaps j and j'. Thus $w_j = w_{j'}$ follows from optimality of both S and S''. Hence $S'' \oplus P$ is a matching in $G_{Q'}$ of the same weight as S'', i.e., it is also an optimum schedule for Q'. See Fig. 1(b) for an illustration.

Applying such changes iteratively transforms S'' to S' such that $S' \cap Q \subseteq S$. To observe that a finite number of iterations suffices, define $\Delta(X) := |X \cap Q \setminus S|$ for any schedule X. It follows that $\Delta(S'' \oplus P) = \Delta(S'') - 1$. Since Δ is non-negative and its value drops by one with each iteration, S' is obtained in a finite number of steps. □

In our case, Q will be the set of packets pending for ReMix (or Adv, since these sets are the same in our analysis) at a certain step, with the lifespans of the packets restricted to that and future steps. A schedule for Q is typically called a *provisional schedule*. The set Q', on the other hand, will be the union of Q with the set of all



(a) P cannot have odd length: in such case the assignment of packets on P in S could be changed to match the strictly better assignment of S''.



(b) P has even length: now the assignment of jobs on P in S'' can be changed to match the assignment of S so that the value of Δ drops.

Fig. 1 The alternating path P. Packets are represented by *discs*, time steps by *squares*. *Dashed lines* represent S'', *solid lines* represent S

packets to be released in the future steps of the instance. Then Lemma 1 implies that (wlog) ADV is going to transmit one of the packets from the optimum provisional schedule. Obviously, there can be arbitrarily many packets in that schedule but in similarly ordered instances the sets Q and Q' have additional properties, and Lemma 1 can be accordingly strengthened.

Lemma 2 *Let t be a step, and let Q and Q' be two sets of packets such that $r_j = t$ for all $j \in Q$, $Q \subseteq Q'$, $t < \min\{r_j \mid j \in Q' \setminus Q\}$, and $\max\{D_j \mid j \in Q\} \leq \min\{D_j \mid j \in Q' \setminus Q\}$. Also, let S be an EDF-ordered optimal schedule over Q. Then, if S is non-empty, there exists an optimal schedule S' over Q' such that in step t either e or h is transmitted in S', where e and h are any earliest-deadline and any maximum-weight packet from S respectively.*

Proof We assume (wlog) that S is EDF-ordered in such a way that $S(t) = e$. Let X be an optimal schedule over Q' that starts in time step t such that $X \cap Q \subseteq S$—its existence is guaranteed by Lemma 1. Moreover, we assume (wlog) that X is EDF-ordered in such a way that the packets from $S \cap X$ appear in X in exactly the same order as in S, and that they are all assigned before those from $Q' \cap X$. If $e \in X$, then we let $S' = X$, and we are done.

Otherwise, observe that since X is optimal, it assigns some packet $z \in Q$ to time step t. If $h \notin X$, then we let S' be X with h substituted for z—such S' is optimal, as $w_h \geq w_z$. Otherwise, we claim that S' can be obtained by reordering X so that h is assigned to time step t. To prove it, let us enumerate the packets from S in the order they are assigned: $j_1, j_2, \ldots, j_s$. Observe that $e = j_1$, $z = j_k$, and $h = j_l$ for some $1 < k < l \leq s$, by our assumptions ($k < l$, because z is assigned before h in both X and S). Let d_i denote the relative deadline of j_i at time t, $1 \leq i \leq s$. Then $d_i \geq i$ for $i = 1, \ldots, s$.

As $e = j_1 \notin X$ and $d_i \geq i$ for $i = 1, \ldots, s$, all the packets $x \in X \cap S$ have *slack* in X, i.e., they are not tight at the beginning of time steps they are assigned to, and thus could also be scheduled one step later. Hence we obtain S' by the following reordering of X. Firstly, $h = j_l$ is assigned to time step t in S'. Then, for every packet $x \in X \setminus \{h\}$ in the increasing order of $X^{-1}(x)$, the packet x is assigned to time step $X^{-1}(x)$ in S' if this step was yet unassigned and otherwise it is assigned to step $X^{-1}(x) + 1$. Note that since we freed the time step $X^{-1}(h)$ in S' by assigning h

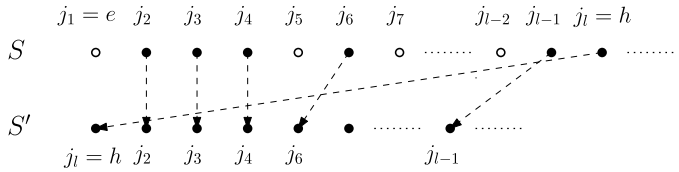


Fig. 2 Construction of the schedule S'. Packets from S are represented by *circles*: the ones included in S' are *filled*, the remaining ones are *hollow*. The packets from $S' \setminus Q$ are not illustrated, since their assignment does not change with respect to X

to time step t, for all $\tau > X^{-1}(h)$ it holds that $S'(\tau) = X(\tau)$. In particular, all the packets from X are assigned before their deadlines in S'. The reordering is illustrated in Fig. 2. □

4 Origins and Development of Analysis Technique

As mentioned before, our analysis technique can be traced back to the article that introduced the first optimal deterministic algorithm for similarly ordered instances [16]. The principle of modifying ADV 's buffer and Lemma 2 both appear there, though densely interwoven. Furthermore, the core analysis therein relied on obliviousness of the adversary (i.e., it following an optimal schedule) more than it was necessary: in what corresponds to Case 2 ($d_f > d_z$) in our Proof of Theorem 1, Li et al. [16] first noted that in ADV's (optimal) schedule f is transmitted later anyway, only then concluding that ADV may thus be allowed to transmit both z and f.

A later article about similarly ordered instances [13] separated the principle of the analysis and Lemma 2, though not completely. While the algorithm therein coincides with REMIX with H_0 initialized to $\mathcal{H} = \{e, h\}$ (as defined in Lemma 2), in the analysis the lemma was used only to handle some of the cases (specifically, that of the algorithm transmitting $h \neq e$ and ADV transmitting $j \neq e$), rather than to notice beforehand that (wlog) ADV transmits either e or h, just like the algorithm.

Subsequently, the technique was applied to analyze two algorithms for the adaptive adversary model: RAND, the optimum algorithm for 2-bounded instances, and RMIX [4]. Comparing the preliminary conference version of [4] to the final one is illustrative: the original analysis of RAND used a potential function argument, which resulted in a longer proof with larger number of cases; also, it did not give any analysis of RMIX, claiming that the original one [6] extends to the adaptive adversary model.

5 Probability Distribution and Tightness

In this section we prove tightness of our analysis, for all values of the parameter N. We begin with explaining how the probability distribution used by the algorithm was chosen; while it does not prove anything on its own, hopefully it is illuminating. Then we give *Packet Scheduling* instances that actually prove tightness, and finally

we note that the algorithm is optimal (in the class of memoryless algorithms) for the generalized problem of *Item Collection*. All these have some implications, as we explain in Sect. 6.

5.1 Probability Distribution Rationale

Recall that the upper bound on the competitive ratio of REMIX is

$$\frac{\max_{1\le z\le m}\{\mathbb{E}[\mathcal{G}^{(z)}_{\mathrm{ADV}}]\}}{\mathbb{E}[\mathcal{G}_{\mathrm{REMIX}}]}, \tag{17}$$

irrespective of the choice of $p_1, \ldots, p_m$.

The particular probability distribution used in REMIX is chosen to (heuristically) minimize the above ratio by maximizing $\mathbb{E}[\mathcal{G}_{\mathrm{REMIX}}]$, while keeping (9) satisfied, which, together with (8), implies $\mathbb{E}[\mathcal{G}_{\mathrm{ADV}}] \le \mathcal{G}^{(1)}_{\mathrm{ADV}} = w_1$.

The first goal can be trivially achieved by setting $p_1 \leftarrow 1$ but that would make $\mathbb{E}[\mathcal{G}^{(z)}_{\mathrm{ADV}}] > w_1$ for all $z > 1$. Therefore, some of the probability mass is transferred to $p_2, p_3, \ldots$ in the following way. To keep $\mathbb{E}[\mathcal{G}_{\mathrm{REMIX}}]$ as large as possible, p_2 is greedily set to its maximum, if there is any unassigned probability left, p_3 is set to its maximum, and so on. As $\mathbb{E}[\mathcal{G}^{(z)}_{\mathrm{ADV}}]$ does not depend on p_i for $i \ge z$, the values $\mathbb{E}[\mathcal{G}^{(z)}_{\mathrm{ADV}}]$ can be equalized with w_1 sequentially, with z increasing, until there is no unassigned probability left. Equalizing $\mathbb{E}[\mathcal{G}^{(j)}_{\mathrm{ADV}}]$ with $\mathbb{E}[\mathcal{G}^{(j-1)}_{\mathrm{ADV}}]$ consists in setting $p_{j-1} \leftarrow 1 - \frac{w_j}{w_{j-1}}$, as shown in (10). The same inequality shows what is intuitively clear: once there is no probability left and further values $\mathbb{E}[\mathcal{G}^{(z)}_{\mathrm{ADV}}]$ cannot be equalized, they are only smaller than w_1.

5.2 Tightness of Analysis

We present a simple *Packet Scheduling* instance that proves tightness of our analysis even in the oblivious adversary model; in fact, all the packets in this instance are released at once.

Let N be any positive integer and let $T \gg N$ be another integer. Let $h_1, h_2, \ldots, h_N$ be a sequence of packets such that h_i has weight $w_i = (1 - 1/N)^i$ and absolute deadline $D_i = T - i$. The instance starts at time 0, when T copies of each of $h_1, h_2, \ldots, h_N$ are released. Note that in each step up to $T - N - 1$ inclusively, REMIX will find a chain of N packets: copies of $h_1, h_2, \ldots, h_N$—this is because it ignores the abundance of copies, whose number is enough so that the algorithm does not run out of h_i's copies ($1 \le i \le N$) before they expire. It is easy to observe that REMIX will assign probability $1/N$ to a copy of h_i for each i in each of these steps, and that its expected gain in each of these will be $(1 - (1 - \frac{1}{N})^N) \cdot w_1$ because the sequence of weights $w_1, w_2, \ldots, w_N$ (and the resulting sequence of probabilities $p_1, p_2, \ldots, p_N$) makes the inequalities (10) and (16) tight. In each of those steps, ADV transmits a copy of h_1, gaining w_1 per step. The last $N - 1$ steps have negligible impact on both players' gains, since $N \ll T$. Hence, REMIX's competitive ratio on these instances is arbitrarily close to $1/(1 - (1 - \frac{1}{N})^N)$.

As one readily observes, these instances are rather artificial: should the algorithm consider only the packets from the optimum provisional schedule in each step, it would gain just as much as ADV; such modification of REMIX is justified by Lemma 1. The same is true for all the variations of this instance we could think of crafted for the oblivious adversary model, i.e., with packet releases specified upfront. We don't know any instance showing the tightness of our analysis of such a variation of REMIX in the oblivious adversary model. Therefore it is possible that such a variation of the algorithm attains better competitive ratio, if only in the oblivious adversary model.

In fact, in the adaptive adversary model our simple instance is easily modified in such a way that determining the optimum provisional schedule does not affect REMIX's behavior. Let us sketch the modification: initially, only a single copy of each of $h_1, h_2, \ldots, h_N$ is released, together with a tight (i.e., expiring at the beginning of the very next step) packet of weight $w_{N+1} := w_N \cdot (1 - 1/N) = (1 - 1/N)^{N+1}$. Note that introduction of this packet does not affect REMIX's probability distribution. In this step ADV transmits the packet ignored by REMIX. Then, for a number of steps much larger than N, say $T/2$, the adversary does the following: it releases a copy of the packet that REMIX last transmitted, and a single tight packet of weight w_{N+1}. In each such step ADV transmits the latter packet, whereas REMIX transmits a copy of one of $h_1, h_2, \ldots, h_N$, chosen uniformly at random. The expected gain of REMIX in each step is the same as in the previous instance, i.e., $(1-(1-\frac{1}{N})^N) \cdot w_1$. Afterwards, no further packets are released, and both players can transmit whatever packets are still pending for them. Again, REMIX's gain for that is insignificant, since $N \ll T$ but in the remaining $T/2$ steps ADV will transmit all the packets that REMIX transmitted in previous steps. Hence, one might think that in each of the first $T/2$ steps ADV gains w_{N+1} for transmitting its packet plus the packet that REMIX transmits, as it were in our amortized analysis. Then ADV's expected amortized gain in each of those steps is almost 1: precisely, it is $w_1 \cdot (1 - (1 - \frac{1}{N})^N) + w_{N+1} = w_1 \cdot (1 - (1 - \frac{1}{N})^N/N)$, which tends to 1 as N grows. Hence, the competitive ratio of REMIX on this instance also tends to $e/(e-1)$ as N tends to infinity.

5.3 Optimality for *Item Collection*

As we mentioned before, REMIX is optimal among randomized memoryless algorithms for *Item Collection* [3, Theorem 7]. In fact, as noted therein, the lower bound proof gives an infinite sequence of adversary's strategies parametrized by N such that the N-th one forces ratio $1/(1 - (1 - \frac{1}{N})^N)$, while ensuring that the number of packets pending for the algorithm never exceeds N. In such case REMIX is guaranteed to match that ratio, hence being the optimal randomized memoryless algorithm for *Item Collection* in a strong sense. We note though that, unlike the s-bounded instances of *Packet Scheduling*, any class of instances (parametrized by N) that would contain the ones defined in the lower bound construction would make little sense on their own: while at all times the algorithm has at most N packets pending, the lifespans of the packets are not restricted at all; in fact, the adversary successively "gathers copies" of the packets pending for the algorithm, only to transmit them all once no further packets are issued; while the proof is technically more involved, the basic idea in this

lower bound is similar to the one behind the second instance discussed in Sect. 5.2. Note that this lower bound gives independent evidence that (17) is indeed minimized by the heuristic described in Sect. 5.1.

6 Conclusion

Our algorithm REMIX is very simple to analyze, yet it unifies almost all previous randomized algorithms for packet scheduling (the optimum algorithm against oblivious adversary for 2-bounded instances [6] is the only exception), all the while providing new bounds for some restricted variants of the problem, thanks to its universal analysis. However unlikely it seems, it may potentially yield further non-trivial upper bounds, should one be able to confine the adversary's choice of packets for transmission for some class of instances in a similar manner to the one described in Sect. 3 for similarly ordered instances.

As explained in Sect. 5.3, REMIX is an optimal randomized memoryless algorithm for *Item Collection*. Therefore, to beat either the general bound of $e/(e-1)$, or any of the $1/(1-(1-\frac{1}{s})^s)$ bounds for s-bounded instances for *Packet Scheduling*, one either needs to consider algorithms that are not memoryless scale-invariant, or better utilize the knowledge of exact deadlines—in the analysis at least, if not in algorithm itself.

This might be achieved with the very same algorithm if H_0 is always initialized to (the set of packets included in) the optimum provisional schedule as discussed in Sect. 5.2. Note that resorting to an optimal provisional schedule takes the deadline values into account, so the lower bound for *Item Collection* no longer applies. It is conceivable that randomized algorithms might benefit from consulting the optimal provisional schedule, as there are already several algorithms that do so [9, 13, 16, 17]. However, among those the only randomized algorithm is the one for similarly ordered instances.

Bridging the gaps in any unresolved variant remains an obvious open problem. Let us point out that settling the question whether one may assume that an adaptive adversary obeys an EDF rule makes for another interesting question. Should the answer be affirmative, the $4/3$ upper bound for similarly ordered instances would apply not only to the oblivious but also adaptive adversary model, and would be optimal for the latter.

Acknowledgements This work was supported under the Polish Ministry's of Science and Higher Education grant N N206 368839, 2010–2013, *Approximation algorithms under uncertainty*, and Grant Agency's of the Academy of Sciences of the Czech Republic project IAA100190902, 2009–2013, *Mathematical logic, complexity, and algorithms*.

References

1. Azar, Y.: Online packet switching. In: Proc. of the 2nd Workshop on Approx. and Online Algorithms (WAOA), pp. 1–5 (2004)

2. Ben-David, S., Borodin, A., Karp, R.M., Tardos, G., Wigderson, A.: On the power of randomization in online algorithms. Algorithmica **11**(1), 2–14 (1994). Also appeared in Proc. of the 22nd ACM Symp. on Theory of Comput. (STOC), pp. 379–386 (1990)
3. Bienkowski, M., Chrobak, M., Dürr, C., Hurand, M., Jeż, A., Jeż, Ł., Stachowiak, G.: Collecting weighted items from a dynamic queue. Algorithmica (2011). doi:10.1007/s00453-011-9574-6. Also appeared in Proc. of the 20th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 1126–1135 (2009)
4. Bienkowski, M., Chrobak, M., Jeż, Ł.: Randomized competitive algorithms for online buffer management in the adaptive adversary model. Theor. Comput. Sci. **412**(39), 5121–5131 (2011). Also appeared in Proc. of the 6th Workshop on Approx. and Online Algorithms (WAOA), pp. 92–104 (2008)
5. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
6. Chin, F.Y.L., Chrobak, M., Fung, S.P.Y., Jawor, W., Sgall, J., Tichý, T.: Online competitive algorithms for maximizing weighted throughput of unit jobs. J. Discrete Algorithms **4**, 255–276 (2006)
7. Chin, F.Y.L., Fung, S.P.Y.: Online scheduling for partial job values: does timesharing or randomization help? Algorithmica **37**, 149–164 (2003)
8. Chrobak, M., Jawor, W., Sgall, J., Tichý, T.: Improved online algorithms for buffer management in QoS switches. ACM Trans. Algorithms **3**(4), 50 (2007). doi:10.1145/1290672.1290687, 19 pp. Also appeared in Proc. of the 12th European Symp. on Algorithms (ESA), pp. 204–215 (2004)
9. Englert, M., Westermann, M.: Considering suppressed packets improves buffer management in QoS switches. In: Proc. of the 18th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 209–218 (2007)
10. Epstein, L., van Stee, R.: Buffer management problems. SIGACT News **35**, 58–66 (2004)
11. Goldwasser, M.: A survey of buffer management policies for packet switches. SIGACT News **41**(1), 100–128 (2010)
12. Hajek, B.: On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time. In: Conf. in Information Sciences and Systems, pp. 434–438 (2001)
13. Jeż, Ł.: Randomized algorithm for agreeable deadlines packet scheduling. In: Proc. of the 27th Symp. on Theor. Aspects of Comput. Sci. (STACS), pp. 489–500 (2010)
14. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. SIAM J. Comput. **33**(3), 563–583 (2004). Also appeared in Proc. of the 33rd ACM Symp. on Theory of Comput. (STOC), pp. 520–529 (2001)
15. Li, F.: A comprehensive study of an online packet scheduling algorithm. In: Proc. of the 5th Int. Conf. on Comb. Optim. and Appl. (COCOA), pp. 52–63 (2011)
16. Li, F., Sethuraman, J., Stein, C.: An optimal online algorithm for packet scheduling with agreeable deadlines. In: Proc. of the 16th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 801–802 (2005)
17. Li, F., Sethuraman, J., Stein, C.: Better online buffer management. In: Proc. of the 18th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 199–208 (2007)
18. Zhu, A.: Analysis of queuing policies in QoS switches. J. Algorithms **53**(2), 137–168 (2004). Also appeared in Proc. of the 14th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 761–770 (2003)