Online Packet Scheduling with Bounded Delay and Lookahead*

Martin Böhm[1], Marek Chrobak[2], Łukasz Jeż[3], Fei Li[4], Jiří Sgall[5], and Pavel Veselý[6]

1 **Computer Science Institute of Charles University, Prague, Czech Republic**
bohm@iuuk.mff.cuni.cz

2 **Department of Computer Science and Engineering, University of California, Riverside, USA**
marek@cs.ucr.edu

3 **Institute of Computer Science, University of Wrocław, Poland**
lje@cs.uni.wroc.pl

4 **Department of Computer Science, George Mason University, USA**
lifei@cs.gmu.edu

5 **Computer Science Institute of Charles University, Prague, Czech Republic**
sgall@iuuk.mff.cuni.cz

6 **Computer Science Institute of Charles University, Prague, Czech Republic**
vesely@iuuk.mff.cuni.cz

Abstract

We study the *online bounded-delay packet scheduling problem (PacketScheduling)*, where packets of unit size arrive at a router over time and need to be transmitted over a network link. Each packet has two attributes: a non-negative weight and a deadline for its transmission. The objective is to maximize the total weight of the transmitted packets. This problem has been well studied in the literature, yet its optimal competitive ratio remains unknown: the best upper bound is 1.828 [6], still quite far from the best lower bound of $\phi \approx 1.618$ [10, 2, 4].

In the variant of PacketScheduling with *s-bounded instances*, each packet can be scheduled in at most s consecutive slots, starting at its release time. The lower bound of ϕ applies even to the special case of 2-bounded instances, and a ϕ-competitive algorithm for 3-bounded instances was given in [3]. Improving that result, and addressing a question posed by Goldwasser [8], we present a ϕ-competitive algorithm for 4-*bounded* instances.

We also study a variant of PacketScheduling where an online algorithm has the additional power of *1-lookahead*, knowing at time t which packets will arrive at time $t+1$. For PacketScheduling with 1-lookahead restricted to 2-bounded instances, we present an online algorithm with competitive ratio $\frac{1}{2}(\sqrt{13}-1) \approx 1.303$ and we prove a nearly tight lower bound of $\frac{1}{4}(1+\sqrt{17}) \approx 1.281$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases buffer management, online scheduling, online algorithm, lookahead

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2016.21

* M. Böhm, J. Sgall, and P. Veselý were supported by project 14-10003S of GA ČR and by the GAUK project 548214. M. Chrobak was supported by NSF grants CCF-1217314 and CCF-1536026. Ł. Jeż was supported by NCN grant DEC-2013/09/B/ST6/01538. F. Li was supported by NSF grant CCF-1216993.

1 Introduction

Optimizing the flow of packets across an IP network gives rise to a plethora of challenging algorithmic problems. In fact, even scheduling packet transmissions from a router across a specific network link can involve non-trivial tradeoffs. Several models for such tradeoffs have been formulated, depending on the architecture of the router, on characteristics of the packets, and on the objective function.

In the model that we study in this paper, each packet has two attributes: a non-negative weight and a deadline for its transmission. The time is assumed to be discrete (slotted), and only one packet can be sent in each slot. The objective is to maximize the total weight of the transmitted packets. We focus on the online setting, where at each time step the router needs to choose a pending packet for transmission, without the knowledge about future packet arrivals. This problem, which we call *online bounded-delay packet scheduling problem* (PacketScheduling), was introduced by Kesselman *et al.* [11] as a theoretical abstraction that captures the constraints and objectives of packet scheduling in networks that need to provide quality of service (QoS) guarantees. The combination of deadlines and weights is used to model packet priorities. In the literature, the PacketScheduling problem is sometimes referred to as *bounded-delay buffer management in QoS switches*. It can also be formulated as the job-scheduling problem $1|p_j = 1, r_j|\sum w_j U_j$, where packets are represented by unit-length jobs with deadlines, with the objective to maximize the weighted throughput.

A router transmitting packets across a link needs to make scheduling decisions on the fly, based only on the currently available information. This motivates the study of online competitive algorithms for PacketScheduling. A simple online greedy algorithm that always schedules the heaviest pending packet is known to be 2-competitive [10, 11]. In a sequence of papers [5, 7, 12, 6], this ratio was gradually improved, and the best currently known ratio is 1.828 [6]. The best lower bound, widely believed to be the optimal ratio, is $\phi = (1 + \sqrt{5})/2 \approx 1.618$ [10, 2, 4]. Closing the gap between these two bounds is one of the most intriguing open problems in online scheduling.

s-Bounded instances. In an attempt to bridge this gap, restricted models have been studied. In the *s-bounded* variant of PacketScheduling, each packet must be scheduled within k consecutive slots, starting at its release time, for some $k \leq s$ possibly depending on the packet. The lower bound of ϕ from [10, 2, 4] holds even in the 2-bounded case. A matching ϕ-competitive algorithm was given Kesselman *et al.* [11] for 2-bounded instances and by Chin *et al.* [3] for 3-bounded instances. Both results are based on the algorithm EDF_α, with $\alpha = \phi$, which always schedules the earliest-deadline packet whose weight is at least the weight of the heaviest pending packet divided by α (ties are broken in favor of heavier packets). EDF_ϕ is not ϕ-competitive for 4-bounded instances; however, a different choice of α yields a 1.732-competitive algorithm for the 4-bounded case [3].

We present a ϕ-competitive online algorithm for PacketScheduling restricted to 4-bounded instances, matching the lower bound of ϕ (see Section 3). This improves the results from [3] and answers the question posed by Goldwasser in his SIGACT News survey [8].

Algorithms with 1-lookahead. We investigate a variant of PacketScheduling where an online algorithm is able to learn at time t which packets will arrive by time $t + 1$. This property is known as *1-lookahead*. From a practical point of view, 1-lookahead corresponds to the situation in which a router can see the packets that are just arriving to the buffer and that will be available for transmission in the next time slot.

The notion of lookahead is quite natural and it has appeared in the online algorithm literature for paging [1], scheduling [13] and bin packing [9] since the 1990s. Ours is the first paper, to our knowledge, that considers lookahead in the context of packet scheduling.

We provide two results about PacketScheduling with 1-lookahead, restricted to 2-bounded instances. First, in Section 4, we present an online algorithm with competitive ratio of $\frac{1}{2}(\sqrt{13}-1) \approx 1.303$. Then, in Section 5, we give a lower bound of $\frac{1}{4}(1+\sqrt{17}) \approx 1.281$ on the competitive ratio of algorithms with 1-lookahead which holds already for the 2-bounded case.

2 Definitions and Notation

Formally, we define the PacketScheduling problem as follows. The instance is a set of packets, with each packet p specified by a triple (r_p, d_p, w_p), where r_p and $d_p \geq r_p$ are integers representing the *release time* and *deadline* of p, and $w_p \geq 0$ is a real number representing the *weight* of p. Time is discrete, divided into unit *time slots*, also called *steps*. A *schedule* assigns time slots to some subset of packets such that (i) any packet p in this subset is assigned a slot in the interval $[r_p, d_p]$, and (ii) each slot is assigned to at most one packet. The objective is to compute a schedule that maximizes the total weight of the scheduled packets, also called the *profit*.

In the *s-bounded* variant of PacketScheduling, we assume that each packet p in the instance satisfies $d_p \leq r_p + s - 1$. In other words, this packet must be scheduled within k_p consecutive slots, starting at its release time, for some $k_p \leq s$.

In the online variant of PacketScheduling, which is the focus of our work, at any time t only the packets released at times up to t are revealed. Thus an online algorithm needs to decide which packet to schedule at time t (if any) without any knowledge of packets released after time t.

As is common in the area of online optimization, we measure the performance of an online algorithm $\mathcal{A}$ by its competitive ratio. An algorithm is R-competitive if, for all instances, the total weight of the optimal schedule (computed offline) is at most R times the weight of the schedule computed by $\mathcal{A}$.

We say that a packet is *pending* for an algorithm at time t, if $r_p \leq t \leq d_p$ and p is not scheduled before time t. A (pending) packet p is *expiring* at time t if $d_p = t$, that is, it must be scheduled now or never. A packet p is *tight* if $r_p = d_p$; thus p is expiring already at its release time.

In Sections 4 and 5, we investigate the PacketScheduling problem *with 1-lookahead*. With 1-lookahead, the problem definition changes so that at time t, an online algorithm can also see the packets that will be released at time $t+1$, in addition to the pending packets. Naturally, only a pending packet can be scheduled at time t.

Other terminology and assumptions. We will make several assumptions about our problem that do not affect the generality of our results. First, we can assume that all packets have different weights. Any instance can be transformed into an instance with distinct weights through infinitesimal perturbation of the weights, without affecting the competitive ratio. Second, we assume that at each step there is at least one pending packet. (If not, we can always release a tight packet of weight 0 at each step.)

We define the *earliest-deadline relation* on packets, or *canonical ordering*, denoted $\prec$, where $x \prec y$ means that either $d_x < d_y$ or $d_x = d_y$ and $w_x > w_y$ (so the ties are broken in favor of heavier packets). At any step t, the algorithm maintains the earliest-deadline

relation on the set of its pending packets. Throughout the paper, "earliest-deadline packet" means the earliest packet in the canonical ordering.

Regarding the adversary (optimal) schedule, we can assume that it satisfies the following *earliest-deadline property*: if packets p, p' are scheduled in steps t and t', respectively, where $r_{p'} \leq t < t' \leq d_p$ (that is, p and p' can be swapped in the schedule without violating their release times and deadlines), then $p \prec p'$. This can be rephrased in the following useful way: at any step, the optimum schedule transmits the earliest-deadline packet among all the pending packets that it transmits in the future.

3 An Algorithm for 4-bounded Instances

In this section, we present a ϕ-competitive algorithm for 4-bounded instances. Ratio ϕ is of course optimal [10, 2, 4, see also Section 1]. Up until now, the best competitive ratio for 4-bounded instances was $\sqrt{3} \approx 1.732$, achieved by algorithm $\mathrm{EDF}_{\sqrt{3}}$ in [3]. Our algorithm can be seen as a modification of EDF_{ϕ}, which under certain conditions schedules a packet lighter than w_h/ϕ where h is the heaviest pending packet.

We remark that our algorithm uses memory; in particular, it marks one pending packet under certain conditions. It is an interesting question whether there is a memoryless ϕ-competitive algorithm for 4-bounded instances.

Our algorithm, which we call ToggleH, maintains one mark that may be assigned to one of the pending packets. For a given step t, we choose the following packets from among all pending packets:

- h = the heaviest packet,
- s = the second-heaviest packet,
- f = the earliest-deadline packet with $w_f \geq w_h/\phi$, and
- e = the earliest-deadline packet with $w_e \geq w_h/\phi^2$.

We then proceed as follows:

if (h is not marked) $\vee$ ($w_s \geq w_h/\phi$) $\vee$ ($d_e > t$)
  schedule f
  if there is a marked packet **then** unmark it
  if ($d_h = t+3$) $\wedge$ ($d_f = t+2$) **then** mark h
else // (h is marked) $\wedge$ ($w_s < w_h/\phi$) $\wedge$ ($d_e = t$)
  schedule e
  unmark h

Note that when $f \neq h$, then the algorithm will always schedule f. This is because in this case f is a candidate for s, so the condition $w_s \geq w_h/\phi$ holds. The algorithm never specifically chooses s for scheduling – it is only used to determine if there is one more relatively heavy pending packet other than h. (But s *may* get scheduled if it so happens that $s = f$ or $s = e$.) Note also that, if $e \neq f$, then e is scheduled only in a very specific scenario, when all of the following hold: e is expiring, h is marked, and $w_s < w_h/\phi$.

We have two types of packets scheduled by Algorithm ToggleH: *f-packets*, scheduled using the first case, and *e-packets*, scheduled using the second case. Similarly, we refer to the steps as *f-steps* and *e-steps*.

Let us give a high-level view of the analysis using charging schemes and an example that motivates both our algorithm and its analysis. The example consists of four packets j, k, f, h released in step 1, with deadlines $1, 2, 3, 4$ and weights $1-\varepsilon, 1-\varepsilon, 1, \phi$ for a small $\varepsilon > 0$, respectively. The optimum schedules all packets.

Algorithm EDF_ϕ performs only f-steps; in our example it schedules f and h in steps 1 and 2, while j and k are lost. Thus the ratio is larger than ϕ. (In fact, after optimizing the threshold and the weight of h, this is the tight example for $\mathrm{EDF}_{\sqrt{3}}$ on 4-bounded instances.) ToggleH avoids this example by performing e-step in step 2 and scheduling k which has the role of e and s in the algorithm.

This example and its variants are also important for our analysis. We analyze the algorithms by charging schemes, where the weight of each packet scheduled by the adversary is charged to one or more of the slots of the algorithm's schedule. If the weight charged to each slot is at most R times the weight of the packet scheduled by the algorithm in that slot, the algorithm is R-competitive. In the case of EDF, we charge the weight of each packet j scheduled by the adversary at time t either fully to the step where EDF schedules j, if it is before t, or fully to step t otherwise. In our example, the weight charged to step 1 is $2-\varepsilon$ while EDF schedules only weight 1, giving the ratio 2. Considering steps 1 and 2 together leads to a better ratio and after balancing the threshold it gives the tight analysis of $\mathrm{EDF}_{\sqrt{3}}$.

Our analysis of ToggleH is driven by the variants of the example above where step 2 is an f-step. This may happen in several cases. One case is if in step 2 another packet s with $w_s \geq w_h/\phi$ arrives. If s is not scheduled in step 2, then s is pending in step 3, thus ToggleH schedules a relatively heavy packet in step 3, and we can charge a part of the weight of f, scheduled in step 3 by the adversary, to step 3. This motivates the definition of regular up and back charges below and corresponds to Case 5.1 in the analysis. Another case is when the weight of k is changed to $1/\phi-\varepsilon$. Then ToggleH performs an f-step because k is not a candidate for e, thus the role of e is taken by the non-expiring packet h. However, then the weight of the four packets charged to steps 1 and 2 in the way described above is at most ϕ times the weight of f and h; this corresponds to Case 5.2 of the analysis. Lemma 3.3 gives a subtle argument showing that in the 4-bounded case essentially these two variants of our example are the only difficult situations. Finally, in the original example, ToggleH schedules k in step 2 which is an e-step. Then again h is a pending heavy packet and we can charge some weight of f to step 3. Intuitively it is important that an e-step is performed only in a very specific situation where it is guaranteed that h can be scheduled in the next two steps (as it is marked) and that there is no other packet of comparable weight due to the condition $w_s < w_h/\phi$. Still, there is a case to be handled: If more packets arrive in step 3, it is also possible that the adversary schedules h already in step 2 and we need to redistribute its weight. This case motivates the definition of the special up and back charges below.

▶ **Theorem 3.1.** *Algorithm ToggleH is ϕ-competitive on 4-bounded instances.*

Proof. Fix some optimal adversary schedule. Without loss of generality, we can assume that this schedule satisfies the earliest-deadline property (see Section 2).

Let t be the current step. By h, f, e, and s we denote the packets from the definition of ToggleH. By j we denote the packet scheduled by the adversary. By h' and h'' we denote the heaviest pending packets in steps $t+1$ and $t+2$, respectively. We use the same convention for packets f, e, s, and j.

Our analysis uses a new charging scheme which we now define. The adversary packet j scheduled in step t is charged according to the first case below that applies:

1. If t is an e-step and $j=h$, we charge w_h/ϕ to step t and w_h/ϕ^2 to step $t-1$. We call these charges a *special up charge* and a *special back charge*, respectively. Note that the total charge is equal to $w_h = w_j$.
2. If j is pending for ToggleH in step t, charge w_j to step t. We call this charge a *full up charge*.

3. Otherwise j is scheduled before step t. We charge w_h/ϕ^2 to step t and $w_j - w_h/\phi^2$ to the step where ToggleH scheduled j. We call these charges a *regular up charge* and a *regular back charge*, respectively. We point out that the regular back charge may be negative, but this causes no problems in the proof.

We start with an easy observation that we use several times throughout the proof.

▶ **Lemma 3.2.** *If an f-step t receives a regular back charge, then the up charge it receives is less than w_h/ϕ.*

Proof. For a regular up charge the lemma is trivial (with a slack of a factor of ϕ). For a full up charge, the existence of a back charge implies that the adversary schedules f after j, thus the earliest-deadline property of the adversary schedule implies that $j \prec f$, as both j and f are pending for the adversary at t. Thus ToggleH would schedule j if $w_j \geq w_h/\phi$. Finally, an f-step does not receive a special up charge. ◀

We examine packets scheduled by ToggleH from left to right, that is in order of time. For each time step t, if p is the packet scheduled at time t, we want to show that the charge to step t is at most ϕw_p. However, as it turns out, this will not always be true. In one case we will also consider the next step $t+1$ and the packet p' scheduled in step $t+1$, and show that the total charge to steps t and $t+1$ is at most $\phi(w_p + w_{p'})$.

Let t be the current step. We consider several cases.

Case 1: t is an e-step. By the definition of ToggleH, $w_e \geq w_h/\phi^2$ and $d_e = t$; the latter implies that step t receives no regular back charge. We further note that the heaviest pending packet h' in step $t+1$ is either released at time $t+1$ or it coincides with h, which is still pending and became unmarked by the algorithm in step t; in either case h' is unmarked at the beginning of step $t+1$, which implies that step $t+1$ is an f-step. Thus, step t receives no special back charge, which, combined with the previous observation, implies it receives no back charge of any kind.

Now we claim that the up charge is at most w_h/ϕ. For a special or regular up charge this follows from its definition. For a full up charge, the job j is pending at time t for ToggleH and $j \neq h$ (as for $j = h$ the special charges are used). This implies that $w_j < w_h/\phi$, as otherwise $w_s \geq w_h/\phi$ and t would be an f-step. Thus the full charge is $w_j \leq w_h/\phi$ as well.

Using $w_e \geq w_h/\phi^2$, the charge is at most $w_h/\phi \leq \phi w_e$ and we are done.

Case 2: t is an f-step and t does not receive a back charge. Then t can only receive an up-charge, and this up charge is at most $w_h \leq \phi w_f$, where the inequality follows from the definition of f.

Case 3: t is an f-step and t receives a special back charge. From the definition of special charges, the next step is an e-step, and therefore h' is marked at its beginning. Since the only packet that may be marked after an f-step is h, we thus have $h = h' = j'$, and the special back charge is w_h/ϕ^2. Since $f \prec h$, the adversary cannot schedule f after step t, so step t cannot receive a regular back charge.

We claim that the up charge to step t is at most w_f. Indeed, a regular up charge is at most $w_h/\phi^2 \leq w_f$, and a special up charge does not happen in an f-step. To show this bound for a full up charge, assume for contradiction that $w_j > w_f$. This implies that $j \neq f$ and, since ToggleH scheduled f, we have $d_j > d_f$. In particular j is pending at time $t+1$.

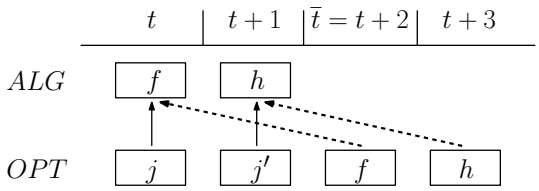

Figure 1 An illustration of the situation in Case 5.2. Up charges are denoted by solid arrows and back charges by dashed arrows.

Thus $w_{s'} \geq w_j > w_f \geq w_h/\phi$, contradicting the fact that $t+1$ is an e-step. Therefore the full charge is $w_j \leq w_f$, as claimed.

As $w_h \leq \phi w_f$, the total charge to t is at most $w_f + w_h/\phi^2 \leq w_f + w_f/\phi = \phi w_f$.

Case 4: t is an f-step, t receives a regular back charge and no special back charge, and $f = h$. The up charge is at most w_h/ϕ by Lemma 3.2 and the back charge is at most w_h, thus the total charge is at most $w_h + w_h/\phi = \phi w_h$, and we are done.

Case 5: t is an f-step, t receives a regular back charge and no special back charge, and $f \neq h$. Let $\bar{t}$ be the step when the adversary schedules f. We distinguish two sub-cases.

Case 5.1: In step $\bar{t}$, a packet of weight at least w_h/ϕ is pending for the algorithm. Then the regular back charge to t is at most $w_f - (w_h/\phi)/\phi^2 = w_f - w_h/\phi^3$. As the up charge to t is at most w_h/ϕ by Lemma 3.2, the total charge to t is at most $w_h/\phi + w_f - w_h/\phi^3 = w_f + w_h/\phi^2 \leq (1 + 1/\phi)w_f = \phi w_f$, and we are done.

Case 5.2: In step $\bar{t}$, no packet of weight at least w_h/ϕ is pending for the algorithm. In this case we consider the charges to steps t and $t+1$ together. First, we claim the following.

▶ **Lemma 3.3.** *ToggleH schedules h in step $t+1$. Furthermore, step $t+1$ receives no special charge and it receives an up charge of at most w_h/ϕ^2.*

Proof. Since $f \neq h$, we have $f \prec h$ and thus, using also the definition of $\bar{t}$ and 4-boundedness, $\bar{t} \leq d_f < d_h \leq t+3$. The case condition implies that h is not pending at $\bar{t}$, thus ToggleH schedules h before $\bar{t}$. The only possibility is that ToggleH schedules h in step $t+1$, $\bar{t} = d_f = t+2$, and $d_h = t+3$; see Figure 1 for an illustration. This also implies that ToggleH marks h in step t.

We claim that $w_{s'} < w_h/\phi$. Indeed, otherwise either s' is pending in step $t+2$, contradicting the condition of Case 5.2, or $d_{s'} = t+1 < d_h$, thus s' is a better candidate for f' than h, which contradicts the fact that the algorithm scheduled $f' = h$.

The claim also implies that $h' = h$, as otherwise $w_{s'} \geq w_h$. Since $h = h'$ is scheduled in step $t+1$, there is no marked packet in step $t+2$ and $t+2$ is an f-step; thus there is no special back charge to $t+1$.

We note that step $t+1$ is also an f-step, since ToggleH schedules h in step $t+1$ and $d_h > t+1$. Since $h' = h$ is marked when step $t+1$ starts and $w_{s'} < w_h/\phi$, the reason that step $t+1$ is an f-step must be that $d_{e'} > t+1$.

There is no special up charge to step $t+1$ as it is an f-step. If the up charge to step $t+1$ is a regular up charge, by definition it is at most $w_{h'}/\phi^2 = w_h/\phi^2$ and the lemma holds.

The only remaining case is that of a full up charge to step $t+1$ from a packet j' scheduled by the adversary in step $t+1$ and pending for ToggleH in step $t+1$. Since $j' \neq h$, it

is a candidate for s', and thus $w_{j'} < w_h/\phi \leq w_f$. The earliest-deadline property of the adversary schedule implies that $j' \prec f$; together with $d_f = t+2$ and $w_{j'} < w_f$ this implies $d_{j'} = t+1$. Therefore $w_{j'} < w_h/\phi^2$, as otherwise j' is a candidate for e', but we have shown that $d_{e'} > t+1$. Thus the regular up charge is at most $w_{j'} < w_h/\phi^2$ and the lemma holds also in the remaining case. ◀

By Lemma 3.3, step $t+1$ receives no special charge and an up charge of at most w_h/ϕ^2 and ToggleH schedules h in step $t+1$. Step $t+1$ thus also receives a regular back charge of at most w_h. So the total charge to step $t+1$ is at most $w_h/\phi^2 + w_h \leq w_f/\phi + w_h$. Moreover, using Lemma 3.2, the total charge to step t is at most $w_h/\phi + w_f$. Thus, the total charge to these two steps is at most $(w_h/\phi + w_f) + (w_f/\phi + w_h) = \phi(w_f + w_h)$, as f and h are the two packets scheduled by ToggleH.

In each case we have shown that a step or a pair of consecutive steps receive a total charge of at most ϕ times the weight of packets scheduled in these steps. Thus ToggleH is ϕ-competitive for the 4-bounded case. ◀

4 An Algorithm for 2-Bounded Instances with Lookahead

In this section, we present an algorithm for 2-*bounded* PacketScheduling *with 1-lookahead*, as defined in Section 2.

Consider some online algorithm $\mathcal{A}$. Recall that, for a time step t, packets *pending* for $\mathcal{A}$ are those that are released at or before time t and have neither expired nor been scheduled by $\mathcal{A}$ before time t. *Lookahead* packets at time t are the packets with release time $t+1$. For $\mathcal{A}$, we define the *plan* in step t to be the optimal schedule in the time interval $[t, \infty)$ that consists of pending and lookahead packets at time t and has the earliest-deadline property. For 2-bounded instances, this plan will only use slots t, $t+1$ and $t+2$. We will typically denote the packets in the plan scheduled in these slots by p_1, p_2, p_3, respectively. The earliest-deadline property then implies that if both p_1 and p_2 have release time t and deadline $t+1$ then p_1 is heavier than p_2 and similarly for p_2 and p_3.

Fix some parameter $\alpha > 1$. At any time step t, our algorithm CompareWithBias(α) proceeds as follows:

let p_1, p_2, p_3 be the plan at time t
if $r_{p_2} = t$ **and** $w_{p_1} < \min(\, w_{p_2}\,,\, w_{p_3}\,,\, \frac{1}{2\alpha}(w_{p_2} + w_{p_3})\,)$
  then schedule p_2
else schedule p_1

Note that if the algorithm schedules p_2 then p_1 must be expiring, for otherwise $w_{p_1} > w_{p_2}$ (by canonical ordering). Also, the scheduled packet is at least as heavy as the heaviest expiring packet q, since clearly $w_{p_1} \geq w_q$ and the algorithm schedules p_2 only if $w_{p_1} < w_{p_2}$.

▶ **Theorem 4.1.** *The algorithm* CompareWithBias(α) *is R-competitive for packet scheduling on 2-bounded instances for $R = \frac{1}{2}(\sqrt{13} - 1) \approx 1.303$ if $\alpha = \frac{1}{4}(\sqrt{13} + 3) \approx 1.651$.*

Let ALG be the schedule produced by CompareWithBias. Let us consider an optimal schedule OPT (a.k.a. schedule of the adversary) satisfying the canonical ordering, i.e., if a packet x is scheduled before a packet y in OPT then either y is released after x is scheduled or $x \prec y$. Recall that we are assuming w.l.o.g. that the weights of packets are different.

The analysis of CompareWithBias is based on a charging scheme. First we define a few packets by their schedule times:

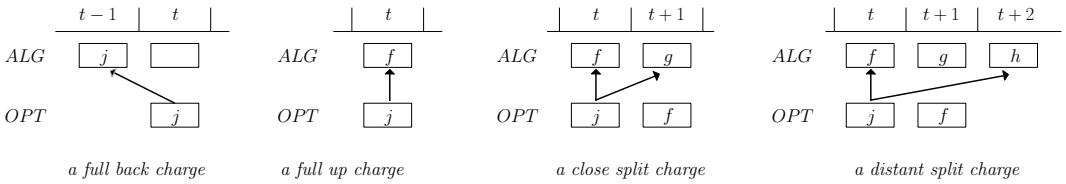

Figure 2 Non-chaining charges. Note that for split charges f is scheduled in step $t+1$ in OPT which follows from the fact that we do not charge j using a full up charge.

- j = packet scheduled in step t in OPT,
- f = packet scheduled in step t in ALG,
- g = packet scheduled in step $t+1$ in ALG.

Informal description of charging. We use three types of charges. The adversary's packet j in step t is charged using a *full charge* either to step $t-1$ if ALG schedules j in step $t-1$ or to step t if $w_f \geq w_j$ (including the case $f = j$) and f is not in step $t+1$ in OPT; the last condition assures that step t does not receive two full charges.

The second type are *split charges* that occur in step t if $w_f > w_j$, j is pending in step t in ALG and f is in step $t+1$ in OPT, i.e., step t receives a full back charge from f. In this case, we distribute the charge from j to f and another relatively large packet f' scheduled in step $t+1$ or $t+2$ in ALG; we shall prove that one of these steps satisfies $2\alpha w_j < w_f + w'_f$. We charge to step $t+2$ only when it is necessary, which allows us to prove that split-charge pairs are pairwise disjoint. Also, in this case we analyze the charges to both steps together, thus it is not necessary to fix a distribution of the weight to the two steps.

The remaining case is when $w_f < w_j$ and j is not scheduled in $t-1$ in ALG. We analyze these steps in maximal consecutive intervals, called *chains* and the corresponding charges are *chain charges*. Inside each chain we distribute the charge of each packet j scheduled at t in OPT to steps $t-1$, t and $t+1$, if these steps are also in the chain. The distribution of weights shall depend on a parameter δ. Packets at the beginning and at the end of the chain are charged in a way that minimizes the charge to steps outside of the chain. In particular, the step before a chain receives no charge from the chain.

Notations and the charging scheme. A step t for which $w_f < w_j$ and j is pending in step t in ALG is called *a chaining step*. A maximal sequence of successive chaining steps is called a *chain*. The chains with a single step are called *singleton chains*, the chains with at least two steps are called *long chains*.

The pair of steps that receive a split charge from the same packet is called a *split-charge pair*. The charging scheme does not specify the distribution of the weight to the two steps of the split-charge pair, as the charges to them are analyzed together.

Let $\delta = \frac{1}{6}(5 - \sqrt{13}) \approx 0.232$. Packet j scheduled in OPT at time t is charged according to the first rule below that applies. See Figures 2 and 3 for an illustration of different types of charges.

1. If j is scheduled in step $t-1$ in ALG, charge w_j to step $t-1$. We call this charge a *full back charge*.
2. If $w_f \geq w_j$ and f is not scheduled in step $t+1$ in OPT (in particular, if $j = f$), charge w_j to step t. We call this charge a *full up charge*.

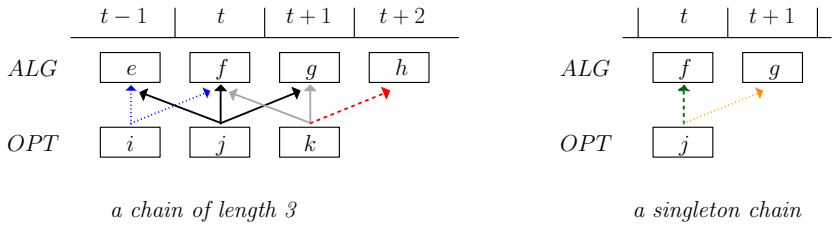

Figure 3 On the left, a chain of length 3 starting in step $t-1$ and ending in step $t+1$. The *chain beginning charges* are denoted by dotted (blue) lines, the chain end charges are denoted by gray lines and the *forward charge from a chain* is depicted by a dashed (red) arrow. Black arrows denote the *chain link charges*. On the right, an example of a singleton chain, with the *up charge from a singleton chain* denoted with a dashed (green) line and the *forward charge from a singleton chain* denoted with a dotted (orange) line.

3. If $w_f > w_j$ and at least one of the following holds:
   - $2\alpha w_j < w_f + w_g$,
   - g does not get a full back charge and $2\alpha(w_{p_1} - w_g) < w_f + w_g$ where p_1 is the first packet in the plan at time t,

   then charge w_j to the pair of steps t and $t+1$. We call this charge a *close split charge*.
4. If $w_f > w_j$, then charge w_j to the pair of steps t and $t+2$. We call this charge a *distant split charge*.
5. Otherwise step t is a chaining step, as $w_f < w_j$ and ALG does not schedule f in step $t-1$ by the previous cases. We distinguish the following subcases.
   a. If step t is (the only step of) a singleton chain, then charge $\min(w_j, Rw_f)$ to step t and $w_j - Rw_f$ to step $t+1$ if $w_j > Rw_f$. We call these charges an *up charge from a singleton chain* and a *forward charge from a singleton chain*.
   b. If step t is the first step of a long chain, charge $2\delta w_j$ to step t, and $(1-2\delta)w_j$ to step $t+1$. We call these charges *chain beginning charges*.
   c. If step t is the last step of a long chain, charge δw_j to step $t-1$, $(R-1+2\delta)w_f$ to step t, and $(1-\delta)w_j - (R-1+2\delta)w_f$ to step $t+1$. We call these charges *chain end charges*; the charge to step $t+1$ is called a *forward charge from a chain*. (Note that we always have $(1-\delta)w_j > (R-1+2\delta)w_f$, since $w_j > w_f$ and $1-\delta = R-1+2\delta$.)
   d. Otherwise, i.e., step t is inside a long chain, charge δw_j to step $t-1$, δw_j to step t, and $(1-2\delta)w_j$ to step $t+1$. We call these charges *chain link charges*.

The analysis of our charging scheme is omitted due to space limitation.

5 A Lower Bound for 2-bounded Instances with Lookahead

In this section, we prove that there is no online algorithm for PacketScheduling with 1-lookahead that has competitive ratio smaller than $\frac{1}{4}(1+\sqrt{17}) \approx 1.281$, even for 2-bounded instances. The idea of our proof is somewhat similar to the proof of the lower bound of ϕ for PacketScheduling [10, 2, 4].

▶ **Theorem 5.1.** *Let $R = \frac{1}{4}(1+\sqrt{17})$. For each $\varepsilon > 0$, no deterministic online algorithm for PacketScheduling with 1-lookahead can be $(R-\varepsilon)$-competitive, even for 2-bounded instances.*

Proof. Fix some online algorithm $\mathcal{A}$ and some $\varepsilon > 0$. We will show that, for some sufficiently large integer n and sufficiently small $\delta > 0$, there is a 2-bounded instance of PacketScheduling

with 1-lookahead, parametrized by n and δ, for which the optimal profit is at least $(R-\varepsilon)$ times the profit of $\mathcal{A}$.

Our instance will consist of phases $0,\ldots,k$, for some $k \le n$. In each phase $i < n$ we will release three packets whose weights will grow roughly exponentially from one phase to next. The number k of phases is determined by the adversary based on the behavior of $\mathcal{A}$.

The adversary strategy is as follows. We start with phase 0. Suppose that some phase i, where $0 \le i < n$, has been reached. In phase i the adversary releases the following three packets:

- A packet a_i with weight w_i, release time $2i+1$ and deadline $2i+1$, i.e., a tight packet.
- A packet b_i with weight w_{i+1}, release time $2i+1$ and deadline $2i+2$.
- A packet c_i with weight w_{i+1}, release time $2i+2$ and deadline $2i+3$.

(The weights w_i will be specified later.) Now, if $\mathcal{A}$ schedules an expiring packet in step $2i+1$ (a tight packet a_i or c_{i-1}, which may be pending from the previous phase), then the game continues; the adversary will proceed to phase $i+1$. Otherwise, the algorithm schedules packet b_i, in which case the adversary lets $k=i$ and the game ends. Note that in step $2i+2$ the algorithm may schedule only b_i or c_i, each having weight w_{i+1}. Also, importantly, in step $2i+1$ the algorithm cannot yet see whether the packets from phase $i+1$ will arrive or not.

If phase $i=n$ is reached, then in phase n the adversary releases a single packet a_n with weight w_n and release time and deadline $2n+1$, i.e., a tight packet.

We calculate the ratio between the weight of packets in an optimal schedule and the weight of packets sent by the algorithm. Let $S_k = \sum_{i=0}^{k} w_i$. There are two cases: either $k < n$, or $k = n$.

Case 1: $k < n$. In all steps $2i+1$ for $i < k$ algorithm $\mathcal{A}$ scheduled an expiring packet of weight w_i and in step $2k+1$ it scheduled packet b_k of weight w_{k+1}. In an even step $2i+2$ for $i \le k$ it scheduled a packet of weight w_{i+1}. Note that there is no packet scheduled in step $2k+3$. Overall, $\mathcal{A}$ scheduled packets of total weight $S_{k-1} + w_{k+1} + S_{k+1} - w_0 = 2S_{k+1} - w_k - w_0$.

The adversary schedules packets of weight w_{i+1} in steps $2i+1$ and $2i+2$ for $i < k$ and all packets from phase k in steps $2k+1$, $2k+2$ and $2k+3$. In total, the optimum has a schedule of weight $2S_{k+1} - 2w_0 + w_k$. The ratio is

$$R_k = \frac{2S_{k+1} + w_k - 2w_0}{2S_{k+1} - w_k - w_0}.$$

Case 2: $k = n$. As before, in all odd steps $2i+1$ for $i < n$ algorithm $\mathcal{A}$ scheduled an expiring packet of weight w_i and in all even steps $2i+2$ for $i < n$ it scheduled a packet of weight w_{i+1}. In the last step $2n+1$ it scheduled a packet of weight w_n as there is no other choice. Overall, the total weight of $\mathcal{A}$'s schedule is $2S_n - w_0$.

The adversary schedules packets of weight w_{i+1} in steps $2i+1$ and $2i+2$ for $i < n$ and a packet of weight w_n in the last step $2n+1$ which adds up to $2S_n - 2w_0 + w_n$. The ratio is

$$\widehat{R}_n = \frac{2S_n + w_n - 2w_0}{2S_n - w_0}.$$

We start with an intuitive explanation which leads to the optimal setting of weights w_i and the ratio R for the instances of the type described above. We normalize the instances so that $w_0 = 1$. We want to set the weights so that $R_k \ge R-\varepsilon$ for all $k \ge 0$ and $\widehat{R}_n \ge R-\varepsilon$. We first find the weights depending on δ such that $R_k = R$ for all $k \ge 1$. Using $w_k = S_k - S_{k-1}$ for $k \ge 1$ and $w_0 = 1$, the condition $R_k = R$ for $k \ge 1$ is rewritten as

$$R = \frac{2S_{k+1} + S_k - S_{k-1} - 2}{2S_{k+1} - S_k + S_{k-1} - 1}, \tag{1}$$

or equivalently as

$$(2R-2)S_{k+1} - (R+1)S_k + (R+1)S_{k-1} = -(2-R)\,. \tag{2}$$

A general solution of this linear recurrence with $S_0 = w_0 = 1$ and a parameter δ is

$$S_k = (\gamma+1)\alpha^k + \delta(\beta^k - \alpha^k) - \gamma\,, \tag{3}$$

where $\alpha < \beta$ are the two roots of the characteristic polynomial of the recurrence $(2R-2)x^2 - (R+1)x + (R+1)$ and $\gamma = (2-R)/(2R-2)$. To justify (3), a general solution is $A\alpha^k + B\beta^k - \gamma$ for parameters A and B and a suitable constant γ. Considering $A = B = 0$, the value $\gamma = (2-R)/(2R-2)$ follows. Considering the constraint $S_0 = 1$, we obtain $A + B = \gamma + 1$; our parametrization by δ in (3) is equivalent but more convenient for further analysis.

In our case of $R = \frac{1}{4}(1+\sqrt{17})$ a calculation gives

$$\alpha = R + \tfrac{1}{2} = \tfrac{1}{4}(3+\sqrt{17})\,, \quad \beta = R+1 = \tfrac{1}{4}(5+\sqrt{17}) \text{ and } \quad \gamma = R = \tfrac{1}{4}(1+\sqrt{17})\,. \tag{4}$$

A calculation shows that for $\delta = 0$, the solution satisfies $R_0 = R$. We choose a solution with a sufficiently small $\delta > 0$ which guarantees $R_0 \geq R - \varepsilon$. Since $1 < \alpha < \beta$, for large n, the dominating term in S_n is $\delta\beta^n$. Thus

$$\lim_{n\to\infty} \widehat{R}_n = \lim_{n\to\infty} \frac{2S_n + S_n - S_{n-1}}{2S_n} = \lim_{n\to\infty} \frac{3\delta\beta^n - \delta\beta^{n-1}}{2\delta\beta^n} = \frac{3\beta-1}{2\beta} = R\,. \tag{5}$$

The last equality is verified by a direct calculation; actually it is the equation that defines the optimal R for our construction (if β as the root of the characteristic polynomial of the recurrence is expressed in terms of R).

For a formal proof, we set $w_0 = 1$ and for $i = 1, 2, \ldots$,

$$w_i = (\gamma+1)\alpha^{k-1}(\alpha-1) + \delta(\beta^{k-1}(\beta-1) - \alpha^{k-1}(\alpha-1))\,,$$

where the parameters α, β and γ are given by (4) and $\delta > 0$ is sufficiently small. By a routine calculation we verify (3) and (2). Thus $R_k = R$ for $k \geq 1$. For R_0, we first verify that $\delta = 0$ would yield $w_1 = \alpha$ and $R_0 = R$. By continuity of the dependence of w_1 and R_0 on δ, for a sufficiently small $\delta > 0$, we have $R_0 \geq R - \varepsilon$; fix such a $\delta > 0$. Now, for $n \to \infty$, $S_n = \delta\beta^n + O(\alpha^n) = \delta\beta^n(1+o(1))$. Thus, the calculation (5) gives $\lim_{n\to\infty} \widehat{R}_n = R$. Consequently, $\widehat{R}_n \geq R - \varepsilon$ for a sufficiently large n of our choice. This defines the required instance and completes the proof. ◀

References

1. Susanne Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, 1997. doi:10.1007/PL00009158.
2. Nir Andelman, Yishay Mansour, and An Zhu. Competitive queueing policies for QoS switches. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*, pages 761–770, 2003.
3. Francis Y. L. Chin, Marek Chrobak, Stanley P. Y. Fung, Wojciech Jawor, Jiří Sgall, and Tomáš Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *J. of Discrete Algorithms*, 4(2):255–276, 2006.
4. Francis Y. L. Chin and Stanley P. Y. Fung. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica*, 37(3):149–164, 2003.

5 Marek Chrobak, Wojciech Jawor, Jiří Sgall, and Tomáš Tichý. Improved online algorithms for buffer management in QoS switches. In *Proc. 12th Annual European Symposium (ESA'04)*, pages 204–215, 2004.

6 Matthias Englert and Matthias Westermann. Considering suppressed packets improves buffer management in QoS switches. In *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, pages 209–218, 2007.

7 Matthias Englert and Matthias Westermann. Lower and upper bounds on FIFO buffer management in QoS switches. *Algorithmica*, 53(4):523–548, 2009.

8 Michael H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, 2010.

9 Edward F. Grove. Online bin packing with lookahead. In *Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95)*, pages 430–436, 1995.

10 Bruce Hajek. On the competitiveness of on-line scheduling of unit-length packets with hard deadlines in slotted time. In *Proc. Conference on Information Sciences and Systems*, pages 434–438, 2001.

11 Alexander Kesselman, Zvi Lotker, Yishay Mansour, Boaz Patt-Shamir, Baruch Schieber, and Maxim Sviridenko. Buffer overflow management in QoS switches. *SIAM Journal on Computing*, 33(3):563–583, 2004.

12 Fei Li, Jay Sethuraman, and Clifford Stein. An optimal online algorithm for packet scheduling with agreeable deadlines. In *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 801–802, 2005.

13 Rajeev Motwani, Vijay Saraswat, and Eric Torng. Online scheduling with lookahead: Multipass assembly lines. *INFORMS J. on Computing*, 10(3):331–340, 1998.