Packet Scheduling: Plans, Monotonicity, and the Golden Ratio

Pavel Veselý

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

Abstract

Online packet scheduling with deadlines is one of the fundamental models in buffer management. Recently, the author together with Chrobak, Jeż, and Sgall (SODA 2019) designed an optimal ϕ-competitive algorithm for this problem, where $\phi \approx 1.618$ is the golden ratio. In this column, we sketch ideas that led us to the development of this algorithm and outline the concepts in its analysis. We also highlight open questions in packet scheduling.

1 Introduction

In the *online packet scheduling problem with deadlines* (PacketSchD), packets arrive over time to a buffer of unlimited capacity to be transmitted over a network link. Each packet is characterized by its deadline d_p, corresponding to its urgency, and weight w_p, which represents its priority. In each time step, only one packet can be selected for transmission, while all packets with expired deadlines are dropped from the buffer. Naturally, the algorithm aims to maximize its profit, equal to the total weight of transmitted packets. This problem is also referred to as the *bounded-delay buffer management in QoS switches* and was introduced independently by Hajek [15] and Kesselman *et al.* [18]; in the former paper, Hajek stated it equivalently as a scheduling problem on a single machine with jobs of unit processing time characterized by weights, release times, and deadlines, where the goal is to maximize the weighted throughput.

While in the offline setting the problem reduces to a weighted bipartite matching, we are naturally interested in online algorithms and in particular, in determining the competitive ratio, which is the worst-case ratio between the optimal profit, computed by an offline algorithm, and the profit of the best online algorithm. Both Hajek [15] and Kesselman *et al.* [18] showed the 2-competitiveness of the Greedy algorithm that always transmits the heaviest pending packet, where a *pending packet* is any (non-expired) packet in the buffer. Hajek also discovered a lower bound of ϕ on the competitive ratio of any deterministic algorithm, where $\phi = (1+\sqrt{5})/2 \approx 1.618$ is the famous golden ratio, defined as the positive root of equation $\phi^2 = \phi + 1$. The same lower bound was independently proven by Andelman *et al.* [2, 25] and by Chin and Fung [9].

Improving upon the (very simple) 2-competitive Greedy is non-trivial. Chrobak *et al.* [10] were the first to beat it by giving a $64/33 \approx 1.939$-competitive algorithm that selects either the heaviest packet h or the earliest-deadline packet e with $w_e \geq (7/11) \cdot w_h$. Li, Sethuraman, and Stein [20] improved the ratio to $3/\phi \approx 1.854$, and Englert and Westermann [13] pushed the ratio further to $2\sqrt{2} - 1 \approx 1.828$, which remained the best result for over 10 years. The latter two algorithms rely on concepts similar to those described in Section 2. Many other papers made progress on this problem by studying restricted instances or by designing randomized algorithms; we outline these results in Section 6.

Recently, the author together with Chrobak, Jeż, and Sgall [24, 23] determined the competitive ratio by developing a deterministic ϕ-competitive algorithm, thus matching the aforementioned lower bound. This article is mainly devoted to providing an insight into this result, in a more

accessible way than in the full paper [24, 23], necessarily skipping some technical details and proofs (in fact, we only sketch the analysis of the algorithm, which would by no means fit into this column). Here, concepts are presented in a slightly different way than in [24, 23], similarly as we discovered them during 2016-2018. We refer an interested reader to the original paper [24] and its full version [23] for details and formal proofs. We hope that such an insight inspires other researchers and will lead to further exciting results!

Organization. Section 2 introduces the concept of a *plan* together with an algorithm, which is a "linear combination" of Greedy and its opposite. We delve into the structure of plans in Section 3, and we sketch why a certain monotonicity property is useful in Section 4, which also describes our ϕ-competitive algorithm. Section 5 provides a bird's eye view of the analysis. We close by giving several open problems in packet scheduling in Section 6, together with more related work.

Preliminaries. We need a few simplifying assumptions. First, we assume that all packet weights are distinct, which is without loss of generality (w.l.o.g.) by an infinitesimal perturbation of weights. Second, in any step t and for any $\tau \geq t$, we assume that there is a pending packet with deadline τ (or even more such packets if needed), which is w.l.o.g. by releasing virtual 0-weight packets with deadline τ.[1]

2 Why Not Make a Plan?

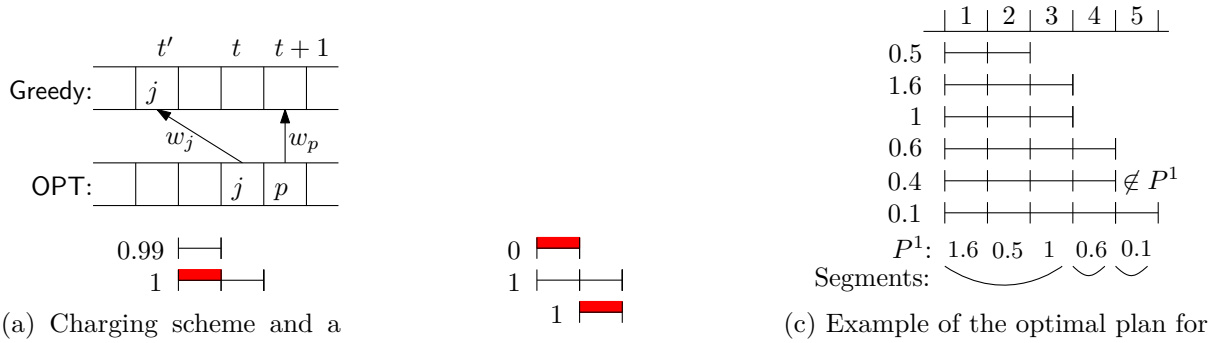
The first idea that may come to mind when designing an algorithm is to select the heaviest pending packet (i.e., the one with the largest weight) for transmission in each step. This Greedy algorithm is 2-competitive [15, 18], and its analysis is nearly as simple as the algorithm: The adversary profit is amortized using a charging scheme, which charges the adversary profit of w_j from transmitting a packet j in step t to the algorithm's profit in step t if j is pending for Greedy, and otherwise, to the step $t' < t$ in which Greedy sends j; see Figure 1(a) for an illustration. It is not difficult to complete the analysis, and we leave it as an exercise. Figure 1(a) also shows a tight instance, consisting of just two packets.

Greedy transmits the heaviest packet regardless of its deadline, thus in a sense making an implicit assumption that similarly heavy packets will arrive in subsequent steps. One may replace this overly "optimistic" approach by a "pessimistic" one: Transmit a packet that maximizes the total profit starting from the current step, assuming that no more packets arrive in subsequent steps. To describe this pessimistic approach more precisely, we define *plans*:

Definition 1. *A* plan X *at a certain time step* t *(w.r.t. an algorithm under consideration) is a feasible subset of pending packets, where feasible means that packets in* X *can be scheduled in slots* $t, t+1, \ldots$ *without violating their deadlines.*

Among many plans at time t, of particular importance is the *optimal plan*, denoted P^t, which is the plan of the largest total weight of packets; see Figure 1(c) for an example. A useful property is that P^t can be computed by the (offline) greedy algorithm that adds pending packets in the order of decreasing weights subject to the set remaining feasible; this works as the set of plans forms a matroid (see [18] for a more general proof). Thus, the optimal plan is unique under the assumption about distinct weights. Note that an (optimal) plan is only a set of pending packets and not a schedule (i.e., an assignment of packets to time slots). The algorithms in [20, 13] rely on the *optimal provisional schedule*, which is a schedule of the optimal plan with packets ordered by deadlines.

[1] In fact, by the first assumption, these virtual packets have distinct infinitesimal positive weights. Also, we gloss over some technical issues in these assumptions, which are discussed in [23].



(a) Charging scheme and a tight example for Greedy

(b) Tight example for PlanGreedy

(c) Example of the optimal plan for a set of pending packets

Figure 1: In the tight examples, each packet p is depicted by a line segment, split into slots in which the packet can be transmitted, and w_p appears to the left of this segment. The slot in which the algorithm transmits p (if any) is depicted by a red rectangle. In examples (a) and (b), the optimal schedule transmits both heavy packets, implying that the ratio is 2. In (a), we also illustrate a charging scheme for Greedy; note that packet p is pending in step $t+1$. In (c), the packet (of weight) 0.4 is not in the optimal plan P^1, and we show a particular schedule of P^1; note that, for example, the slots of packets 1.6 and 0.5 can be swapped. As defined in Section 3, the tight slots of P^1 are 3, 4, and 5, and the segments are $[1,3]$, $[4,4]$, and $[5,5]$.

Now, the pessimistic algorithm, that we call PlanGreedy, is as follows: Transmit packet $p \in P^t$ that maximizes the total weight of the optimal plan Q_p^{t+1} in the next step after transmitting p, assuming no packets arrive in the next step (ties are broken in favor of heavier packets). Naturally, the algorithm is not aware of packets arriving in the next step, and $p \notin Q_p^{t+1}$. The idea is that we are trying to "save" heavy packets for future steps, unless they are urgent. This algorithm can also be defined in a different way: Select a packet with the smallest deadline in the optimal plan P^t. (These two formulations are not really equivalent but the details are beyond the scope of this column.) As it turns out, PlanGreedy is also 2-competitive (by a more involved proof than for Greedy, using some ideas from Section 5), and a tight instance is depicted in Figure 1(b).

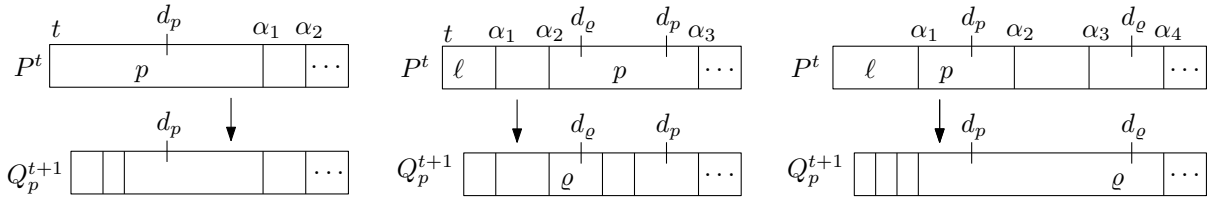
As one can observe, the tight instances for Greedy and PlanGreedy are somewhat opposite. The first key idea on the way to a ϕ-competitive algorithm is to consider a linear combination of these "opposite" algorithms. Namely, for a parameter $\alpha \geq 0$, consider algorithm Plan(α) that transmits packet p maximizing $\alpha \cdot w_p + w(Q_p^{t+1})$, where Q_p^{t+1} is defined in the same way as for PlanGreedy. Observe that for $\alpha = 0$, we get PlanGreedy, whereas for $\alpha \to \infty$ we obtain an algorithm arbitrarily close to Greedy. Naturally, the selected packet p is always in the optimal plan. An analysis of 2-bounded instances, where each packet can only be transmitted in one or two times steps, as well as other examples reveal that one should set the parameter α to ϕ in order to obtain a ϕ-competitive algorithm.

Unfortunately, Plan(ϕ) is *not* ϕ-competitive, and a (somewhat involved) counterexample is described in Appendix C in [23]. In order to understand why and to be able to fix it, we first need to describe the structure of the optimal plan and its changes after packet transmissions.

3 Properties of Plans

As defined above, an (optimal) plan is not a schedule, just a feasible set of pending packets. Thus, there is often some flexibility in assigning packets from P^t into time slots, and our goal is to understand this flexibility, using the following definition.

Definition 2. *A slot τ is* tight *w.r.t. the optimal plan P^t in step t if the number of packets in P^t with deadline at most τ equals $\tau - t + 1$.*



(a) Ordinary step (b) Leap step with $d_\rho \le d_p$ (c) Leap step with $d_\rho > d_p$

Figure 2: Illustration of changes in the optimal plan in three cases: (a) an ordinary step, (b) a leap step with $d_\rho \le d_p$, and (c) a leap step with $d_\rho > d_p$. Optimal plans are depicted by a rectangle, with tight slots given by vertical line segments.

Note that the number of slots $t, t+1, \ldots, \tau$ is $\tau - t + 1$ and thus, P^t could contain no more than $\tau - t + 1$ packets with deadline at most τ. Let $\alpha_1 < \alpha_2 < \alpha_3 < \cdots$ be the tight slots in P^t. Tight slots naturally divide time slots $t, t+1, \ldots$ into intervals, called *segments*. Namely, the segments are $[\alpha_0 + 1, \alpha_1], [\alpha_1 + 1, \alpha_2], \ldots$, where slot $\alpha_0 = t - 1$ is also considered to be tight. A moment's thought reveals that in any schedule of P^t, packets with deadline in $[\alpha_{i-1} + 1, \alpha_i]$ must be assigned to slots $[\alpha_{i-1} + 1, \alpha_i]$. Thus, packets in P^t are also divided into the segments by their deadlines, and a packet from one segment cannot be scheduled in another segment. Within a segment, there exists some limited flexibility and in particular, any packet in P^t can be assigned to the first slot of its segment. See Figure 1(c) for an example.

The first step in understanding $\mathsf{Plan}(\phi)$ (or any algorithm based on plans) is to figure out how the optimal plan changes just after the transmission of a packet p and incrementing the time but before new packets arrive in step $t + 1$; as above, we denote this new optimal plan by Q_p^{t+1}. We distinguish two types of transmissions by $\mathsf{Plan}(\phi)$ w.r.t. changes in the optimal plan:

"Ordinary" step: A packet p from the first segment $[t, \alpha_1]$ is transmitted. Then we simply have $Q_p^{t+1} = P^t \setminus \{p\}$, which can be proven using matroid properties of plans.

"Leap" step: The algorithm chooses a packet p with $d_p > \alpha_1$. In this case, the change in the optimal plan is more involved: Let ℓ be the lightest packet in the first segment, and let α_i be the tight slot such that $d_p \in [\alpha_i + 1, \alpha_{i+1}]$, i.e., p belongs to the segment just after α_i. Finally, let ρ be the heaviest pending packet not in P^t with $d_\rho > \alpha_i$. Then the new optimal plan is $Q_p^{t+1} = P^t \setminus \{\ell, p\} \cup \{\rho\}$. Intuitively, ℓ is pushed out because of incrementing the time step (as slot t cannot be used any more), while p is replaced by ρ. For this reason, ρ is called the *substitute packet* for p.

The structure of the new optimal plan, i.e., the tight slots, may change. Namely, in both cases, the first segment may be split into more segments as new tight slots may appear in (t, α_1). Moreover, for a leap step, the replacement of p by ρ may either cause new tight slots to appear in $[d_\rho, d_p)$ if $d_\rho \le d_p$, or in the case $d_\rho > d_p$, all tight slots in $[d_p, d_\rho)$ (if there are any) are no longer tight in Q_p^{t+1}, i.e., the segments containing d_p and d_ρ and any segments in between are merged. Otherwise, tight slots are preserved between P^t and Q_p^{t+1}. See Figure 2 for an illustration. Similarly, we can derive how the optimal plan changes after a packet arrival, but this is not going to be important in this article. An interested reader is referred to Appendix A in [23] for a detailed treatment of these changes in response to packet arrivals and transmissions.

4 It's All About Monotonicity

The next key step that led us to obtaining the result was to consider the function that for a given slot τ, captures the minimum weight that can appear in a slot $\tau' \le \tau$ in a schedule of P^t. More precisely, for a plan P^t in step t and a slot $\tau \ge t$, let $\mathsf{nextts}(P^t, \tau) = \min\{\alpha_i : \alpha_i \ge \tau\}$ be the next tight slot at or after τ, and define:

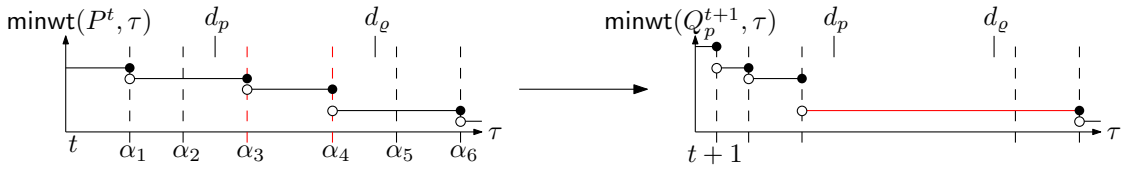


Figure 3: An example of a graph of minwt on the left. Tight slots are depicted by vertical dashed lines. The graph on the right shows how minwt changes after a leap step with $d_\rho > d_p$, when some segments get merged.

$$\mathsf{minwt}(P^t, \tau) = \min\{w_q : q \in P^t \ \&\ d_q \leq \mathsf{nextts}(P^t, \tau)\}.$$

We consider the next tight slot at or after τ because of the aforementioned flexibility in the schedules of P^t, namely, that any packet in P^t may appear in the first slot of its segment. Observe that $\mathsf{minwt}(P^t, \tau)$ for a fixed P^t is non-increasing for increasing τ and constant within a segment; see Figure 3 for an example.

This function is useful as it upper bounds the weights of packets not in the optimal plan P^t, i.e., for any pending packet $q \notin P^t$, it holds that $w_q < \mathsf{minwt}(P^t, d_q)$. Note that at any time step t, the adversary may release a packet with weight just below $\mathsf{minwt}(P^t, t)$ and deadline t (i.e., expiring immediately) without modifying the optimal plan P^t or the behavior of algorithm $\mathsf{Plan}(\phi)$.

Our focus will be on how the value of $\mathsf{minwt}(P^t, \tau)$ for a fixed τ changes as packets arrive and are transmitted. It is relatively easy to see that when a new packet arrives, $\mathsf{minwt}(P^t, \tau)$ does not decrease for any τ, and the same holds when a packet from the first segment is transmitted, i.e., in an ordinary step. However, after a leap step, the value of $\mathsf{minwt}(P^t, \tau)$ decreases for some τ, e.g., for $\tau = d_\rho$, as the substitute packet ρ is not in P^t, implying that $w_\rho < \mathsf{minwt}(P^t, d_\rho)$, but ρ gets added to Q_p^{t+1}.

Slot-monotonicity. One major step in designing a ϕ-competitive algorithm was to prove ϕ-competitiveness of $\mathsf{Plan}(\phi)$ on instances in which the following *slot-monotonicity property* holds: For any τ, the value of $\mathsf{minwt}(P^t, \tau)$ does not decrease in any step until τ. By the above discussion, these are instances on which $\mathsf{Plan}(\phi)$ doesn't execute leap steps (so these are not some "natural" instances as their definition is based on the behavior of a specific algorithm). This analysis of $\mathsf{Plan}(\phi)$ uses the concepts described in Section 5. On the other hand, in the aforementioned example with a ratio over ϕ for $\mathsf{Plan}(\phi)$, the algorithm executes a few leap steps, which cause significant decreases of $\mathsf{minwt}(P^t, \tau)$ for some slots τ.

And here comes the main idea: While the slot-monotonicity property doesn't hold on general instances, our algorithm *increases weights* and *decreases deadlines* of pending packets so that $\mathsf{minwt}(P^t, \tau)$ does not decrease for any τ, even in leap steps. Thus, for a packet p, w_p and d_p are no longer constant during the computation, and by w_p^t and d_p^t we denote the weight and deadline, respectively, of packet p in step t. Naturally, the actual profit of the algorithm from transmitting p is w_p^0, the original weight of p. However, the optimal plan P^t is always computed w.r.t. the current weights and deadlines of packets, i.e., taking the adjustments into account.

Figuring out the particular adjustments to maintain the slot-monotonicity property still requires some care. As argued above, this is only needed in leap steps, and the most obvious adjustment is to increase the weight of ρ to $\mathsf{minwt}(P^t, d_\rho^t)$. While this is sufficient when $d_\rho^t \leq d_p^t$, the value of $\mathsf{minwt}(P^t, \tau)$ may still decrease if $d_\rho^t > \mathsf{nextts}(P^t, d_p^t)$ as at least two segments of p get merged into one segment; see the right part of Figure 3. Hence, we want to avoid merging segments, and we achieve this by decreasing deadlines.

One of the simplest options is to set the new deadline of ρ to $\min(d_p^t, d_\rho^t)$, thus ensuring that $d_\rho^{t+1} \leq d_p^t$ (after the deadline adjustment), which prevents merging segments. In a way, we "shift" ρ from its segment in P^t to the segment of p. Unfortunately, this and other simple ways to avoid

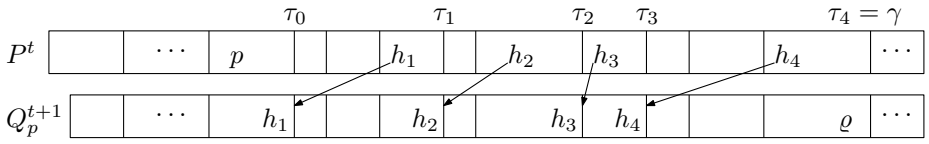


Figure 4: Illustration of the "iterative shift" of packets $h_1, \ldots, h_k$ in a leap step with $d^t_\rho > d^t_p$.

merging segments in a leap step do not yield a ϕ-competitive algorithm (see Appendix C in [23] for counterexamples). In a nutshell, the reason is that there may be a packet g that is nearly as heavy as p and belongs to a segment somewhere in between the segments containing d^t_p and d^t_ρ. If we decrease the deadline of ρ (or of another packet from the segment containing d^t_ρ) to d^t_p, then ρ would become more urgent than a possibly much heavier packet g, and this is intuitively not right — we should rather decrease the deadline of g.

Based on this observation, a right way to avoid merging segments is to do a rather involved "iterative shift" of certain, carefully selected heavy packets. Recall that tight slots of P^t belonging to $[d^t_p, d^t_\rho)$ would disappear after replacing p by ρ in the optimal plan unless we make suitable adjustments of deadlines. Let $\tau_0 = \mathsf{nextts}(P^t, d^t_p)$ and $\gamma = \mathsf{nextts}(P^t, d^t_\rho)$. First, if $d^t_\rho > \tau_0$ (i.e., d^t_ρ is in a later segment than d^t_p), we select the heaviest packet h_1 in P^t with deadline in $(\tau_0, \gamma]$. We decrease the deadline of h_1 to τ_0, which restores all tight slots of P^t in $[\tau_0, d^t_{h_1})$.[2] Next, we let $\tau_1 = \mathsf{nextts}(P^t, d^t_{h_1})$ and if $\tau_1 = \gamma$, then we're done. Otherwise, we iterate the above, with h_1 taking the role of p. Namely, we select h_2 as the heaviest packet in P^t with deadline in $(\tau_1, \gamma]$, decrease the deadline of h_2 to τ_1, and let $\tau_2 = \mathsf{nextts}(P^t, d^t_{h_2})$. Then, if $\tau_2 < \gamma$, we do another iteration, and so on, until $\tau_i = \gamma$. This iterative shift is illustrated in Figure 4.

This is not as simple as one would like and moreover, not even sufficient for the slot-monotonicity property. One last bit is needed: When we shift packet h_i to an earlier segment (by decreasing its deadline to τ_{i-1}), it may happen that $w^t_{h_i} < \mathsf{minwt}(P^t, \tau_{i-1})$, i.e., h_i is too light for its new segment. To rectify this final issue, we increase the weight of h_i to $\mathsf{minwt}(P^t, \tau_{i-1})$. With all these adjustments, the slot-monotonicity property holds and finally, it is possible to prove ϕ-competitiveness.

Algorithm 1 provides a pseudocode of the resulting algorithm PlanM. We use notation w^t_p and d^t_p for the weight and deadline, respectively, of a packet p in step t, before the algorithm transmits a packet (note that there are no adjustments done upon packet arrivals). We remark that line 1 corresponds to algorithm $\mathsf{Plan}(\phi)$, and is stated in a different, but equivalent way compared to [24, 23].

Algorithm 1 Algorithm $\mathsf{PlanM}(t)$

1: transmit packet $p \in P^t$ that maximizes $\phi \cdot w^t_p + w^t(Q^{t+1}_p)$ ▷ P^t is the optimal plan
2: **if** $d^t_p > \alpha_1$ **then** ▷ "leap step"
3: let ρ be the (only) packet in $Q^{t+1}_p \setminus P^t$
4: $w^{t+1}_\rho \leftarrow \mathsf{minwt}(P^t, d^t_\rho)$ ▷ increase w_ρ
5: $\gamma \leftarrow \mathsf{nextts}(P^t, d^t_\rho)$ and $\tau_0 \leftarrow \mathsf{nextts}(P^t, d^t_p)$
6: $i \leftarrow 0$
7: **while** $\tau_i < \gamma$ **do**
8: $i \leftarrow i + 1$
9: $h_i \leftarrow$ the heaviest packet in $P^t(\tau_{i-1}, \gamma]$
10: $\tau_i \leftarrow \mathsf{nextts}(P^t, d^t_{h_i})$
11: $d^{t+1}_{h_i} \leftarrow \tau_{i-1}$ and $w^{t+1}_{h_i} \leftarrow \max(w^t_{h_i}, \mathsf{minwt}(P^t, \tau_{i-1}))$ ▷ adjusting packet h_i

[2] One may ask why we don't set the new deadline of h_1 to d^t_p, instead of $\tau_0 \geq d^t_p$. The reason is that a certain argument in the analysis would not work, while using τ_1 is sufficient for slot-monotonicity.

(a) The first case

(b) The second case

Figure 5: Illustrations of the schedules of PlanM and of OPT in the two cases described in Section 5. The arrows depict our charging in these cases, and they are labeled with the amount being charged.

5 A Few Bits of the Analysis

Intuition. Before we introduce the main concepts used to prove ϕ-competitiveness of PlanM, it is helpful to consider the following two cases. The first case repeatedly appears in the lower bound of ϕ in [15, 2, 25, 9]. Namely, suppose that in step t, PlanM selected a packet e with $d_e = t$, while the adversary transmits a heavy packet h with $d_h > t$ that appears in the optimal plan P^t, and we have that $w_e = w_h/\phi^2$. (A calculation shows that if $w_e < w_h/\phi^2$, PlanM wouldn't transmit e. Here, we drop index t from weights and deadlines for simplicity.) In the analysis, the adversary's profit of w_h in step t is split into two parts: w_h/ϕ^2 is charged to the algorithm's profit of $w_e = w_h/\phi^2$ in step t, while the remaining part of w_h/ϕ (using $\phi^{-2} + \phi^{-1} = 1$) is charged to a future step $t' > t$ — if h is eventually transmitted by PlanM in step t'', then $t' = t''$; otherwise, t' is a step when another packet heavier than h is transmitted. See Figure 5(a) for an illustration.

Thus, the algorithm's profit of $\phi \cdot w_e$ (scaled up by ϕ to facilitate ϕ-competitiveness) is split into w_e used to cover w_h/ϕ^2, while the rest (i.e., w_e/ϕ) covers a possible charge from an adversary transmission before step t (which is the transmission of e or of a packet lighter than e). Intuitively, in such situations, the algorithm is only "catching up" with the adversary, which transmits heavy packets sooner. Consequently, the algorithm has a certain advantage in future steps, namely, still having those heavy packets in the buffer, and this is naturally quantified by a suitable potential function in the analysis.

Still, we are not done as the opposite case may happen: The algorithm transmits a relatively heavy packet p in step t that the adversary saves for a future step $t' > t$. As p is no longer pending in step t', the algorithm may select a very light packet at t', even of weight below w_p/ϕ^2. Thus, we are not able to charge w_p/ϕ^2 to the algorithm's profit in step t'. Instead, we amortize the adversary profit so that it receives an additional profit of $w_p - \mathsf{minwt}(P^t, t')$ in step t, while we decrease the adversary profit in step t' to $\mathsf{minwt}(P^t, t')$. As $\mathsf{minwt}(P^{t'}, t') \geq \mathsf{minwt}(P^t, t')$, a packet of weight at least $\mathsf{minwt}(P^t, t')$ is transmitted in step t'; here, we crucially rely on having the slot-monotonicity property. This amortization of the adversary profit, illustrated in Figure 5(b), comes in handy also in similar cases.

Such intuition (together with considering more involved examples) eventually led us to developing the following three amortization techniques, which we use to show $\phi \cdot w^0(\mathsf{ALG}) \geq w^0(\mathsf{OPT})$ for the schedule ALG computed by PlanM and the optimal schedule OPT (recall that w^0 refers to the original packet weights):

- In the analysis, the algorithm's profit in step t is the *current* weight of the transmitted packet that may have been increased in a previous step $t' < t$. Thus, when the algorithm increases the weight of a packet in a leap step t', we decrease its profit at t' by the weight increase (see (1) below). This ensures that the algorithm's profits sum up to $w^0(\mathsf{ALG})$.
- We amortize the optimal profit of $w^0(\mathsf{OPT})$ using function minwt and an "adversary stash"

A^t, briefly introduced below. Essentially, the adversary gets the profit from packets in A^t in a future step.

- We use a potential function that quantifies a certain advantage of the algorithm over the adversary in future steps. It is based on packets in a so-called "backup plan" B^t, which is a feasible set of pending packets used in the analysis. We maintain the invariant that $B^t \cap P^t = P^t \setminus A^t$ and that B^t is feasible. Intuitively, packets in $P^t \setminus A^t$ present an advantage for the algorithm, since they are pending for the algorithm and relatively heavy as they belong to the optimal plan, but the adversary won't get any profit for them in future steps as they do not appear in A^t. Furthermore, packets in $B^t \setminus P^t$ are available as potential substitute packets ρ in leap steps and thus, also present an advantage for the algorithm. Considering the first case above, where we charge w_h/ϕ from step t to step $t' > t$, we define the potential function as $\Psi^t := \frac{1}{\phi} w^t(B^t)$.

Adversary stash and amortization of $w^0(\mathsf{OPT})$. We need to keep track of the adversary future profit associated with already released packets. However, as suggested in the second example above, we may need to partially charge the adversary future profit to the current step, thus effectively decreasing its profit in a future step.

Another observation is that any released packet q in slot $\tau \geq t$ of OPT with $w_q < \mathsf{minwt}(P^t, \tau)$ doesn't need to be explicitly remembered. This is because of the slot-monotonicity property, i.e., that $\mathsf{minwt}(P^t, \tau)$ for the fixed τ does not decrease, and because in step τ, the adversary may release and transmit a packet with weight just below $\mathsf{minwt}(P^\tau, \tau)$ and deadline τ, which does not change the algorithm's behavior. As packets p of weight at least $\mathsf{minwt}(P^t, d_p)$ are in the optimal plan P^t, we only keep track of the adversary future packets that are in P^t.

In particular, we use an *adversary stash* A^t, which is a subset of packets in P^t together with their assignment to slots $t, t+1, t+2, \ldots$ (i.e., A^t is not only a set of pending packets but also their schedule). We maintain the invariant that $A^t \subseteq P^t$ and that no packet in A^t is assigned to a slot after its *current* deadline. When a packet q arrives in step t, if it is added to P^t and if it appears in a future step $t' > t$ in OPT, we add q to A^t and assign it to slot t'. Packet q may later be removed from A^t or even replaced by a lighter packet; the latter modification is used under certain conditions in a leap step for packets h_i. Removal of q from A^t happens, for example, when the current time t reaches t', or when q is transmitted by the algorithm (as in the second example above), or when w^t_q gets below $\mathsf{minwt}(P^t, t')$, i.e., when q is no longer in the optimal plan. There are also other cases in which we modify A^t, however, their description is beyond the scope of this informal overview.

We now briefly explain the amortization of the adversary profit of $w^0(\mathsf{OPT})$. In the analysis of step t, we consider two cases: If slot t of A^t contains a packet j, the adversary gets a credit of w^t_j. Otherwise, slot t is empty in A^t and the adversary credit is $\mathsf{minwt}(P^t, t)$. Furthermore, whenever we remove a packet q in slot τ of A^t, we give the adversary a credit of $w^t_q - \mathsf{minwt}(P^t, \tau)$ as a compensation for this change. Replacing a packet in A^t by another, lighter packet is also appropriately compensated by the credit equal to the weight difference. Let $\mathsf{advcredit}^t$ be the total credit the adversary receives in step t, including all the compensations for adjustments in A^t. Using the slot-monotonicity property, we can show that $\sum_t \mathsf{advcredit}^t \geq w^0(\mathsf{OPT})$, as required.

The analysis continues by proving that the potential Ψ^t does not decrease when new packets arrive, while we can maintain the feasibility of backup plan B^t and other invariants — this part is relatively straightforward. The main part of the analysis is to analyze packet transmissions. We need to preserve the invariants, while also showing the following inequality:

$$\phi \cdot w^t(\mathsf{ALG}[t]) - \phi \cdot (\Delta^t w) + \Delta^t \Psi \;\geq\; \mathsf{advcredit}^t\,, \tag{1}$$

where $\mathsf{ALG}[t]$ is the packet transmitted by PlanM in step t, $\Delta^t w$ is the total increase of packet weights in step t, and $\Delta^t \Psi$ is the change of the potential in step t. Summing up this inequality over all steps and using $\sum_t \mathsf{advcredit}^t \geq w^0(\mathsf{OPT})$, the proof of ϕ-competitiveness follows.

Admittedly, the resulting algorithm is slightly complicated (one would hope for just a couple of lines of pseudocode), and even worse, the full analysis in [23] with all the details and careful explanations takes about 40 pages. Thus, it would be interesting to see a simpler optimal algorithm or at least an algorithm admitting a simpler analysis. As mentioned in Section 4 and elaborated in Appendix C of [23], our attempts at simplifying PlanM failed.

6 Open Problems in Packet Scheduling

We have summarized ideas behind the optimal deterministic algorithm for PacketSchD. However, this is definitely not the end of the story of packet scheduling algorithms. Here, we list several open problems for future work, together with more related work. A more comprehensive list of open problems in packet scheduling and related models can be found in the author's PhD thesis [22] as well as in the excellent survey by Goldwasser [14] (also in SIGACT News), which is still mostly up-to-date, even though it's from 2010!

Memoryless algorithms. Apart from being a bit more complicated than one would wish, PlanM uses memory to maintain adjustments of packet properties (packets stored in the buffer do not count as a usage of memory). It is thus natural to ask if we can obtain the ratio of ϕ using a *memoryless* algorithm. Such algorithms are likely to be simpler, more practical, and faster, due to requiring less resources. A nice feature is that such algorithms make the same decision given the same buffer contents, i.e., the same set of pending packets; this could be exploited to improve the lower bound of ϕ for them. The 2-competitive Greedy is memoryless, and the only better such deterministic algorithm is the 1.893-competitive algorithm by Englert and Westermann [13].

Open Problem 1. *Design a ϕ-competitive memoryless deterministic algorithm for PacketSchD, or improve the lower bound for memoryless algorithms.*

A promising approach for improving the lower bound is to try to adapt the lower bound of 1.633 by Bieńkowski *et al.* [6] for Item Collection, a more general model where the algorithm is only aware of the ordering of packets by deadlines, not of actual values of the deadlines (before they expire).

Randomized algorithms. There is quite a substantial gap in our understanding of the competitive ratio of randomized algorithms. For both oblivious and adaptive adversaries, the best upper bound is $e/(e-1) \approx 1.582$ [3, 8, 7, 21], and it was proven even for a more general problem of online vertex-weighted bipartite matching [1, 11]. Chin and Fung [9] showed a lower bound of 1.25 for the oblivious case, while Bieńkowski *et al.* [7] obtained a lower bound of 4/3 against the adaptive adversary (both of these lower bounds only use simple 2-bounded instances, defined below). We believe that the recent progress in the deterministic case may inspire new randomized algorithms and in particular, it seems plausible that a better ratio can be attained in the oblivious case.

Open Problem 2. *Improve the bounds on the competitive ratio of randomized algorithms.*

Restricted variants of PacketSchD. Special cases considered in the literature are typically defined by restricting the packet *span*, that is, an interval of time slots in which the packet can be transmitted. In *s-bounded instances*, any packet span has at most s slots, for some $s \geq 2$, i.e.,

$d_p - r_p + 1 \leq s$ for any packet p, where r_p is the release time of p. Note that the lower bound of ϕ for deterministic algorithms holds for s-bounded instances for any $s \geq 2$ [15, 2, 25, 9]. Already Kesselman *et al.* [18] presented a ϕ-competitive algorithm for 2-bounded instances, which was later extended to the 3-bounded case by Chin *et al.* [8] and to 4-bounded instances by Böhm *et al.* [4]. These algorithms are based on selecting the earliest-deadline packet e with $w_e \geq w_h/\phi$, where h is the heaviest pending packet. However, we have some evidence that this algorithmic approach cannot be extended to the general case.

In instances with *agreeable deadlines*, it holds that $r_p < r_q$ implies $d_p \leq d_q$ for any packets p, q; they contain 2-bounded instances, so the lower bound of ϕ holds for agreeable deadlines. A matching ϕ-competitive algorithm for agreeable deadlines was given by Li *et al.* [19] (see also [16]). The lower bound of ϕ does not apply to *s-uniform instances*, in which each packet has span of length *exactly* $s \geq 2$. For 2-uniform instances, the optimal competitive ratio is ≈ 1.377 as shown by Chrobak *et al.* [10].

Finally, on instances without span restrictions but where packet weights increase with respect to deadlines, Bienkowski *et al.* [5] also proved an upper bound of ϕ (even for Item Collection mentioned above). A more comprehensive overview of restricted variants, including results on randomized algorithms, is given in [14, 22]. It appears that in the deterministic case, the main gaps between lower and upper bounds are for s-uniform instances.

Open Problem 3. *Design a better algorithm for s-uniform instances, ideally for any s, and construct lower bounds on the competitiveness in the s-uniform case.*

Additionally, one can consider instances in which the span of every packet is *at least* $s \geq 2$; note that the lower bound of ϕ does not apply in that case. To our best knowledge, such instances have not been (explicitly) studied in the literature yet, only the lower bound of 1.25 for randomized algorithms against the oblivious adversary carries over to instances with arbitrarily large spans [9, 4].

Higher bandwidth. A straightforward generalization of PacketSchD is to let the algorithm transmit $m \geq 1$ packets in each step, instead of just one. This was already proposed by Kesselman *et al.* [18], who show the 2-competitiveness of Greedy and a ϕ-competitive algorithm for 2-bounded instances, both for any $m \geq 1$. Chin *et al.* [8] designed an algorithm with ratio that tends to $e/(e-1) \approx 1.582$ and noted that the randomized lower bound of 1.25 holds for any m. As results for $m = 1$ translate to any m, our ϕ-competitive algorithm actually improves the state-of-the-art ratio for any $m < 13$.

Surprisingly, not only the ratio for $m \to \infty$ matches that of the currently best randomized algorithms, but also the algorithms and their analyses share some similarities. Hence, a new technique for higher bandwidth may translate into a better randomized algorithm, and vice versa.

Open Problem 4. *For the case of higher bandwidth $m > 1$, design better (deterministic or randomized) algorithms, and construct new lower bounds.*

FIFO model. Kesselman *et al.* [18] also introduced the FIFO model, in which packets do not have deadlines, but they are stored in a buffer of limited capacity and need to be transmitted in the same order as they arrive. For deterministic algorithms allowed to preempt (evict) a packet from the buffer, a simple greedy algorithm is 2-competitive [18], and there exists a better algorithm with the ratio of $\sqrt{3}$ [12], while the lower bound is only 1.419 [17]. We refer to [14] for more results and open problems in this model.

Acknowledgments. The author is partially supported by GA ČR project 19-27871X, by Charles University project UNCE/SCI/004, and by European Research Council grant ERC-2014-CoG 647557. The author is grateful to his coauthors in [24], Marek Chrobak, Łukasz Jeż,

and Jiří Sgall, for the fruitful collaboration on algorithms for PacketSchD, and also to Martin Böhm for many interesting discussions.

References

[1] Gagan Aggarwal, Gagan Goel, Chinmay Karande, and Aranyak Mehta. Online vertex-weighted bipartite matching and single-bid budgeted allocations. In *Proc. of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '11)*, pages 1253–1264, 2011.

[2] Nir Andelman, Yishay Mansour, and An Zhu. Competitive queueing policies for QoS switches. In *Proc. of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '03)*, pages 761–770, 2003.

[3] Yair Bartal, Francis Y. L. Chin, Marek Chrobak, Stanley P. Y. Fung, Wojciech Jawor, Ron Lavi, Jiří Sgall, and Tomáš Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. In *Proc. of the 21 st Symposium on Theoretical Aspects of Computer Science (STACS '04)*, volume 2996 of *LNCS*, pages 187–198. Springer Berlin Heidelberg, 2004.

[4] Martin Böhm, Marek Chrobak, Łukasz Jeż, Fei Li, Jiří Sgall, and Pavel Veselý. Online packet scheduling with bounded delay and lookahead. *Theoretical Computer Science*, 776:95 – 113, 2019.

[5] Marcin Bienkowski, Marek Chrobak, Christoph Dürr, Mathilde Hurand, Artur Jeż, Łukasz Jeż, and Grzegorz Stachowiak. A ϕ-competitive algorithm for collecting items with increasing weights from a dynamic queue. *Theoretical Computer Science*, 475:92 – 102, 2013.

[6] Marcin Bienkowski, Marek Chrobak, Christoph Dürr, Mathilde Hurand, Artur Jeż, Łukasz Jeż, and Grzegorz Stachowiak. Collecting weighted items from a dynamic queue. *Algorithmica*, 65(1):60–94, 2013.

[7] Marcin Bienkowski, Marek Chrobak, and Łukasz Jeż. Randomized competitive algorithms for online buffer management in the adaptive adversary model. *Theoretical Computer Science*, 412(39):5121–5131, 2011.

[8] 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.

[9] 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.

[10] Marek Chrobak, Wojciech Jawor, Jiří Sgall, and Tomáš Tichý. Improved online algorithms for buffer management in QoS switches. *ACM Trans. Algorithms*, 3(4), 2007.

[11] Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In *Proc. of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '13)*, pages 101–107, 2013.

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

[13] Matthias Englert and Matthias Westermann. Considering suppressed packets improves buffer management in quality of service switches. *SIAM Journal on Computing*, 41(5):1166–1192, 2012.

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

[15] Bruce Hajek. On the competitiveness of on-line scheduling of unit-length packets with hard deadlines in slotted time. In *Proc. of the 35th Conference on Information Sciences and Systems*, pages 434–438, 2001.

[16] Łukasz Jeż, Fei Li, Jay Sethuraman, and Clifford Stein. Online scheduling of packets with agreeable deadlines. *ACM Trans. Algorithms*, 9(1):5:1–5:11, 2012.

[17] A. Kesselman, Y. Mansour, and R. van Stee. Improved competitive guarantees for QoS buffering. *Algorithmica*, 43(1-2):63–80, 2005.

[18] 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.

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

[20] Fei Li, Jay Sethuraman, and Clifford Stein. Better online buffer management. In *Proc. of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*, pages 199–208, 2007.

[21] Łukasz Jeż. A universal randomized packet scheduling algorithm. *Algorithmica*, 67(4):498–515, 2013.

[22] Pavel Veselý. *Online Algorithms for Packet Scheduling*. PhD thesis, Computer Science Institute of Charles University, 2018.

[23] Pavel Veselý, Marek Chrobak, Lukasz Jez, and Jiří Sgall. A ϕ-competitive algorithm for scheduling packets with deadlines. *CoRR*, abs/1807.07177, 2018. Available at http://arxiv.org/abs/1807.07177.

[24] Pavel Veselý, Marek Chrobak, Lukasz Jez, and Jiří Sgall. A ϕ-competitive algorithm for scheduling packets with deadlines. In *Proc. of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '19)*, pages 123–142, 2019.

[25] An Zhu. Analysis of queueing policies in QoS switches. *J. Algorithms*, 53(2):137–168, 2004.