FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

Pavel Veselý

Online Algorithms for Packet Scheduling

Computer Science Institute of Charles University

Supervisor of the doctoral thesis:	prof. RNDr. Jiří Sgall, DrSc.
Study programme:	Computer Science
Study branch:	Discrete Models and Algorithms

Prague 2018

Title: Online Algorithms for Packet Scheduling

Author: Pavel Veselý

Institute: Computer Science Institute of Charles University

Supervisor: prof. RNDr. Jiří Sgall, DrSc., Computer Science Institute of Charles University

Abstract: We study online scheduling policies for buffer management models, in which packets are arriving over time to a buffer of a network switch to be sent through its single output port. However, the bandwidth of the port is limited and some packets need to be dropped, based on their weights. The goal of the scheduler is to maximize the weighted throughput, that is, the total weight of packets transmitted. Due to the natural lack of information about future, an optimal performance cannot be achieved, we thus pursue competitive analysis and its refinements to analyze online algorithms on worst-case inputs.

Specifically, in the first part of the thesis, we focus on a simple online scheduling model with unit-size packets and deadlines, called Bounded-Delay Packet Scheduling. We design an optimal ϕ-competitive deterministic algorithm for the problem, where $\phi \approx 1.618$ is the golden ratio. It is based on a detailed understanding of an optimal schedule of pending packets, called the plan, which may be of independent interest. We also propose a semi-online setting with lookahead that allows the algorithm to see a little bit of future, namely, packets arriving in the next few steps. We provide an algorithm with lookahead for instances in which each packet can be scheduled in at most two consecutive slots and prove lower bounds for both deterministic and randomized algorithms with lookahead.

In the second part, we consider a model with packets of various sizes and no deadlines, called Packet Scheduling under Adversarial Jamming. The hardness of scheduling decisions comes from unreliability of the channel through which packets are transmitted. This is modeled by an adversary that at any time may interrupt the current transmission by a jamming error. The corrupted packet is lost completely but may be retransmitted immediately or at any time later. The packets are weighted according to their size, thus the goal is to maximize the total size of successfully transmitted packets. We focus on online algorithms with the resource augmentation of speedup which allows the algorithm to run packets faster than the offline solution it is compared against. In particular, we propose an algorithm for which speedup of 4 suffices to be 1-competitive, i.e., to achieve an essentially optimal throughput. We complement it by a lower bound of $\phi + 1 \approx 2.618$ on the speedup of 1-competitive deterministic algorithms.

Keywords: online algorithm, competitive analysis, online scheduling, buffer management, resource augmentation

Acknowledgements

First of all, I would like to thank my advisor Jiří Sgall for his guidance and support during my master and doctoral studies, for showing me how science works, and for his numerous improvements of my write-ups. Thanks to him, I really enjoyed the last more than five years at MFF and I had learned a lot. I am especially grateful to him for involving me in every research project he was working on.

I had the honor to work several times with Marek Chrobak, to whom I particularly thank for hosting me for two weeks in Riverside in May 2016. There, the work on the ϕ-competitive algorithm for Bounded-Delay Packet Scheduling started and his eagerness to find it inspired me.

All results in this thesis are joint work with Łukasz Jeż. He introduced the model with adversarial jamming to us and visited us several times. I also thank him for hosting me in Wrocław and for his many comments and questions regarding preliminary versions of the proofs, which had lead to finding a lot of mistakes and gaps.

I am glad that I spent my PhD studies together with Martin Böhm with whom we collaborated a lot. In particular, I enjoyed working with him on the proof of PSPACE-hardness of computing the online chromatic number. He did a lot of great work on preparing good practicals, namely for *Optimization methods* and for *Introduction to approximation and randomized algorithms*. I was lucky to be a teaching assistant for these lecture after him.

Furthermore, I enjoyed working with Marcin Bieńkowski, Rob van Stee, Andreas E. Feldmann, and our “FPT group” of PhD students from IUUK and KAM departments.

Lastly, I wish to thank my parents, grandparents, and the whole family for inspiring me and supporting me. Most of all, I thank my beloved wife Lída for her lasting support in my studies and any activity I do and for inspiring me. Our little daughter Anežka makes me feel happy every day, even when everything else goes wrong.

Contents

1. Introduction to Online Computation

Suppose that you run a company with a launch system capable of carrying a payload, typically a satellite, into the Earth orbit once per each month. Private companies and governments request issuing satellites (or other types of payload) and each request comes with a revenue for your company which you obtain after launching the satellite and which may differ for different requests. However, each request also has a certain deadline after which launching the satellite is not relevant or meaningful and thus you obtain the revenue (or even the operation costs!) for a request only if you meet the deadline. For simplicity, let us assume that the launch system is very reliable and we thus do not expect any crash, and that we ignore the costs like salaries of the employees in a month in which the launch system is idle.

You face the following dilemma. There is an urgent request u for issuing a satellite which needs to be launched now or never and has revenue of \$100 million, and a request h with higher revenue of \$241.4 million which can be served also the next month. While you can serve both requests in two months, you have no information about requests that will arrive in future and in particular, if you decide to launch u, it may happen that the next month an urgent request h' arrives, also with revenue \$241.4 million, and then one of the requests h and h' will expire unserved and your company loses a lot of income. On the other hand, if you forfeit the urgent request u and deal with h, then if no request arrives during the next month, your launch system will be unused. The decision may be even harder if you have a lot of requests with various deadlines and revenues on the table.

The above situation is an example of a natural online optimization problem, where an algorithm needs to make decisions under the lack of information about future. This is in contrast with the usual algorithmic setting in which we have all the data, but we need to carry out a nontrivial computation using limited resources, typically as fast as possible or within a device with a limited memory. The online algorithm, on the other hand, has usually no time or memory limitations for its computation (in our example, you can spend a day deciding which satellite to launch), but often it decides very fast using a simple and efficient rule.

The decisions in the basic online setting are irrevocable, meaning that the algorithm cannot change its previous decisions. This is indeed natural, as otherwise we can view an algorithm which may change any of its previous decisions as an offline algorithm having the whole input in advance. This restriction and the lack of information about future typically make an optimal behavior on a particular instance impossible and thus the algorithm tries to produce a solution which is somehow close to an optimal solution.

An online model can be described in a fairly abstract and general way as follows: The input is a sequence of *events* of several types, each has its specified time t. At the beginning, the algorithm has no information about the input, it only knows the problem-specific setting and stays in the initial configuration. Then time flows and at time t, the algorithm knows all events with time $t' \leq t$, but no future event. Moreover, some of the events ask the algorithm to make *decisions*, which typically cannot be revoked or changed in future. The decisions somehow influence the *objective function* of the final solution, known to the algorithm, and the algorithm optimizes the value of the objective function. In the following, we assume for simplicity that the goal is to maximize a value, but minimizing a cost is very similar.

In our example, the problem specific scenario, known at the beginning, is that every month a launch system can serve one request and each request has a revenue and a deadline. Naturally, no request can be served more than once or after its deadline. The events are arrivals of requests, which reveal all information about the new request, and each month it is necessary to decide which request to serve if any (this can be thought of as "decision events"). The goal of the algorithm is to maximize the sum of revenues of the served requests.

1.1 Competitive Analysis

We want to design as good algorithm as possible and thus we need to compare algorithms. There are several ways how to do it. The one studied in this work looks at the *worst-case* instance and compares the objective value of the algorithm's solution and that of an offline optimal algorithm, which knows the whole input in advance (note that an offline optimal solution may be NP-hard to compute, or even harder). Namely, the *competitive ratio* of an online algorithm ALG is the supremum of $\mathsf{OPT}(I)/\mathsf{ALG}(I)$ over all possible instances I, where $\mathsf{OPT}(I)$ is the value of an offline optimal solution of I and $\mathsf{ALG}(I)$ is the (expected) value of the online algorithm when executed on input I. As we assume maximization in the objective function, the competitive ratio is always at least one.[1] We say that an algorithm is *R-competitive* if $\mathsf{OPT}(I) \le R \cdot \mathsf{ALG}(I)$, i.e., the optimum is at most R times better.

The idea of competitive analysis is usually attributed to Sleator and Tarjan [ST85], who applied it to simple list update and paging rules, although not using the term competitive ratio (the name "competitive analysis" itself was coined by Karlin *et al.* [KMRS88]). Worst-case analyses of online algorithms, serving only as simple approximation heuristics, appeared in the literature even before that; in particular, Graham's list scheduling algorithm for makespan minimization [Gra66] is considered to be the first work in online optimization.

The definitions of the competitive ratio and competitiveness are sometimes relaxed to avoid certain small pathological examples, on which any algorithm has a high ratio. Namely, the *asymptotic competitive ratio* of an algorithm ALG is

$$\limsup_{n\to\infty} \sup_I \left\{ \frac{\mathsf{OPT}(I)}{\mathsf{ALG}(I)} \,\middle|\, \mathsf{OPT}(I) \ge n \right\} .$$

In words, we compare the algorithm's solution and the optimum solution on the worst-case instance which is required to be large in the sense that the optimum value must be high. Similarly, an algorithm is *asymptotically R-competitive* if $\mathsf{OPT}(I) \le R \cdot \mathsf{ALG}(I) + o(\mathsf{OPT}(I))$; typically the additive term is just a constant. To stress that there is no additive term, the competitive ratio and competitiveness are called *absolute.*

If we want to analyze a deterministic online algorithm or find a lower bound on the competitive ratio of any deterministic algorithm, we may imagine a two-player game between the algorithm, which tries to maximize its value, and an adversary, which decides on further input, based on the previous decisions of the algorithm. The objective of the adversary is to maximize the ratio between the value of an optimal solution and the value of the algorithm's solution, that is, $\mathsf{OPT}(I)/\mathsf{ALG}(I)$. This adversarial behavior models the worst-case instance, or more precisely, an optimal strategy for the adversary yields an input for which the ratio $\mathsf{OPT}(I)/\mathsf{ALG}(I)$ equals the competitive ratio (or is arbitrarily close to it).

[1] We remark that some authors consider the reciprocal of our definition of the competitive ratio, i.e., the "alg-to-opt" ratio $\mathsf{ALG}(I)/\mathsf{OPT}(I)$, which is in turn always at most 1 for maximization problems. For minimization problems, the "alg-to-opt" ratio is used most frequently as it is at least 1.

For randomized algorithms, there are two possible ways how to model the adversary, distinguished by Ben-David *et al.* [BBK$^+$94]. The weaker *oblivious adversary* needs to fix the whole instance beforehand, based only on the algorithm's description and not on random bits that it will use, whereas the stronger *adaptive adversary* decides on further input based on previous random actions of the algorithm. However, the adaptive adversary also needs to make decisions over time, i.e., it cannot compute the optimal solution offline after the algorithm's computation is finished (thus it is sometimes called the *adaptive-online adversary*). Clearly, the adaptive adversary is more powerful, but still, there may be a randomized algorithm with competitiveness against the adaptive adversary smaller than the best possible competitive ratio of a deterministic algorithm. This is due to the requirement that the adaptive adversary needs to make decisions without any knowledge of future random bits used by the algorithm. In the following, when we talk about randomized algorithms, the competitive ratio is implicitly against the oblivious adversary unless we explicitly state that the ratio is against the adaptive adversary. Note also that both adversaries are the same against a deterministic algorithm, as the adversary may simply simulate the algorithm in advance, which is not possible if the algorithm's behavior is randomized.

Competitive analysis suffers from a few drawbacks. In many real-world situations, the adversarial worst-case input produces only strange instances which do not occur in practice, or the particular application does not require a guarantee on the worst-case behavior, but only that the algorithm works well in most situations. For the former, extensions of the competitive analysis were devised; we discuss some in Section 1.3. For the latter, stochastic models, where the input is generated from a certain distribution, are studied and then average case analysis is applied, i.e., the expected behavior of the algorithm is evaluated on the input distribution. However, from a theoretical perspective, it may not be clear which distribution is realistic or natural and such real-world distributions may be hard to analyze.

In contrast, competitive analysis provides guarantees in any situation and thus it is suitable for applications which require high reliability. We refer the interested reader who wants to learn more about competitive analysis to the book by Borodin and El-Yaniv [BE98].

1.2 Buffer Management

A special type of problems, naturally online, are buffer management problems, where the goal is to design a scheduling policy of a network switch transmitting packets over one or more channels. The network switch can have one or more input ports through which packets are arriving to be sent through one of output ports with a limited bandwidth. Usually, time is discrete, consisting of time steps or slots, and in each time step, a constant number of packets may be transmitted through each output port. Because of the limited bandwidth, packets must be stored in one of buffers inside the network switch and some of them need to be dropped.

There are various models of the network switch. Each input or output port may have its buffer for storing pending packets, or there may be a central buffer for storing all the packets. The buffer management models differ by the implementation of buffers, in particular, whether the buffers have limited capacity and whether reordering of packets inside a buffer is allowed. Also, the scheduling policy sometimes needs to deal with transferring packets inside the switch using internal fabric of the switch. See Figure 1.1 for an illustration.

The general aim is to provide a differentiated service, which allows for packet priorities. Typically, packets have weights, representing their importance, and possibly some other properties, specific to the model. The goal of the scheduler inside the switch is then to maximize the weighted throughput, i.e., the total weight of packets successfully transmitted.

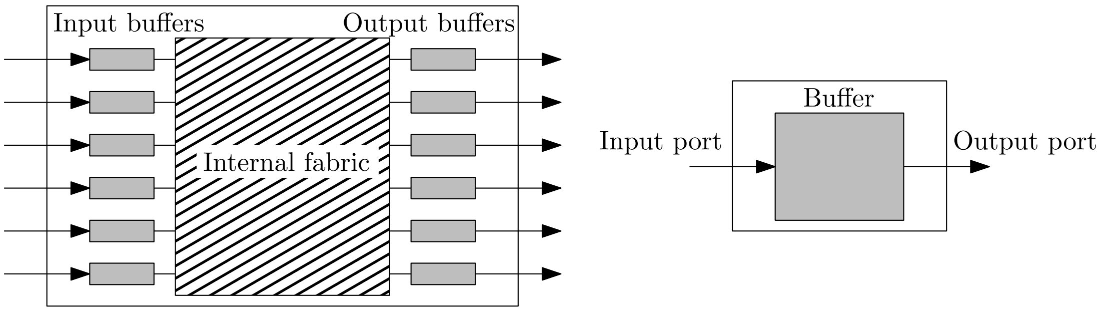


Figure 1.1: On the left a schematic illustration of a switch with multiple input/output ports, each of which has a dedicated buffer, and on the right an illustration of a single input/output switch.

We focus on the important case of one output port with a dedicated buffer. While there might be more input ports, we assume that all of them store arriving packets in the buffer of the output port and we can thus think of all input ports as being one port. Similarly, if the switch has several independent output ports with separated buffers, then we may apply the scheduling policy for a single output port to each of them (we assume that the internal fabric of the switch does not cause any delays). Lastly, we focus on the output port with bandwidth 1, i.e., one packet can be sent in each step.

There are two basic models of the switch with a single output port, in which non-trivial decisions must be made by an algorithm. Both were proposed by Kesselman *et al.* [KLM$^+$04] as abstractions of buffering policies used in network switches supporting Quality of Service (QoS). In particular, the motivation is to provide a differentiated network service, in which some clients may get a better level of service, for example, based on the price they pay for the service.

In the *FIFO model*, the buffer's capacity is limited and moreover, packets cannot be reordered inside the buffer. This means that packets need to be transmitted in the first-in-first-out (FIFO) order, and some packets must be dropped because of the limited capacity, depending on the weight. The packets have no deadlines, but the delay in sending each packet after its arrival is at most the buffer size if it is ever sent. We refer to Section 2.4 for a brief overview of the results for the FIFO model.

The second model assures *bounded delay* by packet deadlines, thus QoS is modeled by packet weights and deadlines. On the other hand, the buffer may hold an unlimited number of packets and packets can be reordered in the buffer. Equivalently, we can view the model as an online scheduling problem on a single machine where each job has unit processing time, integer release time and deadline and the goal is to maximize the weighted throughput; that is, $1|\mathit{online}, r_j, p_j = 1|\sum w_j(1-U_j)$ in the three field scheduling notation. Observe that this problem is an abstraction of the introductory example and we call it Bounded-Delay Packet Scheduling.[2] We study it in depth in Chapter 2; see Section 1.4.1 for a more detailed description of the problem and for a short overview of previous results and our contributions.

For an extensive survey, studying various buffer management models, we refer to a

[2]The problem is sometimes called just PACKET SCHEDULING, but as in this thesis we study two models of scheduling packets, we made explicit that in this one, the hardness is partially due to deadlines. In the literature, name *bounded-delay buffer management in QoS switches* is also used.

SIGACT News column by Goldwasser [Gol10].

1.3 Refinements of Competitive Analysis

In some cases, the adversary in the competitive analysis is so strong that no algorithm can be constant competitive, which may lead to results saying that the best possible (deterministic) algorithm is some greedy algorithm or another very simple or impractical algorithm. In such situations, there are a few possibilities which make sense:

- Restrict the input instances, i.e., weaken the adversary, so that the bad instances are no longer possible (generating the input from a distribution can be seen as an example).
- Reveal some information about the whole input at the beginning, such as the optimal value of the objective function, or a little bit of future in each step. Algorithms with some information about future are called *semi-online*.
- Give some advantage to the online algorithm and still compare it to the adversary without any advantage; this is the idea of *resource augmentation*, which appears already in the work of Sleator and Tarjan [ST85].

1.3.1 Resource Augmentation

In the seminal paper, Kalyanasundaram and Pruhs [KP00b] introduced resource augmentation in online scheduling for a variant of real-time scheduling on a single machine and for a flow-time minimization on a single machine. We describe the former problem only: Each job has a release time, deadline, processing time, and a weight and the objective is to maximize the weight of jobs completed by their deadlines. Preemption is allowed in this model, i.e., the algorithm may pause processing a job and resume it later. Constant competitive algorithms are possible with clairvoyance, which means that the algorithm knows all properties of a job upon its arrival. However, in some situations such as when an operating system schedules tasks on a processor, the processing time of a job is known only when the job completes. This property is called non-clairvoyance and there is no non-clairvoyant algorithm with a constant competitive ratio, even randomized; see e.g. [KP00a].

Kalyanasundaram and Pruhs [KP00b] proved that a constant competitive ratio for the non-clairvoyant real-time scheduling is possible with a constant speedup s which means that the machine of the algorithm runs s times faster than the machine of the adversary. In other words, an algorithm needs only time p/s to process a job of size p, while the adversary still needs to run the job for p units of time. Such an algorithm is often said to be speed-s, running at speed s, or having a speedup of s. Subsequently, resource augmentation was applied in various scenarios.

For real-time scheduling with clairvoyance, i.e., processing times known upon arrival, Phillips *et al.* [PSTW02] considered the underloaded case in which there exists a schedule that completes all the jobs. Observe that on a single machine, the *Earliest-Deadline First* (EDF) algorithm is then an optimal online algorithm. Phillips *et al.* [PSTW02] proved that EDF on m machines is 1-competitive with speedup $2 - \frac{1}{m}$. Intriguingly, finding a 1-competitive algorithm with the minimal speedup for $m > 1$ is wide open: It is known that speedup at least 1.2 is necessary, it has been conjectured that speedup $\frac{e}{e-1} \approx 1.582$ is sufficient, but the best upper bound proven is $2 - \frac{2}{m+1}$ from [LT99]. See the thesis of Schewior [Sch16] for more on this problem and for a related resource augmentation of adding extra machines to the online scheduler.

In buffer management models, where time is usually slotted, speedup corresponds to increasing the bandwidth for the algorithm, i.e., the algorithm sends m packets in each step, while the adversary transmits only $k < m$ packets; typically, $k = 1$. Jeżabek [Jeż09] considered this resource augmentation in the bounded-delay model and designed an algorithm with bandwidth m that is $1+1/(2^m-1)$-competitive against the offline optimum with bandwidth 1. On the other hand, he proved that no 1-competitive scheduling policy is possible even with an arbitrarily large bandwidth.

Another possible resource augmentation in the setting with a limited buffer is to allow the algorithm to use a larger buffer than the adversary. For the FIFO model, Kim [Kim05] studied how much resource augmentation is needed so that there is an optimal (i.e., 1-competitive) online algorithm. He proved that even with a much larger additional buffer this is not possible, but if we increase also the bandwidth to be s times larger than the transmission rate of the adversary, then an additional buffer of size $B/(s-1)$ suffices for a greedy algorithm in the preemptive case. For the nonpreemptive variant, the additional buffer size and the increase of the speed depend on α, the ratio of the maximum to the minimum weight of a packet; precisely, both are equal to $2(\lfloor \log \alpha \rfloor + 1)$ times the amount of the resource used by the adversary.

Resource augmentation also yields ways how to compare online algorithms on worst-case instances, alternative to the (sometimes too pessimistic) competitive ratio. Namely, we can ask how much resource augmentation, such as speed, is needed so that the algorithm is 1-competitive. This may lead to designing better algorithms than using the standard competitive analysis without the resource augmentation technique. We shall see an example in Section 1.4.2.

1.3.2 Semi-online Algorithms

As mentioned above, another possibility how to deal with strange or counter-intuitive results yielded by competitive analysis is to reveal a little bit of future to the algorithm, which is then called *semi-online*.

In some cases, revealing only the sole optimum value allows for a 1-competitive (i.e., optimal) semi-online algorithm. An example is preemptive scheduling on uniformly related machines, where machines have speeds. Jobs, characterized by their processing time only, arrive online in a list such that each needs to be scheduled before the next job arrives. Preemption allows the algorithm to pause processing a job and resume it later, possibly on a different machine. The goal is to minimize the makespan, which is the length of the schedule. The optimal semi-online algorithm, which gets the desired makespan on input, was given by Ebenlendr and Sgall [ES09]. Using the semi-online algorithm and a standard doubling technique, one can get a 4-competitive deterministic online algorithm and an $e \approx 2.718$-competitive randomized algorithm.

In buffer management, knowing the optimum value does not make much sense as one can concatenate several copies of one instance such that the copies are independent (i.e., the next copy starts when all the buffers are empty). However, it may be advantageous to know what happens in the near future, namely, all packets arriving during the next few steps. This property is called *lookahead*. We consider it quite natural, as it corresponds to the situation in which the network switch is able to observe packets that are just arriving in the buffer, yet they cannot be scheduled right now. Lookahead has appeared in the online algorithms literature for paging [Alb97], scheduling [MST98] and bin packing [Gro95] since the 1990s.

1.4 Contributions of the Thesis

In this work, we focus on buffer management models with a single input port and a single output port and in particular, on deterministic algorithms for the following two simple, yet interesting models.

1.4.1 Bounded-Delay Packet Scheduling

In Chapter 2 we study the following model: Packets have unit size and arrive in a buffer of unlimited capacity to be sent over a channel. Time is discrete, consisting of slots or steps of unit length, such that in each step, at most one packet can be transmitted. The number of packets arriving in the buffer is however very large and it is not possible to send them all. Packets are dropped based on their priorities, implemented by weights and deadlines. In particular, it is not possible to transmit a packet after its deadline and the goal is to maximize the total weight of scheduled packets. We remark that this model is an abstraction of our introductory example and it is also called *bounded-delay buffer management in QoS switches.*

Note that if packets have just deadlines and no weight, then the problem can be trivially and optimally solved by Algorithm EDF (*Earliest Deadline First*), which always schedules a pending packet with the smallest deadline. Similarly, if there are no deadlines, just weights, basically any reasonable algorithm clearly achieves optimal throughput. Thus having both weights and deadlines can be seen as (one of) the simplest interesting setting with unit-size packets and a buffer of unlimited size.

In this work we focus on deterministic online algorithms for this model; see Section 2.2 for a more extensive overview of results concerning both deterministic and randomized algorithms on general or restricted instances. There is a well-known lower bound of $\phi = \frac{1}{2}(\sqrt{5}+1) \approx 1.618$ [Haj01, AMZ03, CF03] on the competitive ratio of deterministic algorithms and it is a long-standing open problem to find a ϕ-competitive algorithm, or to improve the lower bound. So far the best algorithm due to Englert and Westermann [EW12] achieves ratio $2\sqrt{2}-1 \approx 1.828$.

Because of the hardness of the general case, the focus of the research shifted to special types of instances and for some of them, ϕ-competitive algorithms were found. First, in *s-bounded instances* the difference between the deadline and the release time of any packet p is at most $s-1$, i.e., each packet can be scheduled in at most s consecutive steps. The aforementioned lower bound of ϕ holds even in the 2-bounded case. A matching ϕ-competitive policy was given already by Kesselman *et al.* [KLM$^+$04] for 2-bounded instances and later by Chin *et al.* [CCF$^+$06] for 3-bounded instances. Recently, in [BCJ$^+$16] we designed a ϕ-competitive algorithm for 4-bounded instances.

Second, other interesting instances are those that satisfy the *agreeable property*, i.e., that packets arrive in the non-decreasing order by deadlines. For such instances, a ϕ-competitive algorithm was devised by Li, Sethuraman, and Stein [LSS05, JLSS12]. Finally, Bieńkowski *et al.* [BCD$^+$13b] studied the case of *increasing weights*, in which a packet with a larger deadline cannot have a smaller weight compared to another packet (the lower bound of ϕ satisfies this property as well). They gave a ϕ-competitive algorithm even for a more general model, in which the algorithm is aware only of the order of packets by deadlines and not the precise values of deadlines.

Contributions. Our focus is on general inputs. First, we investigate the technique of a *plan*, which is basically an optimal schedule of pending packets and which was used in [LSS05, JLSS12, EW12] as well. In Section 2.5 we study the structure of the plan in depth and we analyze its changes after the arrival of a new packet and after an algorithm schedules a packet.

Such a detailed understanding of the plan allows us to prove our main result which is a deterministic ϕ-competitive algorithm. We thus resolve its conjectured existence after more than 15 years. The basic idea underlying our algorithm is relatively simple. When some packet p from the plan is chosen to be scheduled at time t, it will be replaced in the plan by some other packet ϱ. The algorithm chooses p to maximize an appropriate linear combination of the weights of p and ϱ. For technical reasons, it also uses memory to increase weights and decrease deadlines of certain packets, which maintains a crucial monotonicity property. The competitive analysis relies on three amortization techniques: First, to avoid unfairly benefiting the algorithm from increased weights, we charge it a "penalty" equal to ϕ times the total weight increase. Second, we use a potential function, which quantifies the advantage of the algorithm over the adversary in future steps. Third, we modify the adversary schedule to maintain a crucial invariant that allows us to control decreases of the potential function.

We also introduce a semi-online setting with the so-called *lookahead*, which allows the algorithm to see a little bit of future, namely all packets (together with their properties) arriving in the next few steps. More precisely, at time t an algorithm with ℓ-lookahead is aware of all packets arriving in steps $t+1, t+2, \ldots, t+\ell$. Our work is the first, to our knowledge, that considers lookahead in the context of buffer management.

We provide two results about Bounded-Delay Packet Scheduling with 1-lookahead, restricted to 2-bounded instances. First, in Section 2.7.1, we present an online algorithm for this problem with competitive ratio of $\frac{1}{2}(\sqrt{13}-1) \approx 1.303$. Then, in Section 2.7.2, 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. Our argument is an extension of the lower bound proof of ϕ in [Haj01, AMZ03, CF03]. In fact, our lower bound result is more general: Using 2-bounded instances only, for any integer $\ell \geq 0$ we prove a lower bound of $\frac{1}{2(\ell+1)}(1+\sqrt{5+8\ell+4\ell^2})$ for online algorithms with ℓ-lookahead. It follows that there is no 1-competitive algorithm with any constant lookahead, even for 2-bounded instances. In a subsequent work, Kobayashi [Kob18] claims to have an optimal deterministic algorithm with 1-lookahead for 2-bounded instances, matching our lower bound of $\frac{1}{4}(1+\sqrt{17}) \approx 1.281$.

Finally, we argue that no randomized algorithm with an arbitrarily large lookahead can be better than 1.25-competitive against the oblivious adversary, even on instances with the agreeable property. The point is that packets have very large spans, which makes the use of lookahead negligible. The construction is then just a straightforward extension of the corresponding lower bound for 2-bounded instances without lookahead by Chin and Fung [CF03]; see Section 2.7.3.

1.4.2 Packet Scheduling under Adversarial Jamming

In Chapter 3 we study a model in which packets of various sizes arrive over time to a buffer of unlimited capacity to be sent over an unreliable channel. Unreliability is modeled by the adversary which has an additional power of issuing instantaneous jamming errors on the channel. The transmission taking place at the time of jamming is corrupt and completely lost, and the online algorithm learns this fact immediately. However, errors are unknown in advance to the online algorithm. The algorithm may retransmit the packet, whose processing failed, immediately or at any time later, but the retransmission must be complete, i.e., the algorithm cannot just resume the previous transmission of the packet. Moreover, the algorithm cannot pause or stop transmitting a packet; in the schedule jargon, preemption is not allowed. On the other hand, the optimum schedule (a.k.a. the adversary schedule) cannot transmit anything when an error occurs. The goal is to maximize the total size of packets successfully transmitted (thus the weight of each packet equals its size).

Another motivation appearing in the literature is the following: Suppose you have a machine and think of packets as computation tasks of different processing times, which you want to execute and which are injected to the system over time. However, the machine suffers from unexpected crashes or restarts, e.g., due to power outages, and the computation run before the crash is completely lost. The question is then how to schedule the computations to maximize the total length of successfully completed computations.

There are simple and small examples that show a lower bound of ℓ on the *absolute* competitive ratio of any deterministic algorithm, where ℓ is the size of the largest packet (see Section 3.1). To avoid such pathological instances, we study the asymptotic competitive ratio, where the additive constant may depend on packet sizes.

The model was introduced by Anta *et al.* [AGK$^+$16], who resolved it for two packet sizes: If $\gamma > 1$ denotes the ratio of the two sizes, then the optimum (asymptotic) competitive ratio for deterministic algorithms is $(\gamma + \lfloor\gamma\rfloor)/\lfloor\gamma\rfloor$, which is always in the range $[2, 3)$. Subsequently, Jurdziński *et al.* [JKL15] proved that the lower bound of Anta *et al.* [AGK$^+$16] is tight even in the case of multiple (though fixed) packet sizes by providing a deterministic algorithm, whose competitive ratio is given by the formula for the two packet sizes which maximize it. In particular, the algorithm achieves the optimal ratio of 3 on general instances. Jurdziński *et al.* [JKL15] also show more results for restricted inputs, mainly for *divisible* packet sizes, where each size divides every larger size, and for more channels.

Albeit the problem is solved for deterministic algorithms on general instances, these results have two drawbacks. First, the lower bound of 3 is relatively simple and it requires only two packet sizes 1 and $2 - \varepsilon$ (for a tiny $\varepsilon > 0$). On the other hand, the algorithm of Jurdziński *et al.* [JKL15] is not simple, since its description uses a recursive procedure and the algorithm needs to know all the packet sizes in advance. Moreover, the algorithm may be idle unnecessarily, i.e., not transmitting any packet even when there are some pending packets in the buffer.

Algorithms with the resource augmentation of speedup were also studied. Similarly to the real-time scheduling, an algorithm with speedup s transmits packets s times faster than the adversary. Jurdziński *et al.* [JKL15] proved that speedup 2 is sufficient in the case of divisible packet sizes, and optimality of their algorithm follows by the results of Anta *et al.* [AGKZ15]. Recently, Kowalski *et al.* [KWZ17] obtained tight bounds on speedup for 1-competitiveness when there are two packet sizes only.

Contributions. Our main contribution is a simpler and more universal deterministic online algorithm. In contrast to previous algorithms, ours does not need to know packet sizes in advance and is not unnecessarily idle, which makes it more suitable to be used in practice. We devise a local analysis framework which on general instances shows that the algorithm achieves the optimal competitive ratio of 3 and which can be easily applied on restricted instances as we demonstrate in various special cases.

However, the main question of our work is what speedup is sufficient and necessary for a deterministic algorithm to be 1-competitive. The local analysis mentioned above proves that our algorithm needs to run at speed at most 6 to be 1-competitive on general instances. With a more sophisticated non-local analysis we prove that speedup only 4 is sufficient which is our main result and which is tight for our algorithm. We complement our upper bound by showing that no deterministic algorithm is 1-competitive with speedup $s < \phi + 1 \approx 2.618$; our lower bound construction requires many sizes (the closer s is to $\phi + 1$, the more sizes are needed). This suggests that speedup sufficient for 1-competitiveness may be a better measure to compare online algorithms than the more usual competitive ratio (recall that the optimal lower bound of 3 on the competitive ratio needs just two sizes and that we can prove the 3-competitiveness of our algorithm by a simple, though somewhat technical, local analysis).

Golden ratio. As one can see, the golden ratio $\phi = \frac{1}{2}(\sqrt{5} + 1) \approx 1.618$ shows up naturally in the lower and upper bounds for both studied models. Its basic definition says that ϕ is equal to the length of a line segment divided into two segments of lengths 1 and $\phi - 1 < 1$ such that the ratio of the longer to the shorter segment is equal to the ratio of the whole line segment to the longer segment. That is, we have $\frac{1}{\phi-1} = \frac{\phi}{1}$ and by solving the quadratic equation we get $\phi = \frac{1}{2}(\sqrt{5}+1)$.

The golden ratio has several interesting mathematical properties which we use in our proofs. By definition, we have $\phi^2 = \phi + 1$ and more generally $\phi^n = \phi^{n-1} + \phi^{n-2}$. Another basic property is that the infinite sum $\sum_{i=-\infty}^{n} \phi^i$ equals ϕ^{n+2}, which holds as

$$\sum_{i=-\infty}^{n} \phi^i = \frac{\phi^{n+1}}{\phi - 1} = \frac{\phi^{n+1}}{\phi^{-1}} = \phi^{n+2} .$$

Similarly, the infinite sum decreasing by factor ϕ^{-2} from ϕ^n, i.e., $\phi^n + \phi^{n-2} + \phi^{n-4} + \phi^{n-6} + \dots$, equals ϕ^{n+1} (Figure 1.2 provides a proof by picture — the area of each square is an even power of ϕ, while the area of each rectangle is an odd power of ϕ).

The golden ratio appears in many natural mathematical scenarios. For example, ϕ is the limit of the ratio between two consecutive Fibonacci numbers. In geometry, the regular pentagon's ratio of a diagonal to a side is equal to ϕ as well. See Figure 1.2 for interesting geometrical objects called Golden rectangle and Golden spiral.

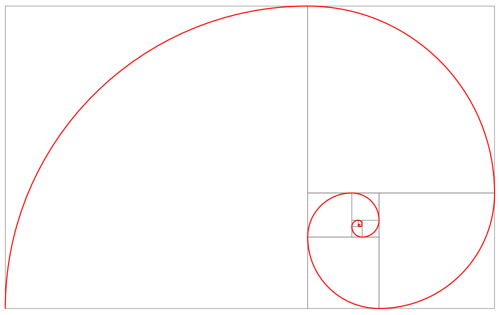


Figure 1.2: The Golden rectangle and the Golden spiral inscribed in red. The ratio of the sides of each rectangle is ϕ, and when we cut off a square from a rectangle, we get the Golden rectangle again, with sizes decreased by the factor of ϕ^{-1}. The Golden spiral is the logarithmic spiral with the growth factor of ϕ, that is, the spiral gets further from its center by the factor of ϕ.[4]

Due to its nice properties, the ratio appears in arts, for example, Salvador Dalí has laid out the composition of his painting *The Sacrament of the Last Supper* using the golden ratio. It is also sometimes used in photography to locate the point at which the human eye is considered to be drawn at first.

Outline of the chapters. In Chapter 2 we study Bounded-Delay Packet Scheduling and then in Chapter 3 we turn our attention to Packet Scheduling under Adversarial Jamming. Each of the two chapters is organized as follows: We start with a formal definition of the problem, together with the terminology used and other preliminaries. In the second section, we provide an overview of previous results and in the next section, we summarize the contributions of this thesis. The main part of the chapter consists of sections containing proofs of the results outlined above, i.e., algorithms and their analyses, or constructions of lower bounds. The general order is that algorithms come before lower bounds. Finally, we discuss possible future research directions in conclusions. Both chapters can be read independently of each other.

[4] Since this thesis is about online algorithms which produce only approximate solutions, the author approximated the Golden spiral by arcs, which form a shape very close to the real Golden spiral.

2. Bounded-Delay Packet Scheduling

In this chapter we study online algorithms for Bounded-Delay Packet Scheduling in depth and in particular, we resolve the open question whether there is a ϕ-competitive deterministic algorithm. The outline of the chapter is as follows:

- First, in Section 2.1 we define the problem formally and describe the terminology used, special types of instances studied in the literature, and other preliminaries.
- We continue by a survey of the previous work in Section 2.2.
- In Section 2.3 we outline the results contained in this chapter.
- Section 2.4 briefly describes several related models.
- Section 2.5 studies a technique of the plans in depth.
- In Section 2.6 we use this technique to design an optimal ϕ-competitive deterministic algorithm for general instances.
- In Section 2.7 we look at the semi-online setting with lookahead, both from the algorithmic and lower bound perspectives.
- Finally, in Section 2.8 we summarize our results and list several interesting open problems.

Section 2.5 about plans and Section 2.6 about our ϕ-competitive algorithm are based on the following paper.

[VCJS18] Pavel Veselý, Marek Chrobak, Łukasz Jeż, and Jiří Sgall. A ϕ-competitive algorithm for scheduling packets with deadlines. 2018. Submitted.

The results for the setting with lookahead in Section 2.7 are contained in the following paper (except for Section 2.7.3 with a lower bound for randomized algorithms):

[BCJ$^+$16] Martin Böhm, Marek Chrobak, Łukasz Jeż, Fei Li, Jiří Sgall, and Pavel Veselý. Online packet scheduling with bounded delay and lookahead. In *Proc. of the 27th International Symposium on Algorithms and Computation (ISAAC '16)*, volume 64 of *LIPIcs*, pages 21:1–21:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

2.1 Problem Definition and Preliminaries

Problem statement. Formally, we define the Bounded-Delay Packet Scheduling 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 some subset S of packets to time slots such that (i) any packet p in S is assigned to one slot in the interval $[r_p, d_p]$, and (ii) each slot is assigned at most one packet. The objective is to compute a schedule that maximizes the total weight of packets in S (the scheduled packets), also called the *profit*.

Other terminology. For a packet p, the interval $[r_p, d_p]$ is called the *span* of the packet; span also refers to the length of the interval plus 1, that is, the number of slots in which p can be scheduled. A *tight* packet p has a span of 1, i.e., $r_p = d_p$. See Figure 2.1 for an example.

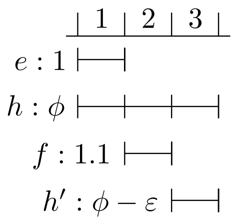


Figure 2.1: An example of an instance with four packets. Each packet is depicted by its identifier and weight (after the colon) and an interval representing its span. In this example, e, f and h' are tight packets, while h has span $[1, 3]$.

We say that a packet p is *pending* for an algorithm at time t, if $r_p \leq t \leq d_p$, and the algorithm have not scheduled p before t. Naturally, the algorithm may send only a pending packet. A (pending) packet p is *expiring* in step t if $d_p = t$; thus t is the last step in which p can be scheduled. Note that a tight packet is expiring already at its release time.

Online algorithms. In the online variant of Bounded-Delay Packet Scheduling, which is the focus of our work, in any step t only the packets released at times up to t are revealed, including all of their properties. Thus an online algorithm needs to decide which packet to schedule in step t (if any) without any knowledge of packets released after t.

As is common in the area of online optimization, we measure the performance of an online algorithm by its competitive ratio. An algorithm $\mathcal{A}$ is said to be *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 $\mathcal{A}$. See Chapter 1 for an introduction to competitive analysis.

Remark. We note that the asymptotic competitive ratio, i.e., considering an additive constant in the definition of competitiveness, does not bring any advantage to the online algorithm, even if the additive constant depends on the maximum weight in the instance. Indeed, any instance on which the algorithm achieves a bad absolute ratio (i.e., without any additive constant) can be copied many times and concatenated so that the copies do not interfere; namely, the adversary starts the next copy of the instance when both its buffer and the algorithm's buffer are empty. After sufficiently many repetitions, the effect of the additive constant vanishes.

Scale-invariant and memoryless algorithms. A natural property of most online algorithms is that they are *scale-invariant*, meaning that the decisions of the algorithm are not affected by scaling weights of all packets in the instance by a factor $\alpha > 0$. All algorithms mentioned in this chapter are scale-invariant, however, some lower bounds are only against scale-invariant algorithms.

There are two types of algorithms depending on the use of memory to remember something about history. An algorithm is *memoryless* if it uses no memory to store some information about its previous decisions or the sets of pending packets in previous steps, thus its decision depends solely on the current set of pending packets. Moreover, memoryless algorithms are usually assumed to be scale-invariant. Other algorithms are *memory-based* and may store even the whole history (including all packets that were released), but typically they just use a modest amount of memory.

Remark. Note that if any reasonable (online) algorithm is about to schedule a packet p, then there is no other pending packet q, heavier than p and with $d_q \leq d_p$ (otherwise, it is clearly better to schedule q). Any algorithm mentioned in this chapter has this property.

Algorithms with lookahead. In Section 2.7, we study the Bounded-Delay Packet Scheduling problem *with ℓ-lookahead*, mainly focusing on the case $\ell = 1$. With ℓ-lookahead, the problem definition changes so that at time t, an online algorithm can also see the packets that will be released at times $t+1, t+2, \ldots, t+\ell$, in addition to the pending packets. Naturally, only a pending packet can be scheduled at time t. This can be thought of as a semi-online setting (see Section 1.3.2).

From a practical point of view, lookahead corresponds to the situation in which a router can see the packets that are just arriving in the buffer and that will be available for transmission in the next few time slots.

Special types of instances. An instance is *s-bounded* if each packet p has span of at most s, i.e., $d_p \leq r_p + s - 1$. In other words, each packet must be scheduled within some specified number, at most s, of consecutive slots starting at its release time. The *s-uniform* variant further restricts the span of any packet to be exactly s, thus s-uniform instances form a special subclass of s-bounded instances. (Note that for $s = 1$ the problem becomes trivial.)

An instance has *agreeable deadlines* (or packets are *similarly ordered*) if for any two packets p and q with $r_p < r_q$ we have $d_p \leq d_q$. In other words, packets are released in order of non-decreasing deadlines. All 2-bounded instances and all s-uniform instances (for any s) have agreeable deadlines. For brevity, we sometimes call them *agreeable instances*. See Figure 2.2 for an illustration of inclusions of these types.

Finally, any instance of the previous type has the *increasing weights* property if a packet with a larger deadline has a larger weight. We remark that in all the lower bounds known so far, weights are actually increasing exponentially with respect to their deadlines and the lower bound instances are 2-bounded unless the lower bound is specifically for the s-uniform case.

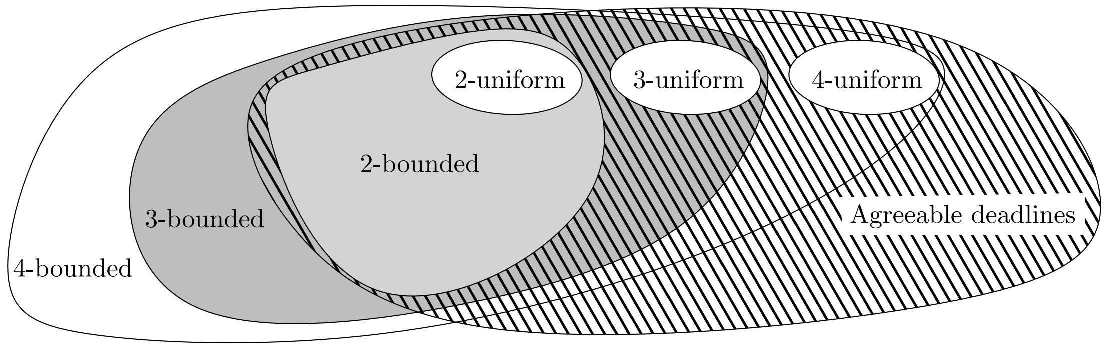


Figure 2.2: A Venn diagram of the main types of special instances and their inclusions.

Assumptions on the instance. We make two assumptions about our problem without loss of generality.

(A1) We assume that there are some pending packets in each step. If not, we can always release some "virtual" packets of weight 0 in each step.

When dealing with general instances, we assume that at each step t and for each $\tau \geq t$ (up to a certain large enough limit), there is a pending packet with deadline τ. This can be achieved by releasing, at time t, a virtual 0-weight packet with deadline τ, for each $\tau \geq t$.

(A2) We also 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. The 0-weight packets from the previous assumption thus, in fact, have an infinitesimal positive weight. The purpose of this assumption is to facilitate consistent tie-breaking, in particular uniqueness of plans (to be defined shortly).[1]

The plan and the canonical ordering. Consider an execution of an online algorithm $\mathcal{A}$. At any time t, $\mathcal{A}$ will have a set of pending packets. The *plan* is the maximum-weight subset of pending packets that can be scheduled starting from the current step; by assumption (A2) the plan is unique. In the literature, the plan (or more precisely, a schedule of the plan) is also called *optimal provisional schedule*. We investigate properties of plans in Section 2.5.

The set of pending packets has a natural ordering, called *canonical ordering* and denoted $\prec$, which orders packets in non-decreasing order of deadlines, breaking ties in favor of heavier packets. (By assumption (A2) the weights are distinct.) Formally, for two pending packets x and y, define $x \prec y$ iff $d_x < d_y$ or $d_x = d_y$ and $w_x > w_y$. The *earliest-deadline packet* in some subset X of pending packets is the first packet in the canonical ordering of X; if set X is not specified, then it is the first packet in the canonical ordering of all pending packets. Similarly, the *latest-deadline packet* in X is the last packet in the canonical ordering of X.

An (optimal) schedule satisfies the *earliest-deadline property* if for any two packets p, p' scheduled in steps t and t', respectively, such that $r_{p'} \leq t < t' \leq d_p$ (that is, p and p' can be swapped in the schedule without violating their release times and deadlines), $p \prec p'$ holds. This can be rephrased in the following useful way: at any step, the schedule transmits the earliest-deadline packet among all of its pending packets that it transmits in the future. Such a schedule is called *canonical*.

Offline optimal schedule. We remark that in the offline setting, where all packets are known, it is easy to find an optimal schedule. Indeed, we can apply any algorithm for the maximum-weight bipartite matching, since we can model the problem as a bipartite graph in which one partition consists of all possible slots and the other partition of all packets, and in which there is an edge of weight w_p for each packet p and each slot $t \in [r_p, d_p]$.

More specifically, we first compute the set of packets in the optimum schedule by adding them one by one in the order by decreasing weight, each time checking whether the new packet can be feasibly scheduled with other added packets. Observe that by Hall's condition for bipartite perfect matching, we just need to check whether for any time interval $[a, b]$ the number of accepted packets with the whole span inside $[a, b]$ is at most $b - a + 1$.

Having a set of packets that can be feasibly scheduled, we turn it into the canonical schedule by assigning to each slot t the earliest-deadline pending packet among the accepted packets. Using an interval tree and a preprocessing step to get rid of "underloaded" time intervals in which a greedy algorithm schedules all packets with span intersecting with the interval, this yields an algorithm running in $\mathcal{O}(n \log n)$.

Notation. In this chapter we use the following notation:

- lower-case letters like p or j denote packets,
- upper-case letter such as P or Q denote plans,

[1] Alternatively, instead of perturbing weights, one can extend the canonical ordering to a linear ordering of the set of all packets. Then the plan P is the maximum-weight feasible subset of pending packets with the property that for any $p \in P$ there is no $q \notin P$ with $w_q = w_p$ and $q \prec p$ such that $P \setminus \{p\} \cup \{q\}$ is feasible.

- t is the current time (step), and $\tau \geq t$ is a slot in the plan,
- ALG is the schedule of a particular online algorithm that we are considering,
- OPT is an optimal schedule, sometimes required to be canonical,
- ALG$[t]$ is the packet scheduled in step t in ALG and similarly, OPT$[t]$ is the packet at t in OPT,
- to avoid double indexing, we sometimes use $w(p)$ to denote w_p and $d(p)$ for d_p,
- for a set of (pending) packets X, let $X_{\leq\tau} = \{j \in X : d_j \leq \tau\}$ be the subset of X consisting of packets with deadline at most τ.

When considering some step t of a particular online algorithm, we use the following notation:

- h is the heaviest pending packet,
- $j = \mathsf{OPT}[t]$ is the packet scheduled in step t in OPT.

We remark that we do not use any specific notation for the set of packets pending for a particular online algorithm in the current step; pending packets are thus present only implicitly.

2.2 Previous Work

In this section we describe known results for Bounded-Delay Packet Scheduling, starting from deterministic policies for general and restricted instances, and ending with randomized algorithms. See Table 2.1 for a summary of the results.

2.2.1 Deterministic Algorithms

General instances. The model was introduced by Kesselman *et al.* [KLM$^+$04] at STOC 2001 as a theoretical abstraction that captures the constraints and objectives of packet scheduling in networks that need to provide Quality of Service (QoS) guarantees.

A simple online algorithm Greedy that always schedules h was shown to be 2-competitive already in 2001 [KLM$^+$04, Haj01, CY03]. The proof by a charging argument is simple as well: We charge a packet $j = \mathsf{OPT}[t]$ to the slot where the algorithm schedules j, if it is before t, and otherwise to the current slot. Observe that in the latter case, j is pending for the algorithm and thus $w_h \geq w_j$. Since any slot receives at most two charges, each of weight at most w_h, 2-competitiveness follows.

Also already in 2001, independently Hájek [Haj01] and Andelman *et al.* [AMZ03] showed a lower bound of $\phi = \frac{1}{2}(1 + \sqrt{5}) \approx 1.618$, so far the best one and widely believed to be the optimal ratio; the same lower bound was later proven by Chin and Fung [CF03] for restricted algorithms for online preemptive scheduling, but the proof carries over to our model. The lower bound is based on the basic dilemma in this model: Shall we take a heavy packet with a further deadline or an urgent but lighter packet? The construction actually uses packets with span at most 2 only and the dilemma is whether to send the heaviest expiring packet x or h with $w_h > w_x$, which has the deadline in the next step. If the algorithm chooses h, then the instance immediately ends, while if the algorithm takes x, then a new packet h with larger weight is issued in the next step and the dilemma repeats. The ratio of w_h to w_x starts from ϕ and gradually becomes ϕ^2; precisely, the weight of the packet with deadline t is $\phi^t + \delta \cdot \phi^{2t}$ for a small $\delta > 0$. For more details see Section 2.7.2 where we generalize this lower bound to the semi-online setting with lookahead.

The barrier of 2 was first broken by Chrobak *et al.* [CJST07] who presented a $\frac{64}{33} \approx 1.939$-competitive algorithm GenFlag. This algorithm balances between scheduling h and e, which is the earliest-deadline packet with weight at least $\frac{7}{11} w_h$. The algorithm uses one bit of memory to ensure that in two consecutive steps it does not schedule a packet e of weight close to the threshold of $\frac{7}{11}w_h$.

At SODA 2007, the upper bound was independently improved by Li, Sethuraman, and Stein [LSS07] to $3/\phi \approx 1.854$ and by Englert and Westermann [EW12] to $2\sqrt{2} - 1 \approx 1.828$, which is the best currently known ratio. The only better than 2-competitive *memoryless* algorithm is due to Englert and Westermann [EW12]; its ratio is approximately 1.893. All these three algorithms are based on plans (a.k.a. optimal provisional schedules), which we describe in detail in Section 2.5, together with these algorithms. While for the algorithm DP of Li *et al.* [LSS07] there is an example forcing a ratio of 1.764, no instance forcing a ratio over ϕ is known for the algorithms of Englert and Westermann.

In a recent paper, Al-Bawani, Englert, and Westermann [ABEW18] study deterministic *comparison-based* algorithms, which do not look at actual weights, but only at the relative order of packets by their weights. They show that Algorithm Greedy is the best such algorithm by providing a lower bound of 2. (In contrast, for the closely related FIFO model, defined in Section 2.4, they give a lower bound of $1+1/\sqrt{2} \approx 1.707$ and a matching comparison-based algorithm for sequences with increasing weights.)

However, the general lower bound still remains $\phi \approx 1.618$ and therefore, in an attempt to bridge the gap, special types of instances were studied.

s-Bounded instances. As mentioned above, the lower bound of ϕ holds even in the 2-bounded case. A matching ϕ-competitive algorithm was given by Kesselman *et al.* [KLM+04] for 2-bounded instances and by Chin *et al.* [CCF+06] for 3-bounded instances. Both results are based on Algorithm EDF_α (*Earliest Deadline First*), with $\alpha = \phi$, which always schedules the earliest-deadline packet f whose weight is at least w_h/α (ties are broken in favor of heavier packets).

EDF_ϕ is not ϕ-competitive for 4-bounded instances by the following example: There are four packets j, k, f, h released in step 1, with deadlines $1, 2, 3, 4$ and weights $1-\varepsilon, 1-\varepsilon, 1, \phi$, respectively, for some small $\varepsilon > 0$. The optimum schedules all packets, while EDF_ϕ transmits only f and h in steps 1 and 2, thus the ratio is $(3 + \phi - 2\varepsilon)/(1 + \phi) \approx 1.764$. However, if we choose $\alpha = \sqrt{3}$, then EDF_α is $\sqrt{3} \approx 1.732$-competitive in the 4-bounded case [CCF+06]. For general s-bounded instances, the competitive ratio of EDF_α is at most $2 - \frac{2}{s} + o(\frac{1}{s})$ [CCF+06] and converges to 2 as s tends to infinity.

In [BCJ+16] we modified EDF_ϕ to a ϕ-competitive algorithm ToggleH for 4-bounded instances. The algorithm behaves like EDF_ϕ, except that if the instance locally looks similar to the example above, then ToggleH chooses the earliest-deadline packet whose weight is at least w_h/ϕ^2 (in the example, ToggleH sends k instead of h in step 2). The algorithm maintains one mark that can be assigned to a pending packet, i.e., it uses memory. However, this approach does not seem promising if packets have arbitrary spans.

Agreeable deadlines (a.k.a. similarly ordered packets). If the instance satisfies the agreeable deadlines property, i.e., that $r_p > r_q$ implies $d_p \geq d_q$, then there is a ϕ-competitive policy by Li, Sethuraman, and Stein [LSS05, JLSS12]. In each step, their algorithm ModifiedGreedy (MG) computes the plan and in particular, it identifies the earliest-deadline packet e in the plan and the heaviest packet h. A simplification of MG by Jeż [Jeż10] schedules e if $w_e \geq w_h/\phi$; otherwise, it transmits h. While most analyses of deterministic algorithms mentioned above are based on charging arguments or potential functions, in the analysis of ModifiedGreedy Li *et al.* [LSS05] used a neat trick. They maintain the invariant that the buffer of the adversary has the same content as that of the algorithm, which is done by sometimes allowing the adversary to transmit more packets at once or one packet twice. This yields a surprisingly simple analysis (the paper has just two pages!).

Algs. / Instances	Deterministic		Randomized	
	Upper bounds	Lower bounds	Upper bounds	Lower bounds
General	$2\sqrt{2}-1 \approx 1.828$ [EW12] $\approx 1.893^{\ddagger}$ [EW12] $\phi \approx 1.618$ [Sec. 2.6]	ϕ^*	$\frac{e}{e-1} \approx 1.582^{\ddagger\P}$ [CCF$^+$06, BCJ11] [Jeż13]	$\frac{5}{4}^*$ $\frac{4}{3}^{*\P}$
Agreeable deadlines	$\phi^{\ddagger}$ [LSS05], [JLSS12]	ϕ^*	$\frac{4}{3}^{\ddagger}$ [JLSS12]	$\frac{5}{4}^*$ $\frac{4}{3}^{*\P}$
Increasing weights	ϕ [BCD$^+$13b]	ϕ^*	$\frac{e}{e-1}^{\ddagger\P}$	$\frac{5}{4}^*$ $\frac{4}{3}^{*\P}$
2-bounded	$\phi^{\ddagger}$ [KLM$^+$04]	ϕ [Haj01], [AMZ03, CF03]	$\frac{5}{4}^{\ddagger}$ [CCF$^+$06] $\frac{4}{3}^{\ddagger\P}$ [BCJ11]	$\frac{5}{4}$ [CF03] $\frac{4}{3}^{\P}$ [BCJ11]
3-bounded	$\phi^{\ddagger}$ [CCF$^+$06]	ϕ^*	$\frac{27}{19} \approx 1.421^{\ddagger\P}$ [Jeż13]	$\frac{5}{4}^*$ $\frac{4}{3}^{*\P}$
4-bounded	ϕ [BCJ$^+$16]	ϕ^*	$\frac{256}{175} \approx 1.463^{\ddagger\P}$ [Jeż13]	$\frac{5}{4}^*$ $\frac{4}{3}^{*\P}$
s-bounded	$2-\frac{2}{s}+o\left(\frac{1}{s}\right)^{\ddagger}$ [CCF$^+$06]	ϕ^*	$1/\left(1-\left(1-\frac{1}{s}\right)^s\right)^{\ddagger\P}$ [Jeż13]	$\frac{5}{4}^*$ $\frac{4}{3}^{*\P}$
2-uniform	≈ 1.377 [CJST07] $\sqrt{2} \approx 1.414^{\ddagger}$ [AMZ03]	≈ 1.377 [CJST07] $\sqrt{2} \approx 1.414^{\ddagger}$ [CCF$^+$06]	$\frac{5}{4}^{*\ddagger}$ $\frac{4}{3}^{*\ddagger\P}$	$4-\sqrt{8} \approx 1.172$ [CCF$^+$06] $\frac{6}{5}^{\P}$ [BCJ11] $\frac{4}{3}^{\ddagger\P}$ [BCJ11]
s-uniform	$\phi^{\natural\ddagger}$		$\frac{4}{3}^{\natural\ddagger}$	$\frac{5}{4}$ for $s \to \infty$ [CCF$^+$06]

* follows from the corresponding result for 2-bounded instances.
$^\natural$ follows from the corresponding result for instances with agreeable deadlines.
‡ applies to memoryless scale-invariant algorithms.
¶ against the adaptive adversary.

Table 2.1: A summary of results for Bounded-Delay Packet Scheduling on various types of instances.

s-Uniform instances. A special subclass of instances with agreeable deadlines and also of s-bounded instances is formed by s-uniform instances in which the span of any packet has length of s. Notice that the lower bound of ϕ does not apply, as it uses packets of spans both 1 and 2. For the 2-uniform case, Chrobak *et al.* [CJST07] show a lower bound of ≈ 1.377, which is the largest root of $x^3 + x^2 - 4x + 1 = 0$, together with a matching memory-based algorithm. Regarding memoryless algorithms, Andelman *et al.* [AMZ03] gave a $\sqrt{2} \approx 1.414$-competitive algorithm for 2-uniform instances and the matching lower bound was proven by Chin *et al.* [CCF$^+$06].

Not much was done for the s-uniform case for $s > 2$ and so far the best ϕ-competitive algorithm follows from the result on agreeable instances [LSS05, JLSS12]. Note also that the lower bounds for the 2-uniform case do not imply lower bounds for larger s and, up to our best knowledge, there are no deterministic lower bounds for $s > 2$.

2.2.2 Randomized Algorithms

Already Chin and Fung [CF03] showed the lower bound of 1.25 on the performance of any randomized algorithm against the oblivious adversary; the construction is again just 2-bounded. We describe an extension of this lower bound for the semi-online setting with lookahead in Section 2.7.3. The first randomized algorithms appeared a few years later in the work of Chin *et al.* [CCF$^+$06]. In particular, they provide an optimal 1.25-competitive algorithm for the 2-bounded case and Algorithm RMix achieving ratio $\frac{e}{e-1} \approx 1.582$ in the general case; both against the oblivious adversary. Later, the analysis of RMix was refined to work in the adaptive adversary model by Bieńkowski, Chrobak, and Jeż [BCJ11]. (See Section 1.1 for a description of both the oblivious and the adaptive adversary model.) Bieńkowski *et al.* [BCJ11] complemented their improved analysis by a lower bound of $\frac{4}{3} \approx 1.333$ against the adaptive adversary, which holds already on 2-bounded instances.

The upper bound of $\frac{e}{e-1}$ was improved in some special cases. For agreeable instances, the work of Jeż *et al.* [JLSS12] also contains a $\frac{4}{3}$-competitive algorithm against the oblivious adversary.

RMix was modified by Jeż [Jeż13] to a more universal, though also more complicated variant, called ReMix. Jeż showed that on s-bounded instances it achieves ratio $(1-(1-\frac{1}{s})^s)$ against the adaptive adversary; the ratio tends to $\frac{e}{e-1}$ as s goes to infinity. For 2-bounded instances, the ratio equals $\frac{4}{3}$, matching the aforementioned lower bound.

Finally, randomized algorithms were studied in the s-uniform case and in particular, Chin *et al.* [CCF$^+$06] proved a lower bound against the oblivious adversary which approaches 1.25 as s tends to infinity and equals $4-2\sqrt{2} \approx 1.172$ for 2-uniform instances. Precisely, the lower bound is equal to $1 + \dfrac{s-1}{2s-1+2\sqrt{s^2-s}}$. Against the adaptive adversary, Bieńkowski *et al.* [BCJ11] showed a lower bound of 1.2 and an improved lower bound of $\frac{4}{3}$ for memoryless scale-invariant algorithms, both for the 2-uniform case.

2.3 Contributions

2.3.1 Algorithms for General Instances

Our main contribution is a ϕ-competitive deterministic algorithm for general instances, which is optimal by the lower bound in [Haj01, AMZ03, CF03]. We thus resolve its conjectured existence after more than 15 years. The algorithm uses memory to maintain a crucial monotonicity property, namely, under certain conditions, it increases weights of some pending packets and sometimes also decreases their deadlines. We describe it in Section 2.6, where we also give its analysis by a combination of a potential function and modifying the adversary schedule.

The algorithm is based on plans and on their detailed understanding provided in Section 2.5, which may be of independent interest. In particular, we analyze the structure of the plan and how it changes after arrival of a new packet and after an algorithm schedules a packet.

2.3.2 Algorithms with Lookahead

In Section 2.7, we investigate semi-online algorithms which at time t are aware of all packets arriving by time $t+1$. This property is known as *1-lookahead.* To our best knowledge, lookahead was not considered in the context of packet scheduling before, but it appeared in the online algorithms literature for paging [Alb97], scheduling [MST98], and bin packing [Gro95] since the 1990s.

We provide two results about Bounded-Delay Packet Scheduling with 1-lookahead, restricted to 2-bounded instances and deterministic algorithms. First, in Section 2.7.1, we present an online algorithm for this problem with competitive ratio of $\frac{1}{2}(\sqrt{13}-1) \approx 1.303$. Then, in Section 2.7.2, 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. Our argument is an extension of the lower bound proof of ϕ in [Haj01, AMZ03, CF03]. In fact, our lower bound result is more general: Using only 2-bounded instances, for any integer $\ell \geq 0$ we prove a lower bound of $\frac{1}{2(\ell+1)}(1+\sqrt{5+8\ell+4\ell^2})$ for online algorithms with ℓ-lookahead, that is, algorithms that at time t can see all packets arriving by time $t+\ell$. It follows that there is no 1-competitive algorithm with any constant lookahead, even on 2-bounded instances. In a subsequent work, Kobayashi [Kob18] claims to have an optimal deterministic algorithm with 1-lookahead for 2-bounded instances, matching our lower bound of $\frac{1}{4}(1+\sqrt{17}) \approx 1.281$.

Finally, in Section 2.7.3 we also show that the lower bound of 1.25 for randomized algorithms against the oblivious adversary can be extended to the setting with ℓ-lookahead for any ℓ. The main idea is to have packet spans so large that the use of lookahead is negligible. The result holds already on instances with agreeable deadlines.

2.4 Closely Related Models

FIFO model. The other well-studied model with a single buffer and a single output port is the FIFO model, in which the buffer has a limited capacity and packets cannot be reordered inside the buffer, i.e., the buffer is implemented as a queue. This means that packets need to be transmitted in the first-in-first-out (FIFO) order and some packets must be dropped because of the limited capacity, depending on the weight. The packets have no deadlines, but the delay in sending each packet after its arrival is at most the buffer size if it is ever sent. There are two variants: In the *nonpreemptive variant*, a packet can be dropped only upon its arrival, i.e., once it is added to the buffer, it must be transmitted. The *preemptive variant*, on the other hand, allows the algorithm to also drop packets in the buffer upon arrival of a new packet. Observe that in the offline setting, both variants are the same as the algorithm may simply accept only packets that are eventually transmitted and hence, algorithms for the preemptive case can only be better.

In the nonpreemptive case, the optimal competitive ratio, even for randomized algorithms, equals $\Theta(\log \alpha)$, where α is the ratio of the largest to the smallest packet weight [AMZ03, AM03, Zhu04]. The preemptive case still remains unsolved with the currently best upper bound of $\sqrt{3} \approx 1.732$ [EW09] and the lower bound of ≈ 1.419 [KMvS05] (both for deterministic algorithms and any buffer size).

It is interesting to note that the FIFO model with buffer size s is related to the s-uniform case of Bounded-Delay Packet Scheduling and in particular, any upper bound for the FIFO model carries over to s-uniform instances. Indeed, on s-uniform input we simulate an algorithm for the (preemptive variant of the) FIFO model with buffer capacity s, in each step sending the same packet, and no deadline will be violated, since any packet stays at most s steps in the buffer.

Collecting items from a dynamic queue. Bieńkowski *et al.* [BCD$^+$13a] introduced a generalization of the bounded-delay model in which the deadline of a packet is revealed to the online algorithm only when the packet already expired, but the algorithm knows the ordering of packets by their deadlines. Thus there is a queue of packets and in each step, several packets from the beginning of the queue may be removed by the adversary. Arriving packets are added to their particular locations in the queue (according to the unknown deadlines) and the algorithm then chooses one of the packets from the queue to be scheduled. This problem can be viewed as a partially non-clairvoyant variant of Bounded-Delay Packet Scheduling and it is called Item Collection. In the special FIFO case, the packets are added to the end of the queue only, which corresponds to agreeable deadlines.

Note that Algorithm Greedy, scheduling always the heaviest packet in the queue, is still 2-competitive, using the same charging argument. Also, the lower bound of ϕ carries over to this model. Using just six packets, Bieńkowski *et al.* [BCD$^+$13a] show an improved lower bound of approximately 1.633 that holds even if all packets are released together and thus only the deadlines remain unknown. Moreover, no deterministic memoryless algorithm can beat the performance of Greedy, in contrast with the 1.893-competitive memoryless algorithm for known deadlines.

On the positive side, they provide a roughly 1.897-competitive policy PRUDENT-MARK and a 1.737-competitive algorithm for the FIFO case. Of course, most algorithms for the model with known deadlines do not work, with an exception of the randomized algorithm RMIX. Bieńkowski *et al.* [BCD$^+$13a] claim that RMIX [CCF$^+$06, BCJ11] applies to this model with the same analysis giving ratio $\frac{e}{e-1} \approx 1.582$ and show a matching lower bound for memoryless algorithms against the adaptive adversary. Also, the randomized algorithm of Jeż [Jeż13] for s-bounded instances still works for collecting items from a dynamic queue.

In [BCD$^+$13b] Bieńkowski *et al.* consider instances with increasing weights, i.e., a packet further in the queue can only be larger. All lower bounds so far have this property. They design an optimal ϕ-competitive deterministic algorithm for this special case, which matches the well-known lower bound.

Higher bandwidth. Already the work of Kesselman *et al.* [KLM$^+$04] considers a generalization of Bounded-Delay Packet Scheduling in which the bandwidth of the output port is m, thus m packets can be transmitted in each step. In the online scheduling jargon, there are m identical machines instead of one. In particular, the 2-competitiveness of Greedy and the ϕ-competitiveness of EDF_ϕ on 2-bounded instances in [KLM$^+$04] are proven for any m. The paper also contains a lower bound of $4 - 2\sqrt{2} \approx 1.172$ for the 2-bounded case and a lower bound of $\frac{10}{9}$ for the 2-uniform case, both for any m.

Later, Chin *et al.* [CCF$^+$06] gave an algorithm with ratio $(1 - (\frac{m}{m+1})^m)^{-1}$, which tends to $\frac{e}{e-1} \approx 1.582$ for $m \to \infty$. Note that the lower bound of 1.25 for randomized algorithms against the oblivious adversary and their lower bounds for the s-uniform case hold for any m.

Resource augmentation: Increasing the bandwidth. Jeżabek [Jeż09] considered increasing the bandwidth (or speed) of the algorithm, i.e., the number of packets sent in each step, while still comparing it to an offline optimum with bandwidth 1 (unlike in the previous model). He designed an algorithm with bandwidth m that is $1 + 1/(2^m - 1)$-competitive and proved that no 1-competitive scheduling policy is possible even with arbitrarily large bandwidth.

For the case of agreeable deadlines, Jeż *et al.* [JLSS12] designed a 1-competitive algorithm with bandwidth 2. As far as we are aware, there is no other work which considers resource augmentation for Bounded-Delay Packet Scheduling.

Limited buffer size. If we restrict buffer capacity to some limit B, then, somewhat surprisingly, no deterministic *plan-based* (or best-effort) algorithm can achieve a better ratio than $2 - \frac{1}{B}$ [Li09], where plan-based means that the algorithm always chooses a packet to transmit from the plan. In contrast, if the buffer has infinite capacity, any algorithm can be replaced by a plan-based algorithm with the same or better performance on any instance. Shortly afterwards, Fung [Fun10] devised a 2-competitive deterministic algorithm, based on a careful adjustment of the plan.

Note also that some algorithms carry over from the unlimited buffer setting, for example, the ϕ-competitive strategy for agreeable deadlines [JLSS12] still works with a limited buffer.

Weights decreasing over time. We propose another possible direction in which the bounded-delay model can be generalized. Here, the weight of a packet decreases over time, or more precisely, the weight of a packet p is a non-increasing function w_p of the delay, i.e., the weight of p at time $t \geq r_p$ is $w_p(t - r_p)$. In general, the weight function can be different for different packets. Notice that the deadline case corresponds to functions that are equal to w_p up to $d_p - r_p$ and then drop to 0, thus no explicit deadlines are needed in this more general model. In the non-clairvoyant setting, only the current weights of the pending packets are known to the online algorithm and not the whole functions. Up to our best knowledge, this model is not yet considered in the literature.

Observe that Algorithm Greedy, which schedules the currently heaviest packet, is 2-competitive even in the non-clairvoyant setting, still by the charging argument given above. On the other hand, it is easy to get a lower bound of 2 for deterministic algorithms without clairvoyance: In step 1 the adversary issues two packets a and b, both with weight 1 in step 1, and w.l.o.g. the algorithm transmits a. Then the weight of b drops to 0, while the weight of a stays 1, so the gain of 2 is possible, but the algorithm's gain is only 1.

The question is whether it is possible to design a better than 2-competitive algorithm with the help of randomization and/or clairvoyance.

We remark that this generalization is in the spirit of recent work in online algorithms. For example, Azar *et al.* [AGGP17] recently generalized the k-server problem by allowing the algorithm to delay serving a request, but for a certain cost, which is determined by a delay penalty function of the request. Also, for the FIFO model, Fiat *et al.* [FMN08] considered latency sensitive packets that loose one unit of weight in every step they spend in the buffer; their results were later extended by Feldman and Naor [FN17].

2.5 Plans

In this section we thoroughly study the plan, which we view as an important tool for designing competitive algorithms. Indeed, in the next section we utilize the detailed understanding of the plan in the description and analysis of an optimal deterministic algorithm for general instances.

Definition 2.1. *A subset X of packets pending for an online algorithm at time t is* feasible *if all packets in X can be scheduled in slots $t, t+1, \ldots,$ respecting their deadlines.*

The plan *is the maximum-weight feasible subset of pending packets.*

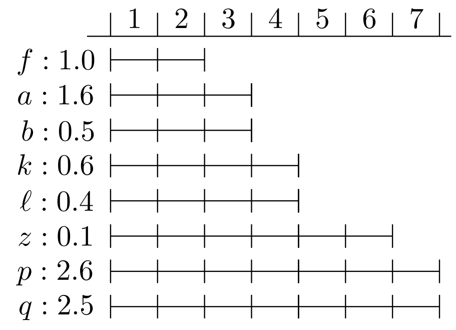
We denote plans by upper-case letters, typically P or Q. Note that there might be more than one plan if there are two packets with the same weight. However, under assumption (A2), which ensures different weights, we shall prove that the plan is unique. An important remark is that the plan is *not* a schedule, i.e., the precise slots for packets are not fixed.

Definition 2.2. *A* realization *of plan P is a schedule of packets in P using slots starting from t only.*

By the definition of the plan, at least one realization exists, but there are usually more. For a realization R of the plan, we say that packets p and q in the plan are *swappable* if their positions in R can be swapped without violating their deadlines. We define the following three realizations; see Figure 2.3 for an example.

- In the *canonical realization*, packets are scheduled in the canonical order, i.e., in the order by deadlines, breaking ties in favor of heavier packets.
- The *front-adjusted realization* ensures that if packets p and q in the plan are swappable and p is before q, then $w_p > w_q$. Note that if j is in slot τ, then all packets in slots $\tau + 1, \ldots, d_j$ are lighter than j.
- The *rear-adjusted realization* is opposite to the front-adjusted one. That is, for any two swappable packets p and q in the plan such that p is before q, $w_p < w_q$ holds. Thus if j is in slot τ, then all packets in slots $\tau + 1, \ldots, d_j$ are heavier than j.

For brevity, the canonical plan refers to the canonical realization of the plan and similarly for the front-adjusted plan and the rear-adjusted plan. The canonical plan is sometimes called the optimal provisional schedule in the literature. We remark that all these three realizations are unique. This is easy to see for the canonical plan and we give an argument for the rear-adjusted plan in Section 2.5.1 and for the front-adjusted plan in Section 2.5.2.



Canonical plan:	f	a	b	k	z	p	q
Front-adjusted plan:	a	f	b	k	p	z	q
Rear-adjusted plan:	b	f	a	k	z	q	p

Figure 2.3: An example of the three realizations of the plan. Packet ℓ is not in the plan, even though it is heavier than z.

One of the most important notions regarding the plan is the following.

Definition 2.3. *Let X be a set of packets pending at time t. For each $\tau \geq t$, we define $\mathsf{pslack}(X, \tau)$ (the* packet slack*) to be $\tau - t + 1 - |X_{\leq\tau}|$, where $X_{\leq\tau}$ is the number of packets in X with deadline at most τ.*

Observe that $\tau - t + 1$ is the number of slots up to τ at time t, thus there are at most $\tau - t + 1$ packets with deadline at most τ in any feasible subset X of pending packets, in particular, in plan P. It follows that if X is feasible, then $\mathsf{pslack}(X, \tau) \geq 0$ for any $\tau \geq t$.

Vice versa, if $\mathsf{pslack}(X, \tau) \geq 0$ for any $\tau \geq t$, then X is feasible. Indeed, we assign packets to slots $t, t+1, \dots$ in the canonical ordering and $\mathsf{pslack}(X, \tau) \geq 0$ for any $\tau \geq t$ ensures that no packet is assigned to a slot after its deadline. This gives us the following observation.

Lemma 2.4. *A subset X of packets pending at time t is feasible if and only if for any $\tau \geq t$ it holds that $\mathsf{pslack}(X, \tau) \geq 0$.*

We remark that if nothing arrives in future, then the best thing to do is to schedule all packets in the plan. However, new packets typically arrive, perhaps with larger weights, and then it is not clear which packet is the best one to be sent.

2.5.1 Computing the Plan

We first observe that feasible subsets of pending packets form a matroid.

Lemma 2.5. *The collection of feasible subsets of packets pending at time t forms a matroid.*

Proof. Clearly, the empty set is feasible and a subset of a feasible set is feasible. For the exchange property, let X and Y be feasible subsets with $|X| > |Y|$. We show that there is a packet $p \in X \setminus Y$ such that $Y \cup \{p\}$ is feasible. Let p be the latest-deadline packet in $X \setminus Y$ and let $Z := Y \cup \{p\}$. We claim that $\mathsf{pslack}(Z, \tau) \geq 0$ for any $\tau \geq t$, which implies that Z is feasible by Lemma 2.4. Note that for $\tau < d_p$, we have $\mathsf{pslack}(Z, \tau) = \mathsf{pslack}(Y, \tau) \geq 0$, where the inequality is by the feasibility of Y and Lemma 2.4.

Next, consider $\tau \geq d_p$, for which $\mathsf{pslack}(Z, \tau) = \mathsf{pslack}(Y, \tau) - 1$ holds. We show that $\mathsf{pslack}(Y, \tau) > 0$, which implies $\mathsf{pslack}(Z, \tau) \geq 0$. Since p is the latest-deadline packet in $X \setminus Y$, we have that any packet in X with deadline after d_p is in Y. Together with $|X| > |Y|$, this implies that $|X_{\leq d_p}| > |Y_{\leq d_p}|$. As $\mathsf{pslack}(X, \tau) \geq 0$, it follows that $\mathsf{pslack}(Y, \tau) > 0$. Hence, the exchange property holds. □

The lemma implies that the maximum-weight feasible subset P of pending packets, i.e., the plan, can be found by the greedy algorithm which we now describe. The algorithm also serves as a tool in our proofs below.

Initially, let P be an empty set. Given a set of pending packets at time t, we first order them by weights and process them in this order, starting from the heaviest. Let j be the currently considered packet (i.e., all heavier packets were already added to P, or rejected). We add j to P if and only if $\mathsf{pslack}(P \cup \{j\}, \tau) \geq 0$ for any $\tau \geq d_j$. After all packets are processed, P is the plan. Since by assumption (A2) no two packets have the same weight, the ordering by weights is linear and thus the plan is unique.

The procedure above, although quite efficient, need not be run every time the algorithm wants to look at the plan. Indeed, the algorithm may start with an empty plan at the beginning and update it every time a packet arrives or is scheduled. We describe these updates in detail in Section 2.5.3. Note that the matroid property implies that at most one other packet in the plan changes.

Notice that we can easily modify the procedure to yield a rear-adjusted plan by adding the considered packet always to the rightmost free slot before its deadline if there is such a slot. (Observe that $\mathsf{pslack}(P, \tau) = 0$ implies that all slots till τ are occupied.) It follows that the rear-adjusted plan is unique and that the procedure can be implemented in $\mathcal{O}(n \log n)$ using (a lazy implementation of) an interval tree for keeping the number of free slots.

We can easily get the canonical plan by sorting the accepted packets according to the canonical order. The front-adjusted plan can be obtained by starting with any realization and performing appropriate swaps of swappable packets, but there is a more direct way which we outline when we analyze the structure of the plan.

2.5.2 Structure of the Plan

In order to be able to use the plan for designing a good algorithm, we need to understand its properties and structure. Consider plan P at time t. For simplicity we assume that P is "fully packed", i.e., a realization of P has a packet in each slot, which is without loss of generality by assumption (A1) (we add sufficiently many virtual 0-weight packets).

Since the plan is fully packed, $\mathsf{pslack}(P, \tau)$ equals the number of packets with deadline more than τ that need to be scheduled in slots up to τ in any realization. Slots τ with $\mathsf{pslack}(P, \tau) = 0$ play a special role as no packet with deadline more than τ can be in a slot up to τ in any realization of the plan, and these "tight" slots divide the plan into segments.

Definition 2.6. *A slot $\tau \geq t$ is called* tight *w.r.t. plan P (at time t) if it holds that $\mathsf{pslack}(P, \tau) = 0$. For convenience, $t - 1$ is a tight slot as well.*

Let $\tau_0 = t - 1 < \tau_1 < \tau_2 < \cdots$ be tight slots. For any integer $i \geq 1$, the interval of slots $[\tau_{i-1} + 1, \tau_i]$ is called a segment.

We denote segments of P by $S_1, S_2, \ldots$, ordered naturally by time. Thus S_1 is the first segment*, while segments $S_2, S_3, \ldots$ are called* later segments.

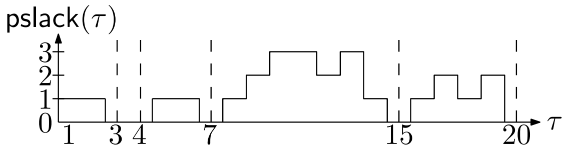


Figure 2.4: An example of a graph of $\mathsf{pslack}(P, \tau)$. The tight slots 3, 4, 7, 15, and 20 are depicted by a vertical dashed line. Note that pslack increases by at most one in each slot and that $\mathsf{pslack}(P, \tau)$ for $\tau \leq 7$ follows the set of packets from Figure 2.3.

In Figure 2.3, the tight slots are 3, 4, and 7, thus segments are $[1, 3]$, $[4, 4]$, and $[5, 7]$. Notice that the tight slots and segments do not depend on a particular choice of a realization of the plan. We remark that the first segment S_1 behaves differently than other segments when a packet is scheduled; we describe this in Section 2.5.3.

We already observed that if τ is a tight slot, no packet with deadline more than τ can be in a slot up to τ in any realization of the plan. This leads to the following observation.

Lemma 2.7. *For each segment S_i of plan P, the packets scheduled in slots in S_i are the same for any realization of P and these are precisely the packets in P with deadlines in S_i.*

Proof. The proof is an easy induction argument. The lemma is trivial for the first segment. For a later segment $S_i = (\tau_{i-1}, \tau_i]$, by induction the set of packets scheduled in segments prior to S_i is determined uniquely and contains all packets with deadlines up to τ_{i-1}. As no packet with deadline more than τ_i can be in a slot up to the tight slot τ_i, the lemma holds for S_i as well. □

Abusing notation, for a segment S_i of plan P, S_i also represents the set of packets in P with deadlines in the interval S_i. The previous lemma thus shows that any realization schedules exactly packets from S_i in each segment S_i. Inside the segment, however, there is some flexibility. In particular, we may schedule any packet from S_i in the first slot of S_i as the next observation shows.

Lemma 2.8. *For each segment $S_i = (\tau_{i-1}, \tau_i]$ of plan P, for each packet $j \in P$ with $d_j \in S_i$, and for each slot $\tau \in (\tau_{i-1}, d_j]$, there is a realization of P which has j in slot τ.*

Proof. Consider adding j to slot τ and scheduling other packets in S_i in the remaining slots according to the canonical ordering. We show that this yields a valid schedule of the segment, which in turn proves the lemma. Suppose for a contradiction that there is a packet $k \in S_i$ scheduled in slot $\tau_k > d_k$. Note that $d_k < \tau_i$ as $\tau_k \leq \tau_i$, since all packets in S_i fit into slots in S_i. Let $k_1, k_2, \dots, k_\ell = k$ be packets in S_i that are not after k in the canonical ordering. It follows that their number ℓ is at least $\tau_k - \tau_{i-1} - 1 \geq d_k - \tau_{i-1}$, (the -1 in the first bound is due to moving j to τ if $\tau < \tau_k$ and $k \prec j$). Counting also packets in previous segments, there are (at least) $d_k - t + 1$ packets in the plan with deadline at most d_k. This implies that $\mathsf{pslack}(P, d_k) \leq 0$, thus $d_k < \tau_i$ is a tight slot in segment S_i, which is a contradiction, since τ_i is the only tight slot in S_i. □

Remark: Another characterization of a tight slot is that τ is tight if and only if a packet with deadline τ is in slot τ in the canonical plan.

Computing a realization by segments. We briefly describe a direct procedure for computing the front-adjusted realization of the plan (computing the rear-adjusted plan or the canonical plan can also be done similarly). Given a set of packets in plan P and the values of pslack, we first partition the packets in P to segments according to their deadlines. The segments of plan P are called *level-0 segments* and we further divide them into "next-level" segments by "next-level" tight slots.

As segments do not interfere by Lemma 2.7, we do the following for each segment $S_i = (\tau_{i-1}, \tau_i]$ separately: For any $\tau \in S_i$, let $\sigma(\tau)$ be a variable for storing a slack, initially set to $\mathsf{pslack}(P, \tau)$. Choose the heaviest packet j in S_i and schedule it in the first slot of S_i, which is possible by Lemma 2.8. Next, decrease $\sigma(\tau)$ for all $\tau \in (\tau_{i-1}, d_j)$. Now, a *level-1* tight slot is a slot τ in S_i that satisfies $\sigma(\tau) = 0$ (this includes the last slot τ_i in S_i, for which $\sigma(\tau_i) = \mathsf{pslack}(P, \tau_i) = 0$). These new tight slots divide S_i without the first slot into level-1 segments with determined packets (similarly to Lemma 2.7). We recurse on each of them and continue in the same way.

We claim that the above procedure computes the front-adjusted plan. Consider packets p and q in the plan such that $w_p < w_q$, p is assigned to slot τ_p and q to slot τ_q with $\tau_p < \tau_q$. We show that $d_p < \tau_q$, meaning that these packets are not swappable. This clearly holds if p and q are in different segments of the plan. Otherwise, let S_i be their segment of P. Consider the segment S' with the highest level that contains both p and q (possibly, $S' = S_i$). Recall that the procedure chooses the heaviest packet h' in S' and puts it in the first slot of S', which shatters S' into several new segments. Note that $h' \neq p$ as $w_p < w_q$ and $h' \neq q$ as $\tau_p < \tau_q$. Thus p and q get into different segments on the next level and then $\tau_p < \tau_q$ implies $d_p < \tau_q$, since each packet in a new segment has its deadline in the new segment. This shows the claim. Finally, it follows that the front-adjusted plan is unique.

Previous and next tight slots and minimum weights. We use the following two notions.

Definition 2.9. *For a slot $\tau \geq t$ of plan P at time t, let $\mathsf{nextts}(P, \tau)$ (the* next tight slot*) be the earliest tight slot τ' in P with $\tau' \geq \tau$; as we assume that the plan is fully packed, it exists for any $\tau \geq t$.*

Similarly, let $\mathsf{prevts}(P, \tau)$ (the previous tight slot*) be the latest tight slot τ' in P with $\tau' < \tau$; as $t - 1$ is assumed to be a tight slot, it is defined for any $\tau \geq t$.*

The crucial notion is the minimum weight in the plan that can be in a slot up to τ in a realization of the plan. Since we may put the lightest packet in a segment S_i to the first slot of S_i by Lemma 2.8, we look at the weights of packets up to the next tight slot.

Definition 2.10. *For a slot $\tau \geq t$ of plan P at time t, let $\mathsf{minwt}(P, \tau)$ be the minimum weight of a packet ℓ in plan P with $d_\ell \leq \mathsf{nextts}(P, \tau)$.*

We observe that minwt upper bounds the weight of a packet not in the plan.

Lemma 2.11. *For any packet $a \notin P$ it holds that $w_a < \mathsf{minwt}(P, d_a)$.*

Proof. Let ℓ be the minimum-weight packet in P with deadline till $\mathsf{nextts}(P, d_a)$. By Lemma 2.8, there is a realization of plan P which has ℓ in the first slot τ of the segment containing d_ℓ. Note that $\tau \leq d_a$ as $d_\ell \leq \mathsf{nextts}(P, d_a)$. The optimality of the plan and assumption (A2) then imply $w_a < w_\ell$. ☐

The lemma also yields the following alternative (dual) definition: $\mathsf{minwt}(P, \tau)$ is the supremum value of λ such that adding a packet v with $d_v = \tau$ and $w_v = \lambda$ does not change the plan.

Next, for a fixed plan P, we observe that $\mathsf{minwt}(P, \tau)$ is monotone in τ and all slots in a segment have the same value of $\mathsf{minwt}(P, \tau)$.

Lemma 2.12. *For plan P at time t it holds that $\mathsf{minwt}(P, \tau)$ is monotone non-increasing in τ, i.e.,*

$$\mathsf{minwt}(P, t) \geq \mathsf{minwt}(P, t + 1) \geq \mathsf{minwt}(P, t + 2) \geq \dots$$

Moreover, for any segment S_i, the value of $\mathsf{minwt}(P, \tau)$ is the same for any $\tau \in S_i$.

Proof. The set of packets considered in the definition of $\mathsf{minwt}(P, \tau)$ gets only larger if we increase τ, proving the monotonicity property. The "moreover" part easily holds by definition as well. ☐

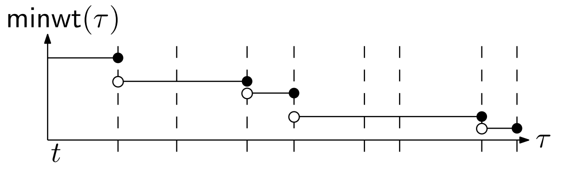


Figure 2.5: An example of a graph of $\mathsf{minwt}(P, \tau)$. The tight slots are depicted by a vertical dashed line.

In the next section, we prove an important monotonicity property of $\mathsf{minwt}(P, \tau)$, in particular, that for a fixed τ, the value of $\mathsf{minwt}(P, \tau)$ does not decrease when a new packet arrives, or when a packet from the first segment of the plan is scheduled. However, if an algorithm schedules a packet from a later segment, then the value of minwt decreases for some slots.

2.5.3 Plan Updates

In this section, we analyze how the plan evolves over time and in particular, how it changes after arrival of a new packet and after an algorithm schedules a packet. First, we define a useful notion that is relevant to plan updates, especially after a packet is scheduled.

Substitute packets. We now take into consideration packets that are not in the plan but will be added if some packet is scheduled. Note that when an algorithm schedules a packet p in a later segment of the plan (i.e., $p \notin S_1$), then intuitively the segment has one free slot. The free slot cannot be filled by a packet with a deadline in a previous segment, thus a new packet ϱ with $d_\varrho > \mathsf{prevts}(P, d_p)$ gets into the plan; we call ϱ the *substitute packet* for p.

Furthermore, scheduling p from a later segment intuitively means that we move it into the first segment S_1 (actually, to the first slot of P). This implies that there are too many packets in S_1 and thus the lightest packet ω in S_1 is kicked out of the plan.

This intuition serves as a motivation for the following definition.

Definition 2.13. *Let P be the plan at time t. For each $j \in P$ we define the* substitute packet *of j, denoted $\mathsf{sub}(P, j)$, as follows. If $j \in S_1$, then $\mathsf{sub}(P, j) = \omega$, where ω is the lightest packet in S_1. If $j \notin S_1$, then $\mathsf{sub}(P, j)$ is the heaviest pending packet $\varrho \notin P$ that satisfies $d_\varrho > \mathsf{prevts}(P, d_j)$ (it exists by assumption (A1)).*

Note that, by definition, all packets in a segment of P have the same substitute packet. Observe that for any $j \in P$ we have $w_j \geq w(\mathsf{sub}(P, j))$. Indeed, if $j \in S_1$, then $\mathsf{sub}(P, j) = \omega$ and clearly $w_j \geq w_\omega$, and otherwise, we have $d(\mathsf{sub}(P, j)) > \mathsf{prevts}(P, d_j)$, thus set $P - \{j\} \cup \{\mathsf{sub}(P, j)\}$ is feasible and the optimality of P implies that $w_j \geq w(\mathsf{sub}(P, j))$. (A nearly the same notion of *suppressed packets* was used by Englert and Westermann [EW12], however, for $j \in S_1$ they define the suppressed packet as a dummy 0-weight packet.)

Plan updates after a packet arrival. As we have demonstrated above, the most important characteristics of the plan are the tight slots and segments. In order to analyze their changes, we look at how the values of pslack change and, as they are used to define tight slots, we also get how the tight slots are updated. We start by describing the plan update after a packet arrival formally. See Figure 2.6 for an illustration.

Lemma 2.14. *(The Plan-Update Lemma for Arrival.) Suppose that a new packet k is released at time t. Denote by P the plan just before k arrives and by Q the plan after k is released. Let $\gamma = \mathsf{nextts}(P, d_k)$. Define f to be the minimum-weight packet in P with $d_f \leq \gamma$, i.e., such that $w_f = \mathsf{minwt}(P, \gamma)$. Let $\delta = \mathsf{prevts}(P, d_f)$.*

If $w_k < w_f$, then $Q = P$, i.e., k is not added to the plan.

If $w_k > w_f$, then $Q = P \cup \{k\} - \{f\}$, i.e., k is added to the plan replacing f. In this case the following holds:

(a) $\mathsf{pslack}(P, \tau)$ does not change for slots $\tau < \min(d_f, d_k)$ or $\tau \geq \max(d_f, d_k)$. For other slots, there are two cases:

   (a1) If $d_k > d_f$, then $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau) + 1$ for $\tau \in [d_f, d_k)$. It follows that all segments of P in the interval $(\delta, \gamma]$ get merged into one segment $(\delta, \gamma]$ of Q.

   (a2) If $d_k < d_f$, then $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau) - 1$ for $\tau \in [d_k, d_f)$ and thus there might be new tight slots in $[d_k, d_f)$. In this case, both d_f and d_k must be in the segment $(\delta, \gamma]$ of P.

(b) For any slot τ, $\mathsf{minwt}(Q, \tau) \geq \mathsf{minwt}(P, \tau)$.

We remark that f always exists by assumption (A1) and by assumption (A2) $w_k \neq w_f$ holds.

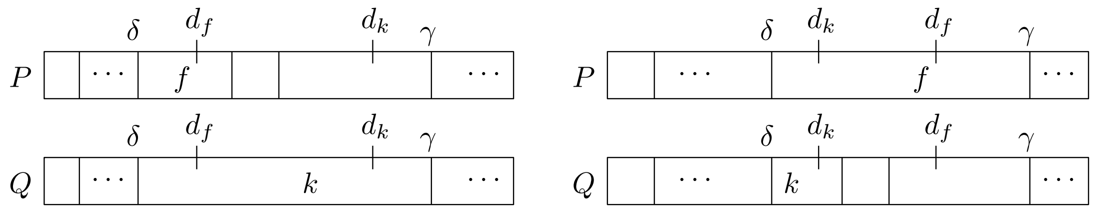


Figure 2.6: An illustration of changes of tight slots and segments in case (a1) of Lemma 2.14 on the left and in case (a2) on the right. Each plan is represented by a rectangle divided into segments by tight slots, which are depicted by vertical line segments.

Proof. We start by proving how the set of packets in the plan changes. Consider the greedy procedure for computing the plan as described in Section 2.5.1, run in parallel for P and Q, i.e., for the set of pending packets without k and for pending packets including k. Note that the order in which packets are considered in both runs is the same, except for packet k which is available only when computing Q.

If $w_k < w_f$, then both runs have the same set of accepted packets after f is added. Moreover, as f is the minimum-weight packet in P with $d_f \leq \gamma = \mathsf{nextts}(P, d_k)$, all packets in P with deadline at most γ are already accepted to both P and Q and thus $\mathsf{pslack}(Q_{\geq f}, \gamma) = \mathsf{pslack}(P_{\geq f}, \gamma) = 0$, where $Q_{\geq f}$ and $P_{\geq f}$ are the sets of packets accepted to Q and P, respectively, after adding f. However, as $d_k \leq \gamma$ and as k is considered after f, packet k is not added to Q, which implies that both runs produce the same plan $P = Q$.

Otherwise, $w_k > w_f$. First, assume that $w_k = w_f + \varepsilon$ for a tiny $\varepsilon > 0$, so that k is just before f in the decreasing order by weights. In the run for computing P (i.e., for pending packets without k), before the procedure adds f it holds that $\mathsf{pslack}(P_{>f}, \tau) > 0$ for any $\tau \geq d_f$, where $P_{>f}$ is the set of packets accepted before f. Moreover, f is the minimum-weight packet in P with $d_f \leq \gamma = \mathsf{nextts}(P, d_k)$ and thus if $d_k < d_f$, we have $\mathsf{pslack}(P, \tau) > 0$ for $\tau \in [d_k, d_f)$. It follows that $\mathsf{pslack}(P_{>f}, \tau) > 0$ for any $\tau \geq \min(d_f, d_k)$ and that $\mathsf{pslack}(P_{>f}, \gamma) = 1$. In the run with k, packet k is considered just before f. Note that $Q_{>k} = P_{>f}$, thus by the inequalities above, k is added to Q. Then $\mathsf{pslack}(Q_{>f}, \gamma) = 0$ and the procedure rejects f. After rejecting f, both runs are again the same, as no packet with deadline at most γ is accepted and the values of pslack after γ are the same. Hence $Q = P \cup \{k\} - \{f\}$ for $w_k = w_f + \varepsilon$.

For arbitrary $w_k > w_f$, observe that if we increase the weight of k from $w_f + \varepsilon$, then the plan does not change as k is already in it. Therefore $Q = P \cup \{k\} - \{f\}$ if $w_k > w_f$.

Next, we analyze the changes in the case $w_k > w_f$.
(a) Suppose that $d_k \geq d_f$. Replacing f by k in the plan does not change any value of $\mathsf{pslack}(P, \tau)$ for slots $\tau < d_f$ or $\tau \geq d_k$. All values of $\mathsf{pslack}(P, \tau)$ for $\tau = d_f, d_f + 1, ..., d_k - 1$ increase by 1 as $|P_{\leq \tau}|$ decreases by one for such τ. So δ and γ remain tight slots and there are no tight slots in the interval $[d_f, d_k)$ in Q. This shows (a1).

The argument for (a2) is similar. Suppose that $d_k < d_f$. The value of $\mathsf{pslack}(P, \tau)$ for $\tau < d_k$ or $\tau \geq d_f$ stays the same. All values of $\mathsf{pslack}(P, \tau)$ for $\tau = d_k, d_k + 1, ..., d_f - 1$ are positive and decrease by 1 (possibly producing new tight slots). Thus δ and γ remain tight slots.

(b) For $\tau \leq \delta$, the set of packets with deadline at most δ does not change and tight slots up to δ remain the same. Hence, $\mathsf{minwt}(Q, \tau) = \mathsf{minwt}(P, \tau)$ for such τ. For $\tau \in (\delta, \gamma]$ it holds that $\mathsf{minwt}(P,\tau) = w_f$ and, as k replaces f and γ remains tight slot, we actually have $\mathsf{minwt}(Q,\tau) > \mathsf{minwt}(P,\tau)$.

Finally, consider $\tau > \gamma$. Since tight slots after γ are unchanged, the set of packets considered in the definition of $\mathsf{minwt}(\tau)$ changes only by replacing f by k, which implies $\mathsf{minwt}(Q,\tau) \geq \mathsf{minwt}(P,\tau)$. This concludes the proof. □

We remark that for any packet $j \notin \{f,k\}$ it holds that $w(\mathsf{sub}(Q,j)) \geq w(\mathsf{sub}(P,j))$. Moreover, in the case $w_k > w_f$, we have $w(\mathsf{sub}(Q,k)) \geq w_f$.

Plan updates after scheduling a packet. Next, we analyze how the plan evolves when a packet is scheduled and the time is increased, but before new packets in the next step arrive. There are two cases, depending on whether the algorithm schedules a packet from the first segment S_1 of P or from a later segment. For clarity, we state a lemma for each case. (As our focus is on algorithms that always transmit a packet which is in the plan, we do not provide a lemma for scheduling a packet not in the plan.) In both cases, p is the scheduled packet, P is the current plan at time t (after all arrivals at time t) and by Q we denote the plan just after p is scheduled, the time is incremented, but before new packets at time $t+1$ arrive. We start with the case of the first segment, which is quite straightforward. See Figure 2.7 for an illustration.

Lemma 2.15. *(The Plan-Update Lemma for Scheduling $p \in S_1$.) Suppose that at time t an algorithm schedules a packet $p \in S_1$ and let $\beta = \mathsf{nextts}(P, d_p)$. Then:*

(a) $Q = P \setminus \{p\}$.

(b) $\mathsf{pslack}(Q,\tau) = \mathsf{pslack}(P,\tau)$ for $\tau \geq d_p$ and $\mathsf{pslack}(Q,\tau) = \mathsf{pslack}(P,\tau) - 1$ for $\tau < d_p$. In particular, β is a tight slot in Q and there might be new tight slots in $[t+1, d_p)$ in Q.

(c) For any slot τ, $\mathsf{minwt}(Q,\tau) \geq \mathsf{minwt}(P,\tau)$.

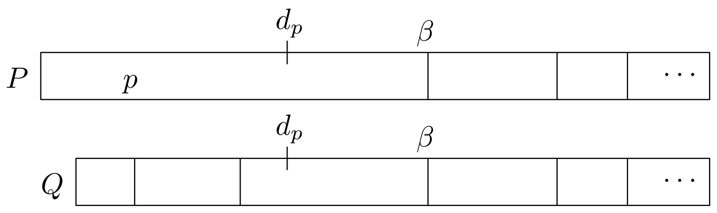


Figure 2.7: An illustration of changes of tight slots and segments in Lemma 2.15.

Proof. (a) This follows from the fact that by Lemma 2.8 there is a realization of P which has p in slot t.

(b) For $\tau \geq d_p$ the value of $\mathsf{pslack}(P,\tau)$ increases by one as the algorithm scheduled p, but decreases by one as the time is increased, thus it does not change. It follows that slot $\beta \geq d_p$ remains to be tight (including the case $\beta = t$). For $\tau < d_p$, $\mathsf{pslack}(Q,\tau) = \mathsf{pslack}(P,\tau) - 1$ as the time is incremented, so there might be new tight slots before d_p.

(c) We have that the value of $\mathsf{nextts}(P,\tau)$ may only decrease by (a). Since also no packet is added to the plan, in the definition of $\mathsf{minwt}(Q,\tau)$ we consider a subset of packets used to define $\mathsf{minwt}(P,\tau)$, which shows (b). □

We remark that when a packet $p \in S_1$ is scheduled, $w(\mathsf{sub}(Q,j)) \geq w(\mathsf{sub}(P,j))$ need not hold for all packets $j \in Q$. Indeed, for $j \in S_1$ such that $j \neq p$, the substitute packet was ω, but if j is no longer in the first segment, then $w(\mathsf{sub}(Q,j)) < w_\omega$. However, $w(\mathsf{sub}(Q,j)) = w(\mathsf{sub}(P,j))$ for j in a later segment.

For the case $p \in P$, but $p \notin S_1$, recall that the substitute packet $\mathsf{sub}(P, p)$ for p is the heaviest pending packet $\varrho \notin P$ satisfying $d_\varrho > \mathsf{prevts}(P, d_p)$. We prove that ϱ really appears in the plan when P is scheduled. See Figure 2.8 for an illustration.

Lemma 2.16. *(The Plan-Update Lemma for Scheduling $p \notin S_1$.) Suppose that at time t an algorithm schedules a packet p which is in a later segment of P. Let $\varrho = \mathsf{sub}(P, p)$ and let ω be the lightest packet in the first segment S_1 of P. Moreover, let $\beta = \mathsf{nextts}(P, t)$ be the first tight slot, let $\delta = \mathsf{prevts}(P, d_p)$ and $\gamma = \mathsf{nextts}(P, d_\varrho)$. Then:*

(a) $Q = P \setminus \{p, \omega\} \cup \{\varrho\}$. That is, ϱ is the unique packet in $Q \setminus P$, and ω is not in Q.

(b) $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau) - 1$ for $\tau \in [t + 1, d_\omega)$ and $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau)$ for $\tau \in [d_\omega, \beta]$.

(c) It holds that $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau)$ for $\beta < \tau < \min(d_p, d_\varrho)$ and for $\tau \geq \max(d_p, d_\varrho)$. For the rest of slots, there are two cases:

(c1) If $d_\varrho > d_p$, then $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau) + 1$ for $\tau \in [d_p, d_\varrho)$. Thus all segments of P in $(\delta, \gamma]$ get merged into one segment $(\delta, \gamma]$ of Q.

(c2) If $d_\varrho < d_p$, then $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau) - 1$ for $\tau \in [d_\varrho, d_p)$. In this case, there might be new tight slots in $[d_\varrho, d_p)$ in Q.

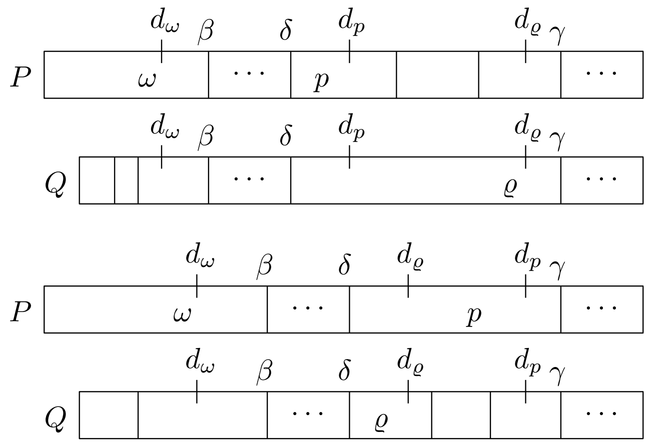


Figure 2.8: An illustration of changes of tight slots and segments in case (c1) of Lemma 2.16 on the top and in case (c2) on the bottom.

Proof. Consider the greedy procedure for computing the plan and run it in parallel for P and for Q. Initially, the runs are the same, i.e., the order of packets is the same and the same packets are added, except for p which is not considered in the run for Q. As ω is accepted to P, we have $\mathsf{pslack}(P_{>\omega}, \tau) > 0$ for any $\tau \geq d_\omega$, where $P_{>\omega}$ is the set of packets accepted to P before ω. This implies $\mathsf{pslack}(Q_{>\omega}, \tau) \geq 0$ for $\tau \geq d_\omega$ as Q is with respect to time $t + 1$. (Note that $p \notin Q_{>\omega}$ and $p \in P_{>\omega}$ as $w_p > w_\omega$, thus $\mathsf{pslack}(Q_{>\omega}, \tau)$ may be equal to $\mathsf{pslack}(P_{>\omega}, \tau)$ for $\tau \geq d_p$.) Moreover, $\mathsf{pslack}(P, \tau) > 0$ for $\tau \in [t, d_\omega)$ as ω is in S_1, thus $\mathsf{pslack}(Q_{>\omega}, \tau) \geq 0$ for any $\tau \geq t+1$. It follows that all packets heavier than ω are accepted to Q. Note also that any packet $a \notin P$ is lighter than ω, thus no such packet is accepted to Q before the procedure considers ω.

Since $\mathsf{pslack}(P_{\geq\omega}, \beta) = 0$ and $d_p > \beta$, we have that $\mathsf{pslack}(Q_{,>\omega}, \beta) = 0$ and thus ω is not accepted to Q. After considering ω in both runs, the values of pslack are the same for slots in $[d_\omega, d_p)$ and $\mathsf{pslack}(P_{\geq\omega}, \beta) = \mathsf{pslack}(Q_{\geq\omega}, \beta) = 0$. Thus in both runs no additional packet with deadline at most β is accepted and the same packets with deadline at most $\delta = \mathsf{prevts}(P, d_p)$ are accepted.

After rejecting ω, the values of pslack after δ can only be higher in the run for Q. It follows that both runs are the same, until considering ϱ, the heaviest pending packet not in P satisfying $d_\varrho > \delta$. We claim that the procedure accepts ϱ to Q. It is sufficient to show that $\mathsf{pslack}(Q_{>\varrho}, \tau) > 0$ for any $\tau > \delta$. This holds, since $\mathsf{pslack}(Q_{>\varrho}, \tau) \geq \mathsf{pslack}(P, \tau) > 0$ for $\tau \in (\delta, d_p)$ and since for $\tau \geq d_p$ we have $\mathsf{pslack}(Q_{>\varrho}, \tau) > \mathsf{pslack}(P, \tau)$, as the procedure rejected ω to Q, $p \notin Q$, but Q has one slot less than P.

Furthermore, as $w_\varrho < \mathsf{minwt}(P, d_\varrho)$, all other packets with deadline not exceeding $\gamma = \mathsf{nextts}(P, d_\varrho)$ are already added to both P and Q (except ω and p), thus no other packet $a \notin P$ is added to Q. Thus after ϱ is accepted to Q, $\mathsf{pslack}(Q_{\geq\varrho}, \gamma) = 0$ and no other packet with deadline up to γ is added to Q. Finally, both runs accept the same set of packets with deadlines after γ as the values of $\mathsf{pslack}(\tau)$ for $\tau > \gamma$ are the same for both after ϱ is added to Q. It follows that $Q = P \setminus \{p, \omega\} \cap \{\varrho\}$. This shows (a).

We now analyze the changes in the values of pslack and show (b) and (c). If $d_\omega > t$, it holds that $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau) - 1$ for $\tau \in [t+1, d_\omega)$, as the time was increased and there is no change in the set of packets taken into account.

For $d_\omega \leq \tau < \min(d_\varrho, d_p)$, we have $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau)$ as the time was increased, but ω was forced out. As $d_\omega \leq \beta < \min(d_\varrho, d_p)$, β is a tight slot in Q (including the case $\beta = t$). Similarly, for $\tau \geq \max(d_\varrho, d_p)$, it holds that $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau)$, since the time was incremented, ω was forced out, p was scheduled, and ϱ appeared in Q.

In case (c1), for $d_p \leq \tau < d_\varrho$ we have $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau) + 1$ as the time was increased, but ω was forced out and p was scheduled, thus Q has no tight slots in $[d_p, d_\varrho)$. It follows that $\mathsf{prevts}(Q, d_\varrho) = \delta$ and $\mathsf{nextts}(Q, d_p) = \gamma$.

In case (c2), for $d_\varrho \leq \tau < d_p$ note that $\mathsf{pslack}(P, \tau) \geq 1$ as $d_p > \tau \geq d_\varrho > \delta = \mathsf{prevts}(P, d_p)$ and $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau) - 1$ as the time was increased, ω was forced out, and ϱ appeared in Q. $\square$

Again, we may have $w(\mathsf{sub}(Q, j)) < w(\mathsf{sub}(P, j))$ for a packet $j \neq p$, in particular, if $\mathsf{sub}(P, j) = \varrho$. Also, $\mathsf{minwt}(d_\varrho)$ decreases, as clearly $\mathsf{minwt}(P, d_\varrho) > w_\varrho$, but $\mathsf{minwt}(Q, d_\varrho) = w_\varrho$.

2.5.4 Plan-Based Algorithms

Equipped with a detailed understanding of the plan and its updates, we turn our focus to algorithms and in particular, to algorithms that deliberately choose a packet to schedule from the plan.

There are two very simple and natural such algorithms: Algorithm Greedy always schedules the heaviest pending packet (which must be in the plan), while GreedyPlan sends the first packet in (a realization of) the plan in each step. Both are 2-competitive [Haj01, KLM$^+$04, CY03]; the proof for GreedyPlan by Chang and Yap [CY03] is for sending the first packet from the rear-adjusted plan.[2] It seems that in order to get a better algorithm, one has to balance between these two algorithms very carefully. In other words, the challenge is to balance the immediate profit (maximized by Greedy) against future profits (maximized by GreedyPlan).

However, the most straightforward way of combining the greedy strategies does not work. Namely, consider the algorithm that sends either the earliest-deadline packet f in the plan if $w_f \geq w_h/\alpha$, or the heaviest packet h otherwise, where $\alpha > 1$ is a parameter. Englert and Westermann [EW12] showed that its competitive ratio is at

[2]We remark that the analysis techniques in Section 2.6 about our ϕ-competitive algorithm can be used to show 2-competitiveness of a general version of GreedyPlan, which transmits an arbitrary packet from the first segment.

least 2 for any $\alpha > 1$. Their proof actually does not use that f is the earliest-deadline packet in the plan, it can also be the heaviest packet in the first segment, since the first segment consists of just one step in the lower bound instances.

At SODA 2007, better than 2-competitive plan-based algorithms appeared in two papers: Li, Sethuraman, and Stein [LSS07] showed that their algorithm DP_ϕ (*Dummy Packets*) with memory is $3/\phi \approx 1.854$-competitive and also at least $(3+\phi)/(1+\phi) \approx 1.764$-competitive (by the example that forces the same ratio for EDF_ϕ on 4-bounded instances). The authors build on their algorithm ModifiedGreedy (MG) [LSS05] for the agreeable deadlines case. To avoid instances on which EDF_α has a ratio close to 2, when DP sends a packet f, which is neither the earliest-deadline packet in the plan, nor h, it generates a dummy (virtual) packet with deadline d_f and weight w_h/ϕ and associates it with h. If it is about to send a dummy packet, instead it schedules its associated (real) packet. Finally, if h has already an associated dummy packet, the algorithm transmits it immediately.

In the second paper, Englert and Westermann [EW12] designed a $2\sqrt{2}-1 \approx 1.828$-competitive strategy using memory and an approximately 1.893-competitive memoryless algorithm. Before describing these two algorithms, let us elaborate on how to design a good plan-based algorithm and discuss which packets make sense to send.

Significant packets. Consider a packet j in a later segment S_i of the plan. Recall that by Lemma 2.16, if an algorithms schedules j, its substitute packet $\varrho = \mathsf{sub}(P, j)$ appears in the plan. Moreover, recall that a realization of the plan can have any packet in the first slot of a segment by Lemma 2.8 and that all packets in a segment have the same substitute packet. This motivates the following definition.

Definition 2.17. *A pending packet j is* significant *if j is in the plan and it is the heaviest packet in its segment of the plan.*

By the observation above, it seems that an algorithm should just choose one of the significant packets to schedule. Pushing it further, among significant packets in later segments that have the same substitute packet, the algorithm perhaps only needs to take into account the heaviest one.

Algorithm Plan. Many algorithms can be described as scheduling a packet with the maximum *value*, where the value v_p of a packet p of course depends on w_p, but it may depend also on the weight of other packets or something else. In particular, if p is in a later segment of the current plan P, then it makes sense that v_p also depends on the weight of its substitute packet $\mathsf{sub}(P, p)$, which gets into the plan when p is scheduled.

Concretely, a natural way how to set up values is as follows: Let P be the plan in step t (after all arrivals at time t) and let α be a parameter. For each $p \in P$ we define its value $v_p = w_p + \alpha \cdot w(\mathsf{sub}(P, p))$. This yields the following simple algorithm, called $\mathsf{Plan}(\alpha)$.

Pseudocode 1 Algorithm: Plan

```
1: schedule the packet $p \in P$ with the maximum value $v_p = w_p + \alpha \cdot w(\mathsf{sub}(P, p))$
```

Note that we just need to calculate the values of significant packets in the plan, since the algorithm always sends a significant packet.

Remark. An equivalent way of defining Algorithm Plan is the following: Schedule the packet $p \in P$ that maximizes the value of $w_p + \alpha' \cdot w(\overline{Q_p})$, where $\alpha' < 1$ is a parameter and $\overline{Q_p}$ is the plan after p is deliberately scheduled and the time is incremented; note that $p \notin \overline{Q_p}$. This alternative formulation is perhaps more intuitive: The term w_p is the immediate gain of the algorithm, while $w(\overline{Q_p})$ represents the optimal future profit if no packet arrives.

We show the equivalence of the two formulations as follows: Let a be a packet in S_1, the first segment of P, and let g be a packet in a later segment of P. Then $\overline{Q_a} = P \setminus \{a\}$ by Lemma 2.15 and $\overline{Q_g} = P \setminus \{g, \omega\} \cup \{\mathsf{sub}(P, g)\}$ by Lemma 2.16.

If packet $p \in S_1$ is scheduled, then $\overline{Q_p} = P \setminus \{p\}$ and by the choice of p we obtain

$$
\begin{aligned}
w_p + \alpha' \cdot w(\overline{Q_p}) &\geq w_g + \alpha' \cdot w(\overline{Q_g}) \\
w_p + \alpha' \cdot w(P \setminus \{p\}) &\geq w_g + \alpha' \cdot w(P \setminus \{g, \omega\} \cup \{\mathsf{sub}(P, g)\} \\
w_p + \alpha' \cdot w(P) - \alpha' \cdot w_p &\geq w_g + \alpha' \cdot w(P) - \alpha' \cdot w_g - \alpha' \cdot w_\omega + \alpha' \cdot w(\mathsf{sub}(P, g)) \\
(1 - \alpha') \cdot w_p + \alpha' \cdot w_\omega &\geq (1 - \alpha') \cdot w_g + \alpha' \cdot w(\mathsf{sub}(P, g))\,, \\
(1 - \alpha') \cdot w_p + \alpha' \cdot w(\mathsf{sub}(P, p)) &\geq (1 - \alpha') \cdot w_g + \alpha' \cdot w(\mathsf{sub}(P, g))\,,
\end{aligned}
$$

which is the inequality implied by Plan, multiplied by $1 - \alpha'$.

If the scheduled packet p is in a later segment of P, then $\overline{Q_p} = P \setminus \{p, \omega\} \cup \{\mathsf{sub}(P, p)\}$ by Lemma 2.16. By similar calculations as above, for any packet $j \in P$ we have

$$
(1 - \alpha') \cdot w_p + \alpha' \cdot w(\mathsf{sub}(P, p))) \geq (1 - \alpha') \cdot w_j + \alpha' \cdot w(\mathsf{sub}(P, j))\,,
$$

It follows that maximizing the value of $w_p + \alpha' \cdot w(\overline{Q_p})$ is equivalent to maximizing $w_p + \alpha \cdot w(\mathsf{sub}(P, p))$ for $\alpha = \alpha'/(1 - \alpha')$.

Algorithms of Englert and Westermann. The memoryless algorithm of Englert and Westermann [EW12], which we denote by EW-Memoryless, is very similar to Plan, but it assigns a smaller value for packets in S_1. The algorithm actually has two parameters, $\alpha \leq 1$ and $\beta \leq 1$.

Pseudocode 2 Algorithm: EW-Memoryless(α, β)

```
1: let $m \in P$ be the packet with the maximum value $v_m$, where
2:        $v_m = \alpha \cdot w_m$ for $m \in S_1$
3:        $v_m = \alpha \cdot w_m + (1 - \alpha) \cdot w(\mathsf{sub}(P, m))$ for $m \notin S_1$
4: let $f$ be the earliest-deadline packet in $P$
5: if $w_f \geq \beta \cdot v_m$ then
6:     schedule $f$
7: else
8:     schedule $m$
```

Their algorithm with memory, which we call EW-Memory, is an extension of the memoryless one by making f sometimes more valuable. The algorithm uses memory to maintain the value of $\delta(\tau)$ for each slot τ, which equals the maximum value of $\mathsf{minwt}(Q, \tau)$ for a plan Q so far. Then they just replace the condition $w_f \geq \beta \cdot v_m$ by $\max(w_f, \delta(t)) \geq v_m$, where t is the current time (the parameter β does not occur in the description, but one may easily add it).

Pseudocode 3 Algorithm: EW-Memory(α)

```
1: for each slot $\tau \geq t$, set $\delta(\tau) \leftarrow \max(\delta(\tau), \mathsf{minwt}(P, \tau))$
2: let $m \in P$ be the packet with the maximum value $v_m$, where $v_m$ is set up in the
   same way as in EW-Memoryless
3: let $f$ be the earliest-deadline packet in $P$
4: if $\max(w_f, \delta(t)) \geq v_m$ then
5:     schedule $f$
6: else
7:     schedule $m$
```

Note that both EW-Memoryless and EW-Memory may schedule a non-significant packet if f is not the heaviest packet in S_1. However, m is always a significant packet.

While these algorithms, especially Plan, seem promising to be ϕ-competitive, we have examples that none of them is actually ϕ-competitive for any setting of parameters.

2.6 ϕ-Competitive Algorithm

2.6.1 Algorithm Description

Intuitions. Recall that Algorithm $\mathsf{Plan}(\alpha)$ schedules packet p that maximizes $w_p + \alpha \cdot w(\mathsf{sub}(P, g))$. By analyzing the 2-bounded case, one can get that the right value of α for ϕ-competitiveness is ϕ.[3] As it turns out, the above strategy for choosing p does not, by itself, guarantee ϕ-competitiveness. The analysis of special cases and an example where this simple idea fails lead to the second idea behind our algorithm. The difficulty is related to how the values of $\mathsf{minwt}(P, \tau)$, for a fixed τ, vary over time. We were able to show ϕ-competitiveness for certain instances where $\mathsf{minwt}(P, \tau)$ monotonely increases with the time increasing. We call this property *slot-monotonicity*. Slot monotonicity does not hold for arbitrary instances. The idea is then to simply *force* it to hold by increasing weights and decreasing deadlines of some packets in the new plan, including increasing the weight of the substitute packet $\mathsf{sub}(P, p)$. (To avoid unfairly benefiting the algorithm from these increased weights, we will need to account for them appropriately in the analysis.) Then the algorithm proceeds using these new weights and deadlines for computing the plan and choosing the packet to schedule.

Notation.

- We use w^t_p and d^t_p for the weight and the deadline, respectively, of packet p in step t before a packet is scheduled. We remark that our algorithm does not change weights and deadlines when a new packet arrives. For simplicity, w_p refers to w^t_p and d_p to d^t_p. By w^0_p we refer to the original weight of packet p.
- P^t is the plan at (the current) time t after all packets j with $r_j = t$ arrive and before a packet is scheduled. By $S_1, S_2, \ldots$ we denote the segments of P^t.
- ω is the lightest packet in P^t in the first segment S_1.
- We use $\mathsf{sub}^t(p)$ to denote $\mathsf{sub}(P^t, p)$ and similarly for $\mathsf{minwt}^t(\tau), \mathsf{nextts}^t(\tau)$, and $\mathsf{prevts}^t(\tau)$.

Pseudocode 4 Algorithm: PlanM

```
1:  schedule packet $p \in P^t$ that maximizes $w^t_p + \phi \cdot w^t(\mathsf{sub}^t(p))$
2:  **if** $p$ is *not* in the first segment $S_1$ of $P^t$ **then**          ▷ "leap step"
3:      $\varrho \leftarrow \mathsf{sub}^t(p)$
4:      $w^{t+1}_\varrho \leftarrow \mathsf{minwt}^t(d^t_\varrho)$          ▷ increase $w_\varrho$
5:      $\gamma \leftarrow \mathsf{nextts}^t(d^t_\varrho)$ and $\tau_0 \leftarrow \mathsf{nextts}^t(d^t_p)$
6:      $i \leftarrow 0$ and $h_0 \leftarrow p$
7:      **while** $\tau_i < \gamma$ **do**
8:          $i \leftarrow i + 1$
9:          $h_i \leftarrow$ the heaviest packet in $P^t$ with $d^t_{h_i} \in (\tau_{i-1}, \gamma]$
10:         $\tau_i \leftarrow \mathsf{nextts}^t(d^t_{h_i})$
11:         $d^{t+1}_{h_i} \leftarrow \tau_{i-1}$ and $w^{t+1}_{h_i} \leftarrow \max(w^t_{h_i}, \mathsf{minwt}^t(\tau_{i-1}))$
12:     $k \leftarrow i$          ▷ final value of $i$
```

[3]Interestingly, there are two linear combinations of w_p and $w(\mathsf{sub}(P, g))$ that give ratio ϕ for the 2-bounded case; the other one is $\phi \cdot w_p + w(\mathsf{sub}(P, g))$, but we were not able to extend this one to the general case.

For a pending packet j, if w_j^{t+1} or d_j^{t+1} are not explicitly set in the algorithm, then $w_j^{t+1} \leftarrow w_j^t$ or $d_j^{t+1} \leftarrow d_j^t$, respectively, i.e., weights and deadline remain the same by default.

If p is in the first segment S_1 of P^t, the step is called a *greedy step*. Otherwise (if $p \notin S_1$), the step is called a *leap step*, and then $\varrho = \mathsf{sub}^t(p)$ is the heaviest pending packet $\varrho \notin P^t$ with $d_\varrho^t > \mathsf{prevts}^t(d_p^t)$. We will further consider two types of leap steps. If p and ϱ are in the same segment (formally, $d_\varrho^t \leq \tau_0$, or equivalently, $k = 0$), then this leap step is called a *simple leap step*. If ϱ is in a later segment than p (that is, $d_\varrho^t > \tau_0$, or $k > 0$) then this leap step is called an *iterated leap step*.

Let p be the packet sent by PlanM in step t. We observe that p is the heaviest packet in its segment of P^t as all packets in this segment have the same substitute packet $\mathsf{sub}^t(p)$. Furthermore, p is not too light, compared to the heaviest pending packet h, namely $w_p \geq w_h/\phi^2$. Indeed, as mentioned earlier, we have $w_p \geq w(\mathsf{sub}^t(p))$. It follows that $\phi^2 w_p = w_p + \phi w_p \geq w_p + \phi \cdot w(\mathsf{sub}^t(p)) \geq w_h + \phi \cdot w(\mathsf{sub}^t(h)) \geq w_h$, where the second inequality follows by the choice of p in line 1.

Note that line 1 of the algorithm is exactly Algorithm $\mathsf{Plan}(\phi)$. Thus an equivalent way of defining line 1 is the following: Schedule the packet $p \in P^t$ that maximizes the value of $\phi \cdot w_p^t + w^t(\overline{Q_p^t})$, where $\overline{Q_p^t}$ is the plan after p is deliberately scheduled and the time is increased; note that $p \notin \overline{Q_p^t}$ and that for $\overline{Q_p^t}$ we do not change weights or deadlines.

Leap Step of Algorithm PlanM and Slot-Monotonicity

Our goal is to maintain the slot-monotonicity property, i.e., to ensure that for any slot τ the value of $\mathsf{minwt}^t(\tau)$ does not decrease. For this reason, we need to increase the weight of the substitute packet ϱ in each leap step (as $w_\varrho^t < \mathsf{minwt}^t(d_\varrho^t)$), which is done in line 4. For the same reason, we also need to adjust the deadlines and weights of the packets h_i, which is done in line 11. The deadlines of h_i's are decreased to make sure that the segments between $\delta = \mathsf{prevts}^t(d_p^t)$ and γ do not merge (as merging could cause decrease of some values of $\mathsf{minwt}^t(\tau)$). These deadline changes can be thought of as a sequence of substitutions, where h_1 replaces p in the segment of P ending at τ_0, h_2 replaces h_1, etc., and finally, ϱ replaces h_k in the segment ending at γ.[4] See Figure 2.9 for an illustration. Then, if the weight of some h_i is too low for its new segment, it is increased to match the earlier minimum of that segment, that is, $\mathsf{minwt}^t(\tau_{i-1})$. (To maintain assumption (A2), we also add an infinitesimal to the new weight of h_i.)

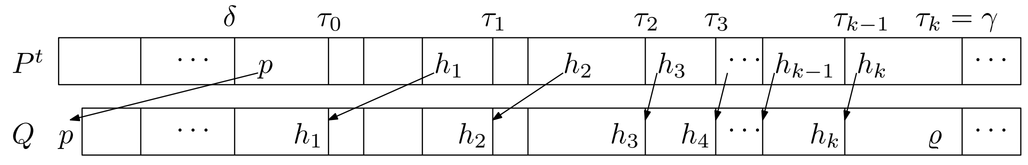


Figure 2.9: An illustration of the shift of the packets $h_1, \ldots, h_k$ in an iterated leap step (lines 6-11 of the algorithm's description). Q is the plan at time $t+1$ just after p is scheduled.

Note that Lemma 2.14 analyzes arrival of a new packet and Lemma 2.15 analyzes scheduling a packet in a greedy step, even for our algorithm. However, we need an adjusted analog of Lemma 2.16 for a leap step, since the algorithm modifies the weights and deadlines in leap steps. Then we use this lemma to prove the slot-monotonicity property stated in Lemma 2.20. In the following, we use P for P^t and by Q we denote

[4] We give an intuition behind such a complicated shift of packets in case L.2, where we analyze an iterated leap step.

the plan just after p is scheduled, the time is incremented and weights and deadlines are changed (and before new packets at time $t+1$ arrive).

Lemma 2.18. *Suppose that t is a leap step in which p was scheduled, and let $\varrho = \mathsf{sub}^t(p)$ be p's substitute packet. Let $\tau_0, \dots, \tau_k = \gamma = \mathsf{nextts}^t(d^t_\varrho)$ be as in the algorithm. Furthermore, let $\delta = \tau_{-1} := \mathsf{prevts}^t(d^t_p)$. Then:*

(a) $Q = P \setminus \{p, \omega\} \cup \{\varrho\}$; in particular, if $k \geq 1$, all packets $h_1, h_2, \dots, h_k$ are in Q.

(b) $w^t_p = w^t_{h_0} > w^t_{h_1} > w^t_{h_2} > \dots > w^t_{h_k} > w^t_\varrho$.

(c) h_k's deadline is in the segment of P ending at γ, that is, $\mathsf{prevts}^t(P, d^t_\varrho) < d^t_{h_k} \leq \gamma$.

(d) The pslack values change as follows (see Figure 2.10)

(d.i) For $\tau \in [t+1, d^t_\omega)$, we have $\mathsf{pslack}(Q,\tau) = \mathsf{pslack}(P,\tau) - 1$.

(d.ii) For $\tau \in [d^t_\omega, \delta]$ it holds that $\mathsf{pslack}(Q,\tau) = \mathsf{pslack}(P,\tau)$.

(d.iii) If $k \geq 1$, then for $i = 0, \dots, k-1$ we have the following changes in $(\tau_{i-1}, \tau_i]$:

(d.iii.1) For $\tau \in (\tau_{i-1}, d^t_{h_i})$ and for $\tau = \tau_i$, we have $\mathsf{pslack}(Q,\tau) = \mathsf{pslack}(P,\tau)$.

(d.iii.2) For $\tau \in [d^t_{h_i}, \tau_i)$, it holds that $\mathsf{pslack}(Q,\tau) = \mathsf{pslack}(P,\tau) + 1$,

(d.iv) For $\tau \in (\tau_{k-1}, \min(d^t_{h_k}, d^t_\varrho))$, we have $\mathsf{pslack}(Q,\tau) = \mathsf{pslack}(P,\tau)$.

(d.v) For $\tau \in [\min(d^t_{h_k}, d^t_\varrho), \max(d^t_{h_k}, d^t_\varrho))$, there are two cases:

(d.v.1) If $d^t_{h_k} < d^t_\varrho$, then $\mathsf{pslack}(Q,\tau) = \mathsf{pslack}(P,\tau) + 1$ for $d^t_{h_k} \leq \tau < d^t_\varrho$.

(d.v.2) If $d^t_\varrho < d^t_{h_k}$, then $\mathsf{pslack}(Q,\tau) = \mathsf{pslack}(P,\tau) - 1$ for $d^t_\varrho \leq \tau < d^t_{h_k}$.

(d.vi) Finally, for $\tau \geq \max(d^t_{h_k}, d^t_\varrho)$ we have $\mathsf{pslack}(Q,\tau) = \mathsf{pslack}(P,\tau)$.

(e) Any tight slot of P is a tight slot of Q, but there might be new tight slots before d^t_ω and in case (d2) also in $[d^t_\varrho, d^t_{h_k})$. In other words, $\mathsf{nextts}(Q,\tau) \leq \mathsf{nextts}(P,\tau)$.

(f) $w^{t+1}_{h_i} \geq \mathsf{minwt}^t(d^{t+1}_{h_i})$ for any $i = 1, \dots, k$, and $w^{t+1}_\varrho \geq \mathsf{minwt}^t(d^{t+1}_\varrho)$.

Proof. (a) The claim clearly holds for a simple leap step by Lemma 2.16, i.e., when $k = 0$, since the algorithm does not decrease deadlines if $k = 0$ and since increasing the weight of ϱ does not change the new plan Q.

Thus consider an iterated leap step. Let $\overline{Q}$ be the plan after p is scheduled and the time is increased, but before weights and deadlines are adjusted by the algorithm, i.e., $\overline{Q}$ is with respect to weights and deadlines at time t. Thus $\overline{Q}$ is the same as Q in Lemma 2.16, which yields that $\overline{Q} = P \setminus \{p, \omega\} \cup \{\varrho\}$; in particular, packets $h_1, h_2, \dots, h_k$ are in $\overline{Q}$. Our goal is to prove $\overline{Q} = Q$. Clearly, increasing the weights of packets in the plan cannot change the plan, but the algorithm also decreases the deadlines of $h_1, h_2, \dots, h_k$. To see that no h_i is forced out of the plan because of decreasing the deadlines, consider again $\overline{Q}$ and note that by Lemma 2.16 all segments between $\delta = \mathsf{prevts}^t(d^t_p)$ and $\gamma = \mathsf{nextts}^t(d^t_\varrho)$ get merged into one and in particular, $\mathsf{pslack}(\overline{Q}, \tau) \geq 1$ for any $\tau \in (\delta, \gamma)$.

Decreasing the deadline of h_i from $d^t_{h_i}$ to $d^{t+1}_{h_i} = \tau_{i-1}$ decreases $\mathsf{pslack}(\tau)$ by one for $\tau \in [\tau_{i-1}, d^t_{h_i})$, $i = 1, \dots, k$; all these intervals are contained in (δ, γ) and since $d^t_{h_i} \leq \tau_i$, these intervals do not overlap. Thus after decreasing the deadlines of h_i's (and keeping the set of packets in the plan) no slot has a negative value of pslack. It follows that $\overline{Q} = Q$; in particular, packets $h_1, h_2, \dots, h_k$ remain in Q.

(b) Note that p is the heaviest packet in segments of P in $(\delta, \gamma]$ as ϱ is the substitute packet for any packet in P in $(\delta, \gamma]$. Since $\delta < d^t_{h_i} \leq \gamma$ for $i = 1, \dots, k$, we get $w^t_p > w^t_{h_i}$.

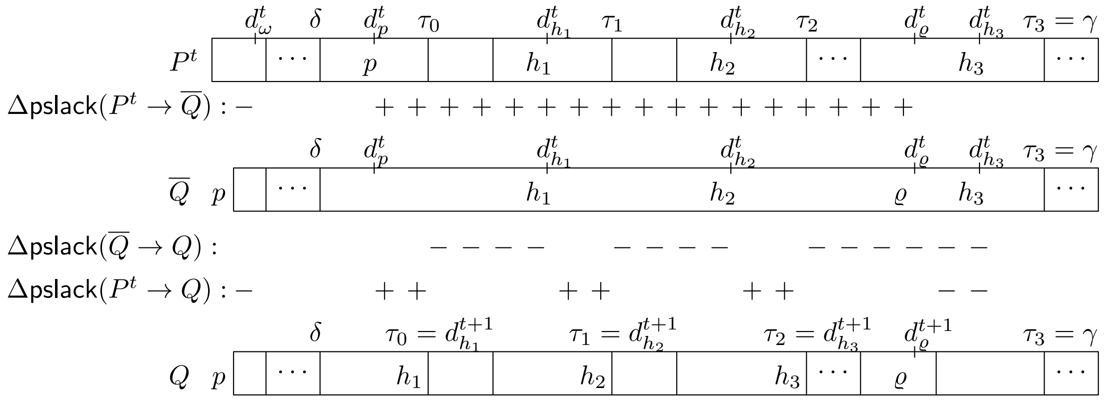
The ordering of weights of h_i's follows by the definition of h_i's in line 9 of the algorithm's description.

Item (c) holds by the definition of h_i's in line 9 and by the condition of the while loop in line 7.

(d) Note that for any slot $\tau \le \delta$ or $\tau > \gamma$ the value of $\mathsf{pslack}(\tau)$ is not affected by the decrease of the deadlines of h_i's, since h_i's are both in P and in Q and since their old and new deadlines are in $(\delta, \gamma]$. We thus get exactly the same changes of the pslack values outside $(\delta, \gamma]$ as in Lemma 2.16. Regarding slots in $(\delta, \gamma]$, Lemma 2.16(c1) shows that $\mathsf{pslack}(\overline{Q}, \tau)$ decreases by one for $\tau \in [d^t_\varrho, d^t_p)$ if $d^t_\varrho < d^t_p$, and increases by one for $\tau \in [d^t_p, d^t_\varrho)$ if $d^t_p < d^t_\varrho$. In the former case, we have $\tau_0 = \gamma$ as d^t_ϱ and d^t_p are in the same segment, thus $k = 0$. In the latter case, we sum the increase of $\mathsf{pslack}(\tau)$ for $\tau \in [d^t_p, d^t_\varrho)$ with the changes of the pslack values due to decreasing the deadlines of h_i's, analyzed in (a), and we get the same changes as in (d); see Figure 2.10.

Finally, item (e) follows from (c), (d) and the fact that the value of $\mathsf{pslack}(\tau)$ does not increase for any tight slot in P.

Item (f) follows from changing the weights in lines 4 and 11. □



d^t_ω δ d^t_p τ_0 $d^t_{h_1}$ τ_1 $d^t_{h_2}$ τ_2 d^t_ϱ $d^t_{h_3}$ $\tau_3 = \gamma$
P^t
$\cdots$ p h_1 h_2 $\cdots$ h_3 $\cdots$
$\Delta\mathsf{pslack}(P^t \to \overline{Q}) : -$ $+ + + + + + + + + + + + + + + + + +$
δ d^t_p $d^t_{h_1}$ $d^t_{h_2}$ d^t_ϱ $d^t_{h_3}$ $\tau_3 = \gamma$
$\overline{Q}$ p
$\cdots$ h_1 h_2 ϱ h_3 $\cdots$
$\Delta\mathsf{pslack}(\overline{Q} \to Q) :$ $- - - -$ $- - - -$ $- - - - - - -$
$\Delta\mathsf{pslack}(P^t \to Q) : -$ $+ +$ $+ +$ $+ +$ $- -$
δ $\tau_0 = d^{t+1}_{h_1}$ $\tau_1 = d^{t+1}_{h_2}$ $\tau_2 = d^{t+1}_{h_3}$ d^{t+1}_ϱ $\tau_3 = \gamma$
Q p
$\cdots$ h_1 h_2 h_3 $\cdots$ ϱ $\cdots$

Figure 2.10: An illustration of changes of the pslack values in an iterated leap step. For plans P', Q', by $\Delta\mathsf{pslack}(P' \to Q')$ we denote the difference of the pslack values of P and of Q. The plus sign represents that $\mathsf{pslack}(\tau)$ increases by one, while the minus sign represents that $\mathsf{pslack}(\tau)$ decreases by one.

Next, we show that the minimum weight in the plan does not decrease even in a leap step.

Lemma 2.19. *If step t is a leap step, then $\mathsf{minwt}(Q, \tau) \ge \mathsf{minwt}(P, \tau)$ for any $\tau > t$.*

Proof. We use notation from Lemma 2.18. By Lemma 2.18(e) all tight slots of P are tight slots of Q (in particular, δ and γ remain tight slots); thus $\mathsf{nextts}(Q, \tau) \le \mathsf{nextts}(P, \tau)$. Fix τ, and let a be the packet that realizes $\mathsf{minwt}(Q, \tau)$, that is the minimum-weight packet in Q with $d^{t+1}_a \le \mathsf{nextts}(Q, \tau)$. We need to show that $w^{t+1}_a \ge \mathsf{minwt}(P, \tau)$. We have three cases.

Case 1: $\tau \le \delta$. Lemma 2.18(a) shows that ω is forced out of the plan, thus not in Q, and otherwise the set of packets in the plan with deadline at most δ does not change. It follows that a is in P and its weight was not increased. Moreover, $d^{t+1}_a = d^t_a \le \mathsf{nextts}(Q, \tau) \le \mathsf{nextts}(P, \tau)$, thus $w^{t+1}_a \ge \mathsf{minwt}(P, \tau)$.

Case 2: $\tau \in (\delta, \gamma]$. If $a \ne h_i$ for all $i = 1, \dots, k+1$, in particular, if $a \ne \varrho = h_{k+1}$, then a is also in P with the same deadline and weight. As $\mathsf{nextts}(Q, \tau) \le \mathsf{nextts}(P, \tau)$, we get $d^t_a \le \mathsf{nextts}(P, \tau)$ and $w^{t+1}_a \ge \mathsf{minwt}(P, \tau)$ easily follows.

Next, suppose that $a = h_i$ for some $i \in \{1, \ldots, k\}$ (excluding the case $a = h_{k+1} = \varrho$). Recall that $w_a^{t+1} \geq \mathsf{minwt}(P, \tau_{i-1})$ and that $d_a^{t+1} = \tau_{i-1}$. Since $\tau > \mathsf{prevts}(Q, \tau_{i-1})$ and since tight slots of P are tight also in Q, we have $\tau > \mathsf{prevts}(P, \tau_{i-1})$. This implies $\mathsf{minwt}(P, \tau_{i-1}) \geq \mathsf{minwt}(P, \tau)$ and we get $w_a^{t+1} \geq \mathsf{minwt}(P, \tau_{i-1}) \geq \mathsf{minwt}(P, \tau)$.

Finally, consider the case $a = \varrho$, which is similar to the previous case. Recall that $w_\varrho^{t+1} = \mathsf{minwt}(P, d_\varrho^t)$ and $d_\varrho^t = d_\varrho^{t+1}$. We have $\tau > \mathsf{prevts}(Q, d_\varrho^t)$, which implies $\tau > \mathsf{prevts}(P, d_\varrho^t)$. It follows that $w_\varrho^{t+1} = \mathsf{minwt}(P, d_\varrho^t) \geq \mathsf{minwt}(P, \tau)$.

<u>Case 3</u>: $\tau > \gamma$. Note that the set of packets in the plan with deadline after γ does not change and also their weights and deadlines remain the same. Thus if $d_a^t > \gamma$, then using $\mathsf{nextts}(Q, \tau) \leq \mathsf{nextts}(P, \tau)$ again, we get $w_a^{t+1} \geq \mathsf{minwt}(P, \tau)$. Otherwise, $d_a^t \leq \gamma$, thus $w_a^{t+1} \geq \mathsf{minwt}(Q, \gamma) \geq \mathsf{minwt}(P, \gamma) \geq \mathsf{minwt}(P, \tau)$, where the second inequality follows from case 2 and the third one from $\gamma < \tau$. □

The previous lemma together with Lemma 2.14(b) for arrival of a new packet and Lemma 2.15(c) for a greedy step immediately yield the slot-monotonicity property in Lemma 2.20.

Lemma 2.20. *Let P be the current plan in step t just before an event of either arrival of a new packet, or scheduling a packet (and increasing the time), and let Q be the plan after the event. Then $\mathsf{minwt}(Q, \tau) \geq \mathsf{minwt}(P, \tau)$ for any $\tau > t$ and also for $\tau = t$ in the case of packet arrival.*

Hence, in the computation of Algorithm PlanM, for any fixed τ, function $\mathsf{minwt}^t(\tau)$ is non-decreasing in t as t grows from 0 to τ.

2.6.2 Competitive Analysis

Overview, Adversary Schedule, and Shadow Packets

Let ALG be the schedule of PlanM and let OPT be a fixed optimal schedule, also called the adversary schedule (actually, OPT can be any schedule for the instance). Our goal is to show that $\phi \cdot w^0(\mathsf{ALG}) \geq w^0(\mathsf{OPT})$, where w^0 refers to the original weights of packets.

We bound the competitive ratio via amortized analysis, using a combination of three techniques:

- In leap steps, when the algorithm increases weights of some packets (the substitute packet and some h_i's), we charge it a "penalty" equal to ϕ times the total weight increase. We remark that we increase the weight only for the algorithm and not in the adversary schedule.
- We use a potential function, which quantifies the advantage of the algorithm over the adversary in future steps. This potential function is defined in Section 2.6.2.
- During the analysis, some packets in the adversary schedule are replaced by lighter packets. If this happens, we add the appropriate "credit" (equal to the weight decrease) to the adversary's gain in this step. In the rest of this subsection, we explain how the adversary schedule is maintained and its packets replaced.

Adversary schedule. We actually work with the adversary schedule ADV that serves as a method of bookkeeping of future adversary's gain on the packets that are already released. (Abusing notation, we use ADV to also denote the set of packets in the adversary schedule.) We enforce that ADV has two types of packets only: (i) real packets that are pending also for the algorithm and are in the plan, and (ii) virtual *shadow packets* that we introduce below. Schedule ADV evolves as follows. Initially, ADV is empty. When a packet j arrives and $j \in \mathsf{OPT}$, then we add j to ADV to the slot in which j is in OPT; note that this slot is empty in ADV. In each step t, we remove packet $\mathsf{ADV}[t]$ from ADV (and the adversary gains its weight). We also occasionally modify ADV by replacing some packets by different and lighter packets (and the adversary gains the credit equal to the weight decrease); as a result, in general, $\mathsf{ADV}[\tau]$ need not be equal to $\mathsf{OPT}[\tau]$.

Packet replacements. We now describe the process of replacing some packets in ADV by lighter packets. Such replacement is done for packets in ADV that are also in the current plan of the algorithm. (Later, we describe replacing packets in ADV that are not in the plan.) So let g be the packet scheduled in slot τ in ADV that we want to replace. Depending on circumstances, we replace g either by a new virtual packet or by another packet in the plan. This new packet will be in the same slot τ in ADV. We now describe these two types of replacements.

Replacement by a shadow packet. In most cases, we replace $g = \mathsf{ADV}[\tau]$ by a new "virtual" packet s, called a *shadow* packet, with weight $\mathsf{minwt}^t(d^t_g)$. Shadow packets exist only in ADV — they are not pending for the algorithm at any time and we do not need to impose a canonical order on them. Consequently, they are also exempt from assumption (A2), which is why the weight $\mathsf{minwt}^t(d^t_g)$ need not be perturbed. Once created, shadow packets are tied to their specific slot and never change, and thus there is no need to specify their release times and deadlines. As g is in the plan, its weight is at least $\mathsf{minwt}^t(d^t_g)$. Thus s is no heavier than g, and its weight will never exceed $\mathsf{minwt}^t(d^t_g)$ in the future, due to the slot-monotonicity property (Lemma 2.20). In essence, shadow packets are an accounting trick: They represent deposits of profit, to be collected when the time reaches their associated time slot.

Replacement by another packet from the plan. In an iterated leap step, we replace each h_i, $i = 0, \ldots, k$, that is in ADV. Some packets $h_i \in \mathsf{ADV}$ are replaced by a shadow packet, but in a certain case, we replace a packet $h_i \in \mathsf{ADV}$ by h_{i+1}. (As a forward reference, we note that this replacement happens only in the case M.ii when processing a middle group of segments.) Recall that h_{i+1} is lighter than h_i (cf. Lemma 2.18(b)); furthermore, we guarantee that in the case of this replacement the weight of h_{i+1} does not change. As a result, we again replace a packet by a lighter packet. During these replacements, we also guarantee that at the end of the process no packet h_i appears twice in ADV.

Replacing real packets not in the plan. As mentioned earlier, all real pending packets that are in ADV must be in the current plan P^t. To ensure this, we replace each real packet in $\ell \in \mathsf{ADV} \setminus P^t$ by a new shadow packet s of the same weight as ℓ and in the same slot of ADV. Naturally, the adversary gets no credit for this replacement. Notice that packet ℓ (and thus s as well) has weight below $\mathsf{minwt}^t(d_\ell)$. Since ADV does not contain any packets pending for the algorithm that are not in P^t, the substitute packet added to the plan in a leap step is never in ADV (although its shadow copy may be in ADV).

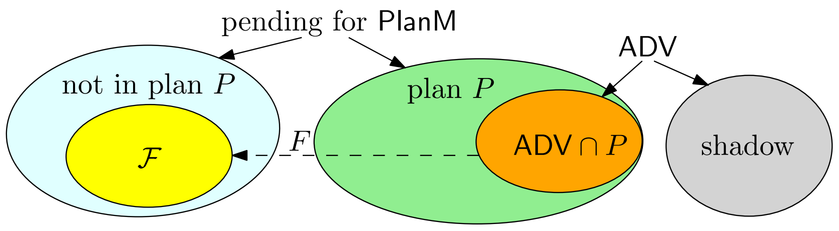


Figure 2.11: The sets of packets in the competitive analysis. Set $\mathcal{F}$ together with bijection $F : \mathsf{ADV} \cap P \to \mathcal{F}$ are introduced in Section 2.6.2.

Summary. To summarize, we will maintain the invariant that the packets in ADV in each step t are of two types:

- Packets in $\mathsf{ADV} \cap P^t$. Each packet $m \in \mathsf{ADV} \cap P^t$ satisfies $w^t_m \geq \mathsf{minwt}^t(d^t_m)$ and has the same weight in ADV and in P^t. These packets may be changed in ADV in the future.
- Packets in $\mathsf{ADV} \setminus P^t$, which are all shadow and not pending for the algorithm. These packets are never changed in the future. Each shadow packet s in slot τ_s in ADV satisfies $w_s \leq \mathsf{minwt}^t(\tau_s)$ and, by Lemma 2.20, its weight does not exceed $\mathsf{minwt}^t(\tau_s)$ until it is scheduled by the adversary.

Set $\mathcal{F}$ and Invariant (P)

In our analysis we maintain a set $\mathcal{F}$, which is a subset of "forced-out" pending packets, i.e., packets that got ousted from the plan, either by arrivals of other packets or in a leap step. A useful property of $\mathcal{F}$ is each packet in $\mathcal{F}$ can be used as a substitute packet (if it has an appropriate deadline).

In our analysis we will maintain the invariant that $|\mathcal{F}| = |\mathsf{ADV} \cap P|$ (where P is the current plan). We also use the following natural bijection F between $\mathsf{ADV} \cap P$ and $\mathcal{F}$: Let $f_1, \ldots, f_\ell$ be all packets in $\mathcal{F}$ in the canonical ordering, i.e., $d_{f_1} \leq d_{f_2} \leq \cdots \leq d_{f_\ell}$ (breaking ties in favor of heavier packets), and let $g_1, \ldots, g_\ell$ be all packets in $\mathsf{ADV} \cap P$, again in the canonical ordering. Then $F(g_i) = f_i$.

For each slot $\tau \geq t$ of the current plan, we define a quantity that will be crucial in our analysis; its name is explained in Section 2.6.2:

$$\#\mathsf{pairs}(\tau) = |\mathcal{F}_{\leq\tau}| - |(\mathsf{ADV} \cap P)_{\leq\tau}|\,; \tag{2.1}$$

(Recall that $X_{\leq\tau} = \{x \in X : d^t_x \leq \tau\}$.)

Throughout the analysis, we will maintain the following important invariant which relates the values of pslack and of $\#\mathsf{pairs}$:

$$\text{For any slot } \tau \geq t \text{ it holds that } \mathsf{pslack}(P, \tau) \geq \#\mathsf{pairs}(\tau). \tag{P}$$

After expanding the definitions of $\mathsf{pslack}(P, \tau)$ and of $\#\mathsf{pairs}(\tau)$ and rearranging, we get that invariant (P) for a slot τ can equivalently be defined as $|\mathcal{F}_{\leq\tau}| + |(P \setminus \mathsf{ADV})_{\leq\tau}| \leq \tau - t + 1$. Thus, intuitively, this invariant guarantees that if we modify P by replacing any subset of packets $g \in \mathsf{ADV} \cap P$ by the corresponding packets $F(g)$, we obtain a feasible set of pending packets.

After each event (i.e., arrival of a packet or scheduling a packet) we change the adversary schedule ADV and set $\mathcal{F}$ so that invariant (P) is preserved. Sometimes, it will be convenient to show this separately in the segments of the plan. Namely, we say that invariant (P) holds for segment S of the plan (or more generally, for an interval S of slots) if (P) holds for any $\tau \in S$.

Potential Function and Overview of the Analysis

Potential function. Sets $\mathcal{F}$, ADV and P undergo changes in the course of our analysis, not only when a packet is scheduled, but also when new packets arrive. We thus index these sets not by the current time, but by *events*, where an event is either the arrival of a new packet, or scheduling a packet in step t (together with increasing the time). Events are numbered by integers, starting from 0. Let P_σ be the plan just before event σ; similarly for $\mathcal{F}$ and ADV. Note that if σ is the scheduling event in step t, then $P^t = P_\sigma$.

The potential just before event σ at time t is the following:

$$\Psi_\sigma := \tfrac{1}{\phi} \cdot \left[\, w^t(P_\sigma) + w^t(\mathcal{F}_\sigma) - w^t(\mathsf{ADV}_\sigma \cap P_\sigma) \,\right]. \tag{2.2}$$

We remark that the potential can equivalently be defined as $\frac{1}{\phi}[w^t(P_\sigma \setminus \mathsf{ADV}_\sigma) + w^t(\mathcal{F}_\sigma)]$, but it is more convenient to explicitly have the term $w^t(\mathsf{ADV}_\sigma \cap P_\sigma)$ in the potential.

Initial and final state. At the beginning, before any packet arrives, we assume that the plan is filled with virtual 0-weight packets, each in a slot equal to its deadline, and none of them scheduled by the adversary. Both set $\mathcal{F}$ and the adversary schedule ADV are empty, thus invariant (P) clearly holds, and $\Psi_0 = 0$. At the end, after all (non-virtual) packets expire, the potential is zero as well.

Adversary gain. In each step t, *the adversary gain*, denoted $\mathsf{advgain}^t$, is the weight of packet $\mathsf{ADV}[t]$ that the adversary schedules in step t plus the credit (the difference between old and new weights) for replacing some packets in ADV by lighter packets. Since each packet in OPT is added to ADV upon its arrival with its original weight into an empty slot and since the adversary gets a corresponding credit for each packet that it schedules and for each replacement in ADV, the sum of $\mathsf{advgain}^t$ over all steps t equals $w^0(\mathsf{OPT})$ as desired.

Amortized analysis. At the core of our analysis are bounds relating amortized gains of the algorithm and the adversary at each event σ. If σ is the index of a packet arrival event, then we will show the following *packet-arrival inequality*:

$$\Psi_{\sigma+1} - \Psi_\sigma \geq 0\,. \tag{2.3}$$

If σ is the index of the scheduling event in a step t, then we will show that the following *packet-scheduling inequality* holds:

$$\phi \cdot w^t(\mathsf{ALG}[t]) - \phi \cdot (\Delta^t\text{Weights}) + (\Psi_{\sigma+1} - \Psi_\sigma) - \mathsf{advgain}^t \geq 0, \tag{2.4}$$

where $\mathsf{ALG}[t]$ is the packet in step t in the algorithm's schedule ALG (thus the algorithm's gain), w^t refers to its weight at time t, and Δ^tWeights is the total amount by which the algorithm increases the weights of its pending packets in step t.

We prove the packet-arrival inequality in Section 2.6.3 and the packet-scheduling inequality in Section 2.6.4. Assuming these two, we now show our main result.

Theorem 2.21. *Algorithm PlanM is ϕ-competitive for Bounded-Delay Packet Scheduling.*

Proof. We show that $\phi \cdot w^0(\mathsf{ALG}) \geq w^0(\mathsf{OPT})$, which implies the theorem. First, note that the sum of terms $\Psi_{\sigma+1} - \Psi_\sigma$ over all events σ equals $\Psi_{T+1} - \Psi_0$, where $\Psi_0 = 0$ is the initial potential and $\Psi_{T+1} = 0$ is the final potential after the last (scheduling) event T. Second, as we noted above, the sum of $\mathsf{advgain}^t$ over all steps t equals $w^0(\mathsf{OPT})$. Finally, observe that

$$\sum_t [\, w^t(\mathsf{ALG}[t]) - (\Delta^t\text{Weights})\,] \leq w^0(\mathsf{ALG})\,. \tag{2.5}$$

This follows from the observation that if the weight of $\mathsf{ALG}[t]$ was increased by some value $\delta > 0$ at some step $t' < t$, then δ also contributes to $\Delta^{t'}$Weights. (There may be several such δ's, as the weight of a packet may have been increased multiple times.) Note that the bound (2.5) may not be tight if some packets with increased weights are later dropped.

Hence, using (2.3) for each arrival event, (2.4) for each scheduling event, and (2.5) yields

$$\begin{aligned}0 &\leq \sum_\sigma (\Psi_{\sigma+1} - \Psi_\sigma) + \sum_t \left(\phi \cdot w^t(\mathsf{ALG}[t]) - \phi \cdot (\Delta^t\text{Weights}) - \mathsf{advgain}^t\right)\\ &\leq \phi \cdot w^0(\mathsf{ALG}) - w^0(\mathsf{OPT})\,,\end{aligned}$$

concluding the proof. □

Pairs and Consequences of Invariant (P)

In this section, we first give an intuitive view of bijection $F : \mathsf{ADV} \cap P \to \mathcal{F}$ and invariant (P) and then we state the corollaries of the invariant. Recall that bijection F assigns f_i to each $g_i \in \mathsf{ADV} \cap P$, where f_i and g_i are the i-th packets in the canonical ordering of $\mathcal{F}$ and $\mathsf{ADV} \cap P$, respectively. An equivalent view is that there are pairs (f_i, g_i), $i = 1, \ldots, \ell$; we will work with both, i.e., with these pairs and F.

We classify the pairs and define their *d-intervals* as follows: A pair (f, g) is *positive* if $d_f < d_g$, *negative* if $d_f > d_g$, and otherwise, if $d_f = d_g$, the pair is *neutral.* The d-interval of a pair (f, g) is $[d_f, d_g)$ if the pair is positive, and $[d_g, d_f)$ otherwise. Note that the d-interval of a pair is always left-closed and right-open. Moreover, a pair contains a slot τ if its d-interval contains τ, i.e., if $d_f \le \tau < d_g$ for a positive pair, and if $d_g \le \tau < d_f$ for a negative pair. A neutral pair does not contain any slot, as the corresponding d-interval is empty.

By the definition of F, the pairs are *agreeable*, i.e., for any two pairs (f, g) and (f', g'), if $d_f < d_{f'}$, then $d_g \le d_{g'}$. Indeed, if $d_f < d_{f'}$, then f is before f' in the canonical ordering of $\mathcal{F}$, thus also g is before g' in the canonical ordering of $\mathsf{ADV} \cap P$ and $d_g \le d_{g'}$ follows. Similarly, a positive pair does not *overlap* with a negative pair (f', g'), i.e., there is no slot contained in both pairs.

Recall that $\#\mathsf{pairs}(\tau) = |\mathcal{F}_{\le\tau}| - |(\mathsf{ADV} \cap P)_{\le\tau}|$. Observe that $\#\mathsf{pairs}(\tau)$ equals the number of positive pairs containing τ minus the number of negative pairs containing τ. As positive and negative pairs do not overlap, $\#\mathsf{pairs}(\tau)$ is either the number of positive pairs containing τ, or minus the number of negative pairs containing τ.

Since $\mathsf{pslack}(\tau)$ is non-negative, an equivalent formulation of invariant (P) is that $\mathsf{pslack}(\tau)$ is at least the number of positive pairs containing the slot τ. From the invariant it follows that there is no positive pair containing a tight slot, although a negative pair may contain a tight slot. It follows that the d-interval of a positive pair is fully contained in a single segment of the plan, while the d-interval of a negative pair may span several segments.

The important, though simple consequences of invariant (P) are summarized in the following lemma, which in particular shows that each g in $\mathsf{ADV} \cap P$ has a good substitute packet.

Lemma 2.22. *Suppose that g is in $\mathsf{ADV} \cap P$ and let $f = F(g)$, i.e., (f, g) is a pair. Then:*

(a) $d_f > \mathsf{prevts}(P, d_g)$,
(b) $w(\mathsf{sub}(P, g)) \ge w_f$, *and*
(c) $w_f < \mathsf{minwt}(P, d_g) \le w_g$.

Proof. (a) Since $\delta = \mathsf{prevts}(P, d_g)$ is a tight slot, $\mathsf{pslack}(P, \delta) = 0$, and invariant (P) for δ implies that no positive pair contains δ. As $\delta < d_g$ by definition, $d_f > \delta$ follows.

(b) Note that $f \in \mathcal{F}$ is pending, but not in P. If $g \in S_1$, then $\mathsf{sub}(P, g) = \omega$ and $w_\omega \ge w_f$ as ω is heavier than any pending packet not in P. Otherwise, by (a) $d_f > \delta = \mathsf{prevts}(P, d_g)$ and thus f is a candidate for the substitute packet $\mathsf{sub}(P, g)$, which implies the inequality.

(c) As f is pending, but not in P and as $d_f > \mathsf{prevts}(P, d_g)$ by (a), we have $w_f < \mathsf{minwt}(P, d_g)$. The inequality $\mathsf{minwt}(P, d_g) \le w_g$ follows from the fact that $g \in P$. □

The next lemma bounds the number of packets in $\mathcal{F}$ that are expiring in the current step t.

Lemma 2.23. *In each step t, $|\mathcal{F}_{\le t}| \le 1$, i.e., there is at most one packet $f \in \mathcal{F}$ with $d_f = t$.*

Proof. Recall that invariant (P) can equivalently be stated as $|\mathcal{F}_{\le\tau}| \le (\tau - t + 1) - |(P \setminus \mathsf{ADV})_{\le\tau}|$. For $\tau = t$, this gives $|\mathcal{F}_{\le t}| \le 1 - |(P \setminus \mathsf{ADV})_{\le t}| \le 1$. □

Finally, in some cases of the analysis we just remove a pair (f, g), that is, whenever g is either removed from ADV or no longer in the plan, then we remove the corresponding $f = F(g)$ from $\mathcal{F}$. The next observation shows that this affects $\#\mathsf{pairs}$ in a way that preserves invariant (P).

Lemma 2.24. *Suppose that (f, g) is a pair, f is removed from $\mathcal{F}$, and g is removed from $\mathsf{ADV} \cap P$. If $d_f \le d_g$, then $\#\mathsf{pairs}(\tau)$ decreases by one for $\tau \in [d_f, d_g)$. Otherwise, $d_f > d_g$ and $\#\mathsf{pairs}(\tau) \le 0$ for $\tau \in [d_g, d_f)$ both before and after these removals. In both cases, for other slots the value of $\#\mathsf{pairs}$ stays the same.*

Moreover, in both cases, other pairs remain the same.

Proof. First, suppose $d_f \le d_g$. Note that $\#\mathsf{pairs}(\tau)$ remains the same for $\tau \ge d_g$ and for $\tau < d_f$ as both f and g are taken into account before their removals, or none of them, respectively. For $\tau \in [d_f, d_g)$, only f appears in (2.1), thus $\#\mathsf{pairs}(\tau)$ decreases by one after we remove τ.

Otherwise $d_f > d_g$. Similarly, $\#\mathsf{pairs}(\tau)$ remains the same for $\tau \ge d_f$ and for $\tau < d_g$. Consider a slot $\tau \in [d_g, d_f)$ and note that for such a slot, $\#\mathsf{pairs}(\tau)$ increases by one as only g was taken into account and not f. Recall that the position of f in the canonical ordering of $\mathcal{F}$ is the same as the position of g in the canonical ordering of $\mathsf{ADV} \cap P$. We get that $|\mathcal{F}_{\le\tau}| < |(\mathsf{ADV} \cap P)_{\le\tau}|$, meaning that $\#\mathsf{pairs}(\tau) < 0$ before the removals. It follows that $\#\mathsf{pairs}(\tau) \le 0$ after the removals.

The claim that other pairs are unchanged follows from the fact that the positions of f and g in the canonical orderings of $\mathcal{F}$ and $\mathsf{ADV} \cap P$, respectively, were equal. □

2.6.3 Arrival of a Packet

Let σ be the index of an arrival event. Let $P = P_\sigma$ be the plan just before a new packet j arrives and let $Q = P_{\sigma+1}$ be the plan just after j arrives. Our aim is to maintain invariant (P) using appropriate modifications of sets $\mathcal{F}$ and ADV. We also show that the packet-arrival inequality (2.3) holds after the modifications. All weights and deadlines in this subsection are implicitly at the current time t (as the algorithm does not change the weights and deadlines after packet arrival). There are two cases, depending on whether or not $j \in Q$.

Case A.1: j is not added to the plan, i.e., $P = Q$. Then $w_j < \mathsf{minwt}(P, d_j) = \mathsf{minwt}(Q, d_j)$. If $j \in \mathsf{OPT}$, we add a new shadow packet s of weight w_j to the adversary schedule ADV to the slot where j is in OPT. Otherwise, i.e., if $j \notin \mathsf{OPT}$, we do nothing. In both subcases $\Psi_{\sigma+1} = \Psi_\sigma$, implying the packet-arrival inequality (2.3). Also, the functions pslack and $\#\mathsf{pairs}$ do not change, so invariant (P) is preserved.

Case A.2: j is added to the plan. Let u be the lightest packet in P with $d_u \le \mathsf{nextts}(P, d_j)$; by assumption (A1) u exists. Then $Q = P \cup \{j\} \setminus \{u\}$ and $w_j > w_u$ by Lemma 2.14. Note that the values of pslack change according to Lemma 2.14.

Dealing with $u \in \mathsf{ADV}$. If u is in ADV, then $u \in \mathsf{ADV} \cap P$, so $F(u) \in \mathcal{F}$ is defined. In this case, we remove packet $F(u)$ from $\mathcal{F}$. Moreover, $u \notin \mathsf{ADV} \cap Q$ and we replace u in ADV by a new shadow packet s of weight w_u, which is placed in ADV in the former slot of u. We use Lemma 2.24 for u and $F(u)$ to get that invariant (P) is preserved by these removals. The contribution of these changes to the potential change is positive, as $w(\mathcal{F})$ decreases by $w_{F(u)}$, but $w(\mathsf{ADV} \cap P)$ decreases by w_u and $w_u > w_{F(u)}$, by

Lemma 2.22(c). In the cases below, when bounding $\Psi_{\sigma+1} - \Psi_\sigma$, we will account for this contribution without an explicit reference.

There are several cases, depending on whether or not $j \in \mathsf{OPT}$ and on the ordering of d_u and d_j.

Case A.2.a: j is not in the optimal schedule OPT.

Case A.2.a.P: $d_u \le d_j$ (the positive case). We do not further change $\mathcal{F}$ or ADV. Thus $\Psi_{\sigma+1} - \Psi_\sigma \ge \frac{1}{\phi}(w(Q) - w(P)) = \frac{1}{\phi}(w_j - w_u) > 0$. The function $\#\mathsf{pairs}$ does not change and pslack does not decrease, so invariant (P) is preserved.

Case A.2.a.N: $d_u > d_j$ (the negative case). Let $\delta = \mathsf{prevts}(P, d_u)$. If there is no packet $f \in \mathcal{F}$ with $d_f \in (\delta, d_u)$, then we do nothing. Otherwise, let f^* be the earliest-deadline packet in $\mathcal{F}$ with deadline after δ, i.e., the first such packet in the canonical ordering of $\mathcal{F}$; note that $d_{f^*} < d_u$. We remove f^* from $\mathcal{F}$ and add u to $\mathcal{F}$.

If we replaced f^* by u in $\mathcal{F}$, then, as f^* is pending but not in P and $d_{f^*} > \delta$, we get that $w_{f^*} < \mathsf{minwt}(P, d_u) \le w_u$. Since also $w_u < w_j$, we obtain that $\Psi_{\sigma+1} - \Psi_\sigma \ge \frac{1}{\phi}(w_j - w_u + w_u - w_{f^*}) > 0$. Otherwise, $\Psi_{\sigma+1} - \Psi_\sigma \ge \frac{1}{\phi}(w_j - w_u) > 0$. This shows the packet-arrival inequality (2.3).

We claim that invariant (P) is maintained. Recall that by Lemma 2.14(a2) and the case assumption $d_u > d_j$, both d_j and d_u are in the same segment of P. Lemma 2.14(a2) also shows that $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau) - 1$ for $\tau = d_j, d_j + 1, \ldots, d_u - 1$, while for other slots pslack is not changed.

If there was no packet $f \in \mathcal{F}$ with $d_f \in (\delta, d_u)$, invariant (P) is preserved since no pair changes and since for any $\tau \in [d_j, d_u) \subseteq (\delta, d_u)$ we have $\#\mathsf{pairs}(\tau) \le 0$, whereas $\mathsf{pslack}(\tau)$ does not change for other τ.

Otherwise, $d_{f^*} < d_u$ and $\#\mathsf{pairs}(\tau)$ decreases by one for $\tau \in [d_{f^*}, d_u)$. Since $\#\mathsf{pairs}(\tau) \le 0$ for $\tau \in (\delta, d_{f^*})$ even after replacing f^* by u in $\mathcal{F}$, invariant (P) is maintained.

Case A.2.b: Otherwise, $j \in \mathsf{OPT}$. We add j to ADV in the same slot as in OPT and add u to $\mathcal{F}$. We first analyze $\Psi_{\sigma+1} - \Psi_\sigma$. The weight of the plan increases by $w(Q) - w(P) = w_j - w_u$, the term $w(\mathsf{ADV} \cap P)$ increases by w_j, and $w(\mathcal{F})$ increases by w_u. Summing it up, $\Psi_{\sigma+1} - \Psi_\sigma \ge \frac{1}{\phi}((w_j - w_u) - w_j + w_u) = 0$. Hence the packet-arrival inequality (2.3) holds.

We now show that (P) holds, splitting the proof into two cases, depending on the order of d_u and d_j:

Case A.2.b.P: $d_u \le d_j$ (the positive case). In this case, Lemma 2.14(a1) shows that $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau) + 1$ for $\tau \in [d_u, d_j)$, while for other slots pslack is not changed. It holds that $\#\mathsf{pairs}(\tau)$ increases by one for $\tau \in [d_u, d_j)$ and for other slots it stays the same. Hence (P) is maintained.

Case A.2.b.N: $d_u > d_j$ (the negative case). By Lemma 2.14(a2) we have that $\mathsf{pslack}(Q, \tau) = \mathsf{pslack}(P, \tau) - 1$ for $\tau \in [d_j, d_u)$, while for other slots pslack is not changed. Similarly, it holds that $\#\mathsf{pairs}(\tau)$ decreases by one for $\tau \in [d_j, d_u)$ and for other slots it stays the same, which implies that (P) holds.

2.6.4 Scheduling a Packet

After all packets with release time equal to t arrive, the algorithm schedules a packet p. Let j be the packet scheduled in ADV at t. Recall that since we change ADV, j is *not* necessarily the packet scheduled in step t in the original optimal schedule OPT; in particular, j can be a shadow packet that replaced it or another real packet. Let $P = P^t$ be the plan just before scheduling p and let Q be the plan after the algorithm schedules p and possibly adjusts weights and deadlines, and after the time is incremented.

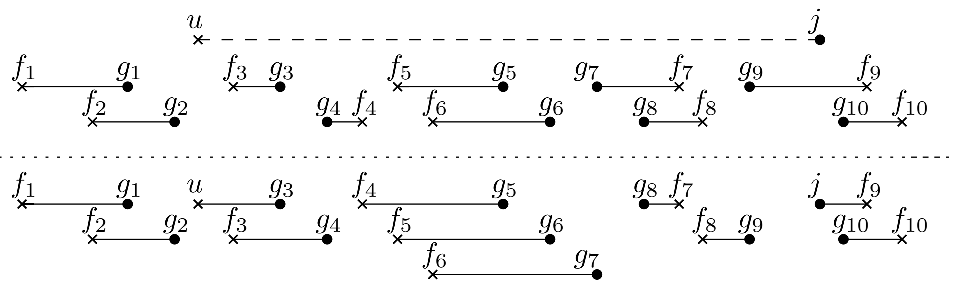


Figure 2.12: An illustration of the changes of the pairs in the positive case of packet arrival. In the upper part (above the dotted line) there are "old" pairs $(f_1, g_1), \dots, (f_\ell, g_\ell)$ and a (virtual) dashed pair (u, j). Each pair (f, g) is represented by its d-interval, i.e., the interval between d_f (depicted by a cross) and d_g (depicted by a dot). Below the dotted line, new pairs $(f'_1, g'_1), \dots, (f'_{\ell+1}, g'_{\ell+1})$ are shown. Note that $\#\mathsf{pairs}(\tau)$ increases by one for $\tau \in [d_u, d_j)$ and for other slots it stays the same; intuitively, this means that we add a new positive pair (u, j), and then restructure the pairs to the implicit structure; the latter does not change $\#\mathsf{pairs}(\tau)$ for any slot.

We split the scheduling step into the *adversary step* and the *algorithm's step* such that in the former, the adversary schedules j, but the plan remains the same, and in the latter, the algorithm schedules p and the time is increased, thus the plan changes from P to Q. We make changes in set $\mathcal{F}$ and in the adversary schedule ADV and then we show that invariant (P) holds after each of the two steps. Some of the changes are enforced by scheduling a packet or by changes of the plan, e.g., if a packet from $\mathcal{F}$ gets into the plan as a substitute packet or if the algorithm schedules a packet in ADV. Furthermore, during processing the algorithm's step, if there is an expiring packet in $\mathcal{F}$, we remove it; there is at most one such packet by Lemma 2.23. In addition to the enforced changes, we also do other deliberate modifications to maintain invariant (P).

First, we process the adversary step and bound the adversary gain in this step. Then we deal with the algorithm's step, which differs in a greedy step and in a leap step; these are the two main cases. In both of them, our goal is to show the packet-scheduling inequality (2.4). Processing of a leap step will be further divided into several parts.

Processing a step (or a part of it) means that we first make changes in set $\mathcal{F}$ and in ADV, then we show that invariant (P) is preserved and finally, we calculate the change of the potential caused by the changes together with the credit for the adversary for replacing packets in ADV and the penalty for increasing weights. To make calculations less cluttered, j stands for w^t_j and similarly for other packets. (Later, in the case of a leap step, we extend the notation, as the weights change.)

Adversary Step

The adversary schedules $j = \mathsf{ADV}[t]$, thus j is removed from the adversary schedule ADV. If $j \in P$, we also need to remove a packet from $\mathcal{F}$.

Changes in the adversary step. If $j \in P$, then $F(j) \in \mathcal{F}$ is defined. We remove $F(j)$ from $\mathcal{F}$.

Invariant (P) after the adversary step. If $j \notin P$, then invariant (P) is clearly preserved. Otherwise, j is no longer in $\mathsf{ADV} \cap P$ and we removed $F(j)$ from $\mathcal{F}$, thus the pair $(F(j), j)$ is removed. By Lemma 2.24, other pairs remain the same and invariant (P) holds after the adversary step.

The calculation in the adversary step. The adversary gain $\mathsf{advgain}^t$ is $j = w_j^t$ plus some credit for changing ADV in the algorithm's step (which we take into account when processing this step); we remark that if j is a shadow packet, then w_j^t refers to its weight upon creation (it never changes in future). By $\Delta_{\mathsf{ADV}}\Psi$ we denote the change of the potential in the adversary step. We now bound $\Delta_{\mathsf{ADV}}\Psi - j$, for which we have two cases:

Case ADV.1: $j \in P$. Then removing $F(j)$ from $\mathcal{F}$ decreases the potential by $F(j)/\phi$, but as j is no longer in $\mathsf{ADV} \cap P$, the potential increases by j/ϕ (the term $w^t(\mathsf{ADV} \cap P)$ decreases by j). By Lemma 2.22(b) we have $w(\mathsf{sub}^t(j)) \geq F(j)$. It follows that

$$\Delta_{\mathsf{ADV}}\Psi - j = \frac{j}{\phi} - \frac{F(j)}{\phi} - j = -\frac{j}{\phi^2} - \frac{F(j)}{\phi} \geq -\frac{j}{\phi^2} - \frac{w(\mathsf{sub}^t(j))}{\phi} \geq -\frac{p}{\phi^2} - \frac{w(\mathsf{sub}^t(p))}{\phi}, \tag{2.6}$$

where that last inequality follows from the choice of p in line 1 of the algorithm's description; here we use that $j \in P$.

Case ADV.2: $j \notin P$. In this case j is a shadow packet and thus $w_j \leq \mathsf{minwt}^t(d_j) \leq \omega$, as ω is in the first segment. Note that $\mathsf{sub}^t(\omega) = \omega$ and that $\Delta_{\mathsf{ADV}}\Psi = 0$. Then clearly

$$\Delta_{\mathsf{ADV}}\Psi - j = -j \geq -\omega = -\frac{\omega}{\phi^2} - \frac{w(\mathsf{sub}^t(\omega))}{\phi} \geq -\frac{p}{\phi^2} - \frac{w(\mathsf{sub}^t(p))}{\phi}, \tag{2.7}$$

where the last inequality holds by the choice of p again.

Thus in both cases

$$\Delta_{\mathsf{ADV}}\Psi - j \geq -\frac{p}{\phi^2} - \frac{w(\mathsf{sub}^t(p))}{\phi}. \tag{2.8}$$

Greedy Step

In this case p is the heaviest packet in the first segment of P. All weights and deadlines in this subsection are at time t, as the algorithm does not change them in a greedy step. Also $\Delta^t\text{Weights} = 0$. Since $w(\mathsf{sub}^t(p)) = \omega$, we get that $\Delta_{\mathsf{ADV}}\Psi - j \geq -p/\phi^2 - \omega/\phi$ by (2.8).

We now process the algorithm's step. First, note that packet p is in P, but not in Q, which decreases the potential by p/ϕ.

By Lemma 2.15 the values of $\mathsf{pslack}(P, \tau)$ are not changed for $\tau \geq d_p$, while for $\tau < d_p$, the value of $\mathsf{pslack}(P, \tau)$ decreases by one. We have two cases, depending on whether $p \in \mathsf{ADV}$ or a change of $\mathcal{F}$ and of ADV is necessary to maintain invariant (P).

Case G.1: If $p \notin \mathsf{ADV}$ and there is no positive pair containing a slot $\tau < d_p$, then we do not further change any set and $\mathsf{advgain}^t = j$. Invariant (P) holds since the pairs do not change, there is no positive pair containing a slot $\tau < d_p$ by the case assumption, and the value of $\mathsf{pslack}(\tau)$ remains the same for $\tau \geq d_p$.

We claim that there is no $f' \in \mathcal{F}$ with $d_{f'} = t$, which implies that no packet in $\mathcal{F}$ expires in this step. Suppose for a contradiction that $d_{f'} = t$. Note that f' is in a pair with g' and $d_{g'} > t$, since after the adversary step, there is no packet in ADV in slot t. We thus have $d_p > t$ as g' is in the first segment by invariant (P). Moreover, pair (f', g') is positive, implying that $t < d_p$ is in a positive pair, which contradicts the case condition.

The calculation showing the packet-scheduling inequality (2.4) is easy, as we just need to take into account the adversary gain bounded in (2.8) and the contribution of removing p from the plan to the potential change, denoted $\Delta_p\Psi$:

$$\begin{aligned}\phi \cdot w^t(\mathsf{ALG}[t]) - \phi \cdot (\Delta^t\text{Weights}) + (\Psi_{\sigma+1} - \Psi_\sigma) - \mathsf{advgain}^t \\ &= \phi \cdot p - \phi \cdot 0 + [\,\Delta_p\Psi + \Delta_{\mathsf{ADV}}\Psi\,] - j \\ &= \phi \cdot p + \Delta_p\Psi + [\,\Delta_{\mathsf{ADV}}\Psi - j\,] \\ &\geq \phi \cdot p - \frac{p}{\phi} + \left[-\frac{p}{\phi^2} - \frac{\omega}{\phi}\right] = \frac{p}{\phi} - \frac{\omega}{\phi} \geq 0\,,\end{aligned}$$

where we use $p \geq \omega$ in the last inequality.

Case G.2: Otherwise, $p \in \mathsf{ADV}$ or there is a positive pair containing a slot $\tau < d_p$.

Changes in case G.2. Let f_1 be the earliest-deadline packet in $\mathcal{F}$; note that possibly $d_{f_1} = t$, which means that in such a case, f_1 cannot be in $\mathcal{F}$ in the next step.

Let g^* be the latest-deadline packet in $\mathsf{ADV} \cap P$ with deadline at most $\beta := \mathsf{nextts}^t(d_p)$. Packet g^* is trivially defined if $p \in \mathsf{ADV}$. Otherwise, the case condition implies that there is a positive pair (f', g') containing a slot $\tau < d_p$, thus $d_{f'} \leq \tau < d_p$. By invariant (P), the first tight slot β is not contained in any positive pair, thus $d_{g'} \leq \beta$, so g' is a candidate for g^*.

If $p \in \mathsf{ADV}$, let $g = p$; otherwise let $g = g^*$. We remove f_1 from $\mathcal{F}$ and we replace g in ADV by a new shadow packet s of weight $\mathsf{minwt}^t(d_p) = \omega$, which is added to the slot of g in ADV. Of course, s is not pending for the algorithm. Note that by Lemma 2.23 all packets in $\mathcal{F}$ except f_1 have deadline after t and as we removed f_1, no packet in $\mathcal{F}$ expires in the current step.

Invariant (P) in case G.2. We show that invariant (P) holds. By Lemma 2.15 the value of pslack decreases by one for slots in (t, d_p) and for other slots it is not changed. We analyze how the values of $\#\mathsf{pairs}$ change. If $d_{f_1} < d_g$, then $\#\mathsf{pairs}(\tau)$ decreases by one for $\tau \in [d_{f_1}, d_g)$ and for other slots it remains the same. Otherwise, $d_{f_1} \geq d_g$ and the value of $\#\mathsf{pairs}(\tau)$ increases by one for $\tau \in [d_g, d_{f_1})$, while for other slots it does not change.

From the definitions of f_1, g^*, and invariant (P), we have that $\#\mathsf{pairs}(\tau) \leq 0$ holds for $\tau \in (t, d_{f_1}) \cup [d_{g^*}, \beta]$, even after the step. If follows that we just need to show that invariant (P) holds for $\tau \in [d_{f_1}, d_{g^*})$ and we can assume that $d_{f_1} < d_{g^*}$. In particular, as $d_{g^*} \leq \beta$ and $d_p \leq \beta$, invariant (P) holds for slots outside S_1.)

We only need to consider the case when either $\#\mathsf{pairs}(\tau)$ increases or $\mathsf{pslack}(\tau)$ decreases, because in other cases, the inequality $\mathsf{pslack}(\tau) \geq \#\mathsf{pairs}(\tau)$ is preserved after the step as these quantities change by at most 1.

The first case, when $\#\mathsf{pairs}(\tau)$ increases, is actually already covered. Indeed, if $\#\mathsf{pairs}(\tau)$ increases, then $d_g < d_{f_1}$ and $\tau \in [d_g, d_{f_1})$, thus $\#\mathsf{pairs}(\tau) \leq 0$ as shown above (even after the step).

The second case, when $\mathsf{pslack}(\tau)$ decreases, happens when $\tau < d_p$. This, combined with $\tau \in [d_{f_1}, d_{g^*})$, implies that $\tau \in [d_{f_1}, \min(d_p, d_{g^*}))$. As $g \in \{g^*, p\}$, it holds that $\tau \in [d_f, d_g)$, so $\#\mathsf{pairs}(\tau)$ decreases as well. Therefore, invariant (P) holds after the greedy step.

Calculation in case G.2. Let $\Delta_{p,g,f_1}\Psi$ be the change of Ψ caused by removing p from the plan, removing f_1 from $\mathcal{F}$, and g from $\mathsf{ADV} \cap P$, that is, $\Delta_{p,g,f_1}\Psi = (-p - f_1 + g)/\phi$. The adversary gain is $\mathsf{advgain}^t = g - s + j = g - \omega + j$. Note that $f_1 \leq \omega$ as $f_1 \notin P$ and that $g \leq p$ as p is the heaviest packet in S_1 and $d_g \leq \beta$. We show the packet-scheduling inequality (2.4) by summing these changes and the cost of the adversary step bounded in (2.8):

$$\begin{aligned}
\phi \cdot w^t(\mathsf{ALG}[t]) - \phi \cdot (\Delta^t \text{Weights}) + (\Psi_{\sigma+1} - \Psi_\sigma) - \mathsf{advgain}^t \\
&= \phi \cdot p - \phi \cdot 0 + [\,\Delta_{p,g,f_1}\Psi + \Delta_{\mathsf{ADV}}\Psi\,] - [\,g - \omega + j\,] \\
&= \phi \cdot p + \Delta_{p,g,f_1}\Psi - g + \omega + [\,\Delta_{\mathsf{ADV}}\Psi - j\,] \\
&\geq \phi \cdot p + \left[-\frac{p}{\phi} - \frac{f_1}{\phi} + \frac{g}{\phi}\right] - g + \omega + \left[-\frac{p}{\phi^2} - \frac{\omega}{\phi}\right] \\
&= \frac{p}{\phi} - \frac{f_1}{\phi} - \frac{g}{\phi^2} + \frac{\omega}{\phi^2} \\
&\geq \frac{p}{\phi} - \frac{\omega}{\phi} - \frac{p}{\phi^2} + \frac{\omega}{\phi^2} = \frac{p}{\phi^3} - \frac{\omega}{\phi^3} \geq 0\,,
\end{aligned}$$

where the penultimate inequality holds by $f_1 \leq \omega$ and by $g \leq p$, and the last inequality uses $p \geq \omega$. This concludes the analysis of a greedy step.

Leap Step

Suppose that the algorithm schedules p from a segment of P^t other than the first segment. Recall that in a leap step ω disappears from the plan, while $\varrho = \mathsf{sub}^t(p)$ appears. As in this case the weights change, we use ϱ^t for w^t_ϱ and ϱ^{t+1} for w^{t+1}_ϱ, and similarly for other packets. The deadlines are implicitly at time t, i.e., $d_a = d^t_a$. Let $\delta = \mathsf{prevts}^t(d^t_p)$ and $\gamma = \mathsf{nextts}^t(d^t_\varrho)$.

Regarding the cost in the adversary step, we have that $\Delta_{\mathsf{ADV}}\Psi - j^t \geq -p^t/\phi^2 - \varrho^t/\phi$ by (2.8). We now process the algorithm's step. For any $\tau \geq t$, we use the following inequalities

$$\frac{p^t}{\phi^2} + \frac{\varrho^t}{\phi} \geq \omega^t \geq \mathsf{minwt}^t(\tau)\,, \tag{2.9}$$

where the first one follows from the choice of p in line 1 of the algorithm's description and the second one uses $\omega^t = \mathsf{minwt}^t(t) \geq \mathsf{minwt}^t(\tau)$ by the monotonicity of minwt in τ.

We split processing the algorithm's step into more parts. We start by accounting for the changes of the plan and for increasing the weight of ϱ. Then we process the first segment S_1 and in most cases we make changes in $\mathcal{F}$ and in ADV with the goal to maintain invariant (P) in S_1, while not violating it outside S_1. The final part of the step will be processing segments in $(\delta, \gamma]$, which we further split into groups in the case of an iterated leap step (i.e., if $k > 0$). The changes of pslack are divided into two or more parts as well, i.e., changes of pslack in S_1 are taken into account when processing S_1 and similarly for the changes in $(\delta, \gamma]$.

In the rest of segments, i.e., in segments after S_1 not contained in $(\delta, \gamma]$, the values of pslack are not changed by Lemma 2.18. However, in some cases, we remove a packet with deadline in such a segment from ADV, namely, the latest-deadline packet which is in a segment before the interval $(\delta, \gamma]$. Similarly, from $\mathcal{F}$ we may remove some packets with deadline outside S_1 and outside $(\delta, \gamma]$. We show that such removals preserve invariant (P); more precisely, that for a slot $\tau \notin S_1 \cup (\delta, \gamma]$ either $\#\mathsf{pairs}(\tau) \leq 0$ even after the removals, or $\#\mathsf{pairs}(\tau)$ does not change.

Changes of the plan and increasing the weights. Recall that by Lemma 2.18(a) $Q = P \setminus \{p, \omega\} \cup \{\varrho\}$ and that the algorithm increases the weight of ϱ to $\mu := \mathsf{minwt}(P, d^t_\varrho)$. This changes the plan's weight by

$$\Delta_{p,\omega,\varrho} w(P) := -p^t - \omega^t + \mu \geq -p^t - \frac{p^t}{\phi^2} - \frac{\varrho^t}{\phi} + \mu\,, \tag{2.10}$$

where the inequality follows from (2.9). We do not take into account yet that the plan's weight changes by the increase of weights of h_i's in an iterated leap step. We remark that if still $p \in \mathsf{ADV}$ (and thus $p \neq j$) or if $\varrho \in \mathcal{F}$, we remove p from ADV or ϱ from $\mathcal{F}$ later, when we process $(\delta, \gamma]$. Similarly, we deal with $\omega \in \mathsf{ADV}$ when we process S_1. Note that $\varrho \notin \mathsf{ADV}$, since ϱ was not in the plan and we enforce that all real packets in ADV are in the plan. (However, a shadow copy of ϱ may be in ADV.)

Let $\Delta^t w_\varrho := \mu - \varrho^t$ be the amount by which the algorithm increases the weight of ϱ; we need to pay $\phi \cdot \Delta^t w_\varrho$ for this change. In the case of an iterated leap step, the algorithm increases the weights of h_i's as well. Let $H = \{h_1, \ldots, h_k\}$ be the set of h_i's and let $\Delta^t w(H)$ be the total amount by which the weights of h_i's are increased; we set $H = \emptyset$ and $\Delta^t w(H) = 0$ in the case of a simple leap step. Thus $\Delta^t \text{Weights} = \Delta^t w_\varrho + \Delta^t w(H)$.

Calculation in the leap step. Before processing S_1 and segments in $(\delta, \gamma]$, we show what we want to achieve in these parts, which we use to prove the packet-scheduling inequality (2.4).

Let $\Delta_{S_1}\Psi$ be the total change of the potential due to changes in $\mathcal{F}$ when processing S_1; we remark that we do not change ADV when processing the first segment. We show below that

$$\Delta_{S_1}\Psi \geq 0\,. \tag{2.11}$$

Similarly, let $\Delta_{(\delta,\gamma]}\Psi$ be the total change of the potential due to changes done when processing segments in $(\delta, \gamma]$ and let $\mathsf{advgain}^t_{(\delta,\gamma]}$ be the credit for the adversary for replacing packets in $(\delta, \gamma]$ in ADV by lighter packets; note that $\mathsf{advgain}^t = j^t + \mathsf{advgain}^t_{(\delta,\gamma]}$. We also include $\Delta^t w(H)$ in calculations when processing segments in $(\delta, \gamma]$ and our aim will be to show

$$\Delta_{(\delta,\gamma]}\Psi - \phi \cdot \Delta^t w(H) - \mathsf{advgain}^t_{(\delta,\gamma]} \geq -\frac{p^t}{\phi^2} - \frac{\varrho^t}{\phi} + \mu\,. \tag{2.12}$$

Note that $-p^t/\phi^2 - \varrho^t/\phi + \mu$ is non-positive as $\mu \leq p^t/\phi^2 + \varrho^t/\phi$ by (2.9), thus if we change nothing while processing segments in $(\delta, \gamma]$ and if the weights of h_i's do not change, (2.12) holds.

Given that (2.11) and (2.12) hold, we prove the packet-scheduling inequality (2.4) by the following calculation:

$$\begin{aligned}
&\phi \cdot w^t(\mathsf{ALG}[t]) - \phi \cdot (\Delta^t \text{Weights}) + (\Psi_{\sigma+1} - \Psi_\sigma) - \mathsf{advgain}^t \\
&\quad = \phi \cdot p^t - \phi \cdot [\, \Delta^t w_\varrho + \Delta^t w(H) \,] + [\, \tfrac{1}{\phi}\Delta_{p,\omega,\varrho} w(P) + \Delta_{\mathsf{ADV}}\Psi + \Delta_{S_1}\Psi + \Delta_{(\delta,\gamma]}\Psi \,] \\
&\qquad\quad - [\, j^t + \mathsf{advgain}^t_{(\delta,\gamma]} \,] \\
&\quad = \phi \cdot p^t - \phi \cdot (\Delta^t w_\varrho) + \tfrac{1}{\phi}\Delta_{p,\omega,\varrho} w(P) + [\, \Delta_{\mathsf{ADV}}\Psi - j^t \,] + \Delta_{S_1}\Psi \\
&\qquad\quad + [\, \Delta_{(\delta,\gamma]}\Psi - \phi \cdot \Delta^t w(H) - \mathsf{advgain}^t_{(\delta,\gamma]} \,] \\
&\quad \geq \phi \cdot p^t - \phi(\mu - \varrho^t) + \frac{1}{\phi}\left[-p^t - \frac{p^t}{\phi^2} - \frac{\varrho^t}{\phi} + \mu \right] + \left[-\frac{p^t}{\phi^2} - \frac{\varrho^t}{\phi} \right] + 0 \\
&\qquad\quad + \left[-\frac{p^t}{\phi^2} - \frac{\varrho^t}{\phi} + \mu \right] = 0
\end{aligned} \tag{2.13}$$

where the inequality uses, in this order, (2.10), (2.8), (2.11), and (2.12).

Processing S_1. We first deal with the case of ω in ADV, and then we make changes in $\mathcal{F}$ to maintain invariant (P) in S_1.

Removing the pair with ω if ω is in ADV. If $\omega \in \mathsf{ADV}$, then $F(\omega)$ is defined. We remove $F(\omega)$ from $\mathcal{F}$ and replace ω by a shadow packet of the same weight ω^t in ADV, which is placed in the former slot of ω in ADV. As ω is not in $\mathsf{ADV} \cap Q$, the potential increases by ω^t/ϕ. On the other hand, removing $F(\omega)$ from $\mathcal{F}$ decreases Ψ by $F(\omega)^t/\phi$. By Lemma 2.22(c) $F(\omega)^t < \omega^t$, so the overall change of the potential is positive. By Lemma 2.24 these removals do not affect other pairs and preserve invariant (P) in all segments.

Maintaining invariant (P) in S_1. By Lemma 2.18 the value of $\mathsf{pslack}^t(\tau)$ decreases by one for $\tau < d_\omega$, thus for such τ we need to decrease $\#\mathsf{pairs}(\tau)$ if $\#\mathsf{pairs}(\tau) > 0$. If f_1, the earliest-deadline packet in $\mathcal{F}$, has deadline at least d_ω, then we do nothing. Otherwise, we replace f_1 by ω in $\mathcal{F}$, which increases the potential by $(\omega^t - f_1^t)/\phi \geq 0$. As these are all changes done when processing S_1, we get $\Delta_{S_1}\Psi \geq 0$ and (2.11) holds.

We show that invariant (P) holds for slots in S_1. If $d_{f_1} \geq d_\omega$, then there is clearly no positive pair containing a slot before d_ω and no pair changes, thus invariant (P) is maintained. Otherwise, note that after replacing f_1 by ω in $\mathcal{F}$, the value of $\#\mathsf{pairs}(\tau)$ decreases by one for $\tau \in [d_{f_1}, d_\omega)$ and for other slots it remains the same. Moreover, $\#\mathsf{pairs}(\tau) \leq 0$ for $\tau < d_{f_1}$. Hence invariant (P) holds for slots in S_1 and it is preserved in any segment after S_1.

No packet from $\mathcal{F}$ expires. We claim that after processing S_1, no packet in $\mathcal{F}$ expires in the current step. This holds by Lemma 2.23 if we removed f_1 from $\mathcal{F}$. Otherwise, $d_{f_1} \geq d_\omega$, thus the claim trivially holds if $d_\omega > t$. In the remaining case, $d_\omega = t$, which implies that S_1 consists of just a single slot t. Note that f_1 is in a pair with g_1 and $d_{g_1} > t$, since after the adversary step, there is no packet in ADV in slot t. However, invariant (P) implies that g_1 needs to be in S_1, which is a contradiction. Hence the claim holds.

Processing segments in $(\delta, \gamma]$. Similarly to the analysis of a greedy step, in most cases we make the following changes in ADV and $\mathcal{F}$: (i) in ADV we replace some packet g with $g^t \leq p^t$ by a shadow packet, and (ii) from $\mathcal{F}$ we remove some packet f of weight at most ϱ^t. Of course, things get more complicated in an iterated leap step, where we possibly do several changes in ADV and $\mathcal{F}$.

In each of the cases below, depending on whether the leap step is simple or iterated, our goal is to show that (2.12) holds and that invariant (P) is maintained.

Case L.1: Simple leap step, i.e., $k = 0$ and $d_\varrho \leq \mathsf{nextts}^t(d_p)$. In this case, d_ϱ and d_p are in the same segment $(\delta, \gamma]$, i.e., $\mathsf{nextts}^t(d_\varrho) = \mathsf{nextts}^t(d_p) = \gamma$. We have $H = \emptyset$ and $\Delta^t w(H) = 0$. There are two subcases, depending on whether some changes are needed or not. We remark that it may happen that ϱ was in $\mathcal{F}$, but we have already removed it and thus $\varrho \notin \mathcal{F}$ now (in such a case, either ϱ was in a pair with ω, i.e., $\varrho = F(\omega)$, or ϱ was in a pair with j, i.e., $\varrho = F(j)$).

<u>Case L.1.A</u>: First, suppose that $\varrho \notin \mathcal{F}$ and there exists no packet in $\mathsf{ADV} \cap P$ with deadline in $(\delta, \gamma]$; in particular $p \notin \mathsf{ADV} \cap P$. Then g remains undefined, we do not further change the set $\mathcal{F}$ or ADV, and the pairs remain the same as well. Observe that there was no positive (or neutral) pair containing a slot in $(\delta, \gamma]$, as such a pair (f', g') satisfies $d^t_{g'} \in (\delta, \gamma]$ by invariant (P). Hence, $\#\mathsf{pairs}(\tau) \leq 0$ for $\tau \in (\delta, \gamma]$, thus invariant (P) holds for slots in $(\delta, \gamma]$ and it is clearly preserved for a slot outside $(\delta, \gamma]$.

Since we do no changes in ADV and $\mathcal{F}$, we have $\Delta_{(\delta,\gamma]}\Psi = 0$, $\mathsf{advgain}^t_{(\delta,\gamma]} = 0$, and the left-hand side of (2.12) is zero. As the right-hand side is non-positive, (2.12) holds.

<u>Case L.1.B</u>: Otherwise, either $\varrho \in \mathcal{F}$ or there is a packet in $\mathsf{ADV} \cap P$ with deadline in $(\delta, \gamma]$. In this case, ADV and $\mathcal{F}$ will be changed to maintain invariant (P).

Changes in case L.1.B. We now define packets g and f as follows.

Let g^* be the latest-deadline packet in $\mathsf{ADV} \cap P$ with deadline not exceeding γ. We note that g^* is well defined. This is trivially true if the second condition of the case is satisfied. If $\varrho \in \mathcal{F}$ then $F^{-1}(\varrho)$ is a candidate, because $d_\varrho \leq \gamma$ and thus $d_{F^{-1}(\varrho)} \leq \gamma$ by invariant (P). (It is possible that $d_{g^*} \leq \delta$ in this case.) If $p \in \mathsf{ADV}$, let $g = p$; otherwise let $g = g^*$.

Similarly, let f^* be the earliest-deadline packet in $\mathcal{F}$ with deadline larger than δ. This f^* is also well-defined, because either $\varrho \in \mathcal{F}$, in which case ϱ is a candidate, or $d_{g^*} \in (\delta, \gamma]$, in which case $F(g^*)$ is a candidate by Lemma 2.22(a). (It is possible that $d_{f^*} > \gamma$.) If $\varrho \in \mathcal{F}$, let $f = \varrho$; otherwise let $f = f^*$.

We remove f from $\mathcal{F}$ and we replace g in ADV by a new shadow packet s of weight $\mu = \mathsf{minwt}^t(d_p)$, added to the slot of g in ADV. It follows that g is no longer in $\mathsf{ADV} \cap P$.

Calculation in case L.1.B. Note that $f^t \leq \varrho^t$ as $d_f > \delta$ and as ϱ is the heaviest pending packet not in P with deadline after δ. Furthermore, $g^t \leq p^t$ as $w(\mathsf{sub}(P, g)) \geq \varrho^t$ and thus if $g^t > p^t$, the algorithm schedules g instead of p. In this case, removing packet f from $\mathcal{F}$ and replacing g in ADV cause the following change:

$$\Delta_{(\delta,\gamma]}\Psi - \mathsf{advgain}^t_{(\delta,\gamma]} = -\frac{f^t}{\phi} + \frac{g^t}{\phi} - (g^t - \mu) = -\frac{f^t}{\phi} - \frac{g^t}{\phi^2} + \mu \geq -\frac{\varrho^t}{\phi} - \frac{p^t}{\phi^2} + \mu\,, \quad (2.14)$$

which shows (2.12).

Invariant (P) in case L.1.B. We claim that after we remove f from $\mathcal{F}$ and g from ADV, invariant (P) holds for slots in $(\delta, \gamma]$. Moreover, we show that these removals do not violate invariant (P) for a slot outside $(\delta, \gamma]$, namely, for $\tau \notin (\delta, \gamma]$ either $\#\mathsf{pairs}(\tau)$ does not change, or $\#\mathsf{pairs}(\tau) \leq 0$ after the removals.

Recall that by Lemma 2.18 the value of $\mathsf{pslack}(\tau)$ increases by one for $\tau \in [d_p, d_\varrho)$ if $d_p < d_\varrho$, and decreases by one for $\tau \in [d_\varrho, d_p)$ if $d_\varrho < d_p$, while for other slots in $(\delta, \gamma]$ it remains the same. If $d_f < d_g$, then $\#\mathsf{pairs}(\tau)$ decreases by one for $\tau \in [d_f, d_g)$. Otherwise, $d_f \geq d_g$ and $\#\mathsf{pairs}(\tau)$ increases by one for $\tau \in [d_g, d_f)$. For other slots, $\#\mathsf{pairs}(\tau)$ remains the same.

From the definitions of f^*, g^*, and invariant (P), we have that $\#\mathsf{pairs}(\tau) \leq 0$ holds for $\tau \in (\delta, d_{f^*}) \cup [d_{g^*}, \gamma]$, even after the step. Thus the claim holds in this range, which includes slots outside $(\delta, \gamma]$ if $d_{f^*} > \gamma$ or if $d_{g^*} \leq \delta$ (note that either of the two conditions implies $d_{f^*} > d_{g^*}$). So for the rest of the proof we can assume that $d_{f^*} < d_{g^*}$ and that $\tau \in [d_{f^*}, d_{g^*})$.

Moreover, $\delta < d_{f^*} < d_{g^*} \leq \gamma$ together with $d_\varrho \in (\delta, \gamma]$ and $d_p \in (\delta, \gamma]$ implies that $\#\mathsf{pairs}(\tau)$ remains the same for a slot $\tau \notin (\delta, \gamma]$ in this case. It follows that invariant (P) is not violated for such τ by removing f from $\mathcal{F}$ and g from ADV.

We only need to consider the case when either $\#\mathsf{pairs}(\tau)$ increases or $\mathsf{pslack}(\tau)$ decreases, because in other cases, the inequality $\mathsf{pslack}(\tau) \geq \#\mathsf{pairs}(\tau)$ is preserved after the step as these quantities change by at most 1. The first case, when $\#\mathsf{pairs}(\tau)$ increases, happens when $d_g < d_f$ and $\tau \in [d_g, d_f)$. Since also $\tau \in [d_{f^*}, d_{g^*})$, this gives us that $g \neq g^*$ and $f \neq f^*$. Therefore $g = p$ and $f = \varrho$, which means that $\mathsf{pslack}(\tau)$ also increases.

The second case, when $\mathsf{pslack}(\tau)$ decreases, happens when $d_\varrho < d_p$ and $\tau \in [d_\varrho, d_p)$. This, combined with $\tau \in [d_{f^*}, d_{g^*})$, implies that $\tau \in [\max(d_\varrho, d_{f^*}), \min(d_p, d_{g^*}))$. As $f \in \{f^*, \varrho\}$ and $g \in \{g^*, p\}$, this implies that $\tau \in [d_f, d_g)$, so $\#\mathsf{pairs}(\tau)$ decreases as well.

Case L.2: Iterated leap step, i.e., $k > 0$ and $d_\varrho > \mathsf{nextts}^t(d_p)$. Let $h_0 = p, h_1, \ldots, h_k$ be as in the algorithm and let $h_{k+1} = \varrho$. Recall that any h_i for $i \leq k$ is in P by definition. First, we give a reason, why the algorithm does such a complicated shift of packets $h_1, \ldots, h_k$ in this case (the shift is implemented by the decrease of the deadlines).

Intuition on the shift of h_i's. Recall that without the shift, the new plan Q would have a (long) segment $(\delta, \gamma]$, consisting of several segments of P, and then the monotonicity of minwt for a fixed slot does not hold.

There is actually a simpler way how to avoid merging the segments: Let h' be the heaviest packet in P in the segment ending at $\gamma = \mathsf{nextts}^t(d^t_\varrho)$; note that this choice of h' is equivalent to the definition of h_k in the algorithm by Lemma 2.18(c). Then we just set the new deadline of h' to $\tau_0 = \mathsf{nextts}^t(d^t_p)$, and we get the same changes of tight slots as in Lemma 2.18(e); in particular, we avoid merging segments. Increasing the weight of h' still works similarly — we set its new weight to $\mathsf{minwt}^t(d^t_p)$ if it is below that. Since the simpler algorithm also increases the weight of the substitute packet ϱ to $\mathsf{minwt}^t(d^t_\varrho)$, it is possible to prove that the slot-monotonicity property holds for it.

However, the following breaks down in the analysis: Suppose that $\varrho \in \mathcal{F}$ is in a pair with a packet $g \in \mathsf{ADV} \cap P$ with deadline *before* the segment ending at γ and after the segment containing d^t_p. Moreover, g is heavier than h'. As ϱ gets into the plan, we need to remove it from $\mathcal{F}$ and then there may be no way how to create a new pair for g, while preserving invariant (P). Moreover, changing g into a shadow packet would cost too much even though $g^t \le p^t$, especially if $p \in \mathsf{ADV}$ or if the algorithm increases the weight of h'. Therefore, maintaining invariant (P) is impossible or too costly in some cases.

PlanM avoids this problem by choosing g (or a heavier packet with deadline at most γ) as h_1. If h_1 has deadline before the segment of P ending at γ, we need to iterate the choice of the packet to shift, yielding the iterative definition of h_i's. (Similar reasons as above show that if we choose h_2 as the heaviest packet in the segment ending at γ, instead of in any segment in $(\tau_1, \gamma]$, then the analysis does not work.)

We proceed similarly to case L.1 and split the analysis according to whether a change of the pairs is needed or not. For $i = 0, \ldots, k$, let $\mu_i = \mathsf{minwt}^t(d^t_{h_i})$; note that $\omega \ge \mu_0 \ge \mu_1 \ge \cdots \ge \mu_k = \mu = \mathsf{minwt}^t(d^t_\varrho)$ and the algorithm ensures that $h^{t+1}_i = w^{t+1}_{h_i} \ge \mu_{i-1}$ for $i = 1, \ldots, k+1$. In both cases below, we use the following simple bound on the change of weights of h_i's.

Lemma 2.25. *For any $1 \le a' \le b' \le k$, let $\Delta^t(w_{h_{a'}}, \ldots, w_{h_{b'}})$ be the total amount by which the algorithm increases the weights of packets $h_{a'}, \ldots, h_{b'}$. Suppose that there exists $i \in [a', b']$ such that $h^t_i < \mu_{i-1}$, i.e., the algorithm increases the weight of h_i. Then $\Delta^t(w_{h_{a'}}, \ldots, w_{h_{b'}}) \le \mu_{a'-1} - h^t_{b'}$.*

Proof. Let c be the largest $c' \in [a', b']$ such that $h^t_{c'} < \mu_{c'-1}$; such c exists by the assumption of the lemma. We show the claim as follows:

$$\begin{aligned}
\Delta^t(w_{h_{a'}}, \ldots, w_{h_{b'}}) &= \sum_{i=a'}^{c} (\max(\mu_{i-1}, h^t_i) - h^t_i) \\
&= \sum_{i=a'}^{c} \max(\mu_{i-1} - h^t_i, 0) \\
&\le \sum_{i=a'}^{c-1} \max(\mu_{i-1} - \mu_i, 0) + \max(\mu_{c-1} - h^t_c, 0) \qquad (2.15) \\
&= \sum_{i=a'}^{c-1} (\mu_{i-1} - \mu_i) + \mu_{c-1} - h^t_c \qquad (2.16) \\
&= \mu_{a'-1} - h^t_c \\
&\le \mu_{a'-1} - h^t_{b'} \,. \qquad (2.17)
\end{aligned}$$

where inequality (2.15) follows from $h_i^t \geq \mu_i$, equation (2.16) from $\mu_{i-1} \geq \mu_i$ and from $\mu_{c-1} > h_c^t$ (by the choice of c), and inequality (2.17) from $h_c^t \geq h_{b'}^t$ as $c \leq b'$. □

Case L.2.A: First, suppose that $\varrho \notin \mathcal{F}$ and there exists no packet in $\mathsf{ADV} \cap P$ with deadline in $(\delta, \gamma]$. Then f and g remain undefined, we do not further change the adversary schedule ADV or set $\mathcal{F}$, and the pairs remain the same as well. From the case condition, no h_i, $i = 0, \ldots, k$, is in ADV, because each h_i is in P and its deadline is in $(\delta, \gamma]$. Also note that invariant (P) holds for slots in $(\delta, \gamma]$ as there is no positive pair containing a slot in $(\delta, \gamma]$ and that invariant (P) is preserved for a slot outside $(\delta, \gamma]$ as we do not change the pairs.

It remains to show (2.12). Since we have not changed ADV, we have $\mathsf{advgain}^t_{(\delta,\gamma]} = 0$. Next, we claim $\Delta^t w(H) \leq \mu_0 - \mu_k$. If there is no $i \in [1, k]$ such that $h_i^t < \mu_{i-1}$, then $\Delta^t w(H) = 0 \leq \mu_0 - \mu_k$ as $\mu_0 \geq \mu_k$. Otherwise, we use Lemma 2.25 with $a' = 1$ and $b' = k$ to get $\Delta^t w(H) \leq \mu_0 - h_k^t$, which is a stronger upper bound than $\mu_0 - \mu_k$ as $h_k^t \geq \mu_k$.

In this case, the potential change $\Delta_{(\delta,\gamma]}\Psi$ only reflects the increase of weights of h_i's, that is, $\Delta_{(\delta,\gamma]}\Psi = \Delta_H w(P)/\phi$, where $\Delta_H w(P)$ is the change of the plan's weight due to increasing the weights of h_i's. Since all h_i's, for $i = 1, \ldots, k$, are both in P and in Q (by Lemma 2.18(a)), we have $\Delta_H w(P) = \Delta^t w(H)$, and therefore $\Delta_H w(P)/\phi - \phi \cdot \Delta^t w(H) = -\Delta^t w(H)$. Then (2.12) follows from the above bound on $\Delta^t w(H)$ and an easy calculation:

$$\begin{aligned} \Delta_{(\delta,\gamma]}\Psi - \phi \cdot \Delta^t w(H) - \mathsf{advgain}^t_{(\delta,\gamma]} &= \frac{\Delta_H w(P)}{\phi} - \phi \cdot \Delta^t w(H) - 0 \\ &= -\Delta^t w(H) \\ &\geq -\mu_0 + \mu_k \\ &\geq -\frac{p^t}{\phi^2} - \frac{\varrho^t}{\phi} + \mu_k \,, \end{aligned} \tag{2.18}$$

where inequality (2.18) follows from $\mu_0 \leq p^t/\phi^2 + \varrho^t/\phi$ by (2.9).

Case L.2.B: Otherwise, $\varrho \in \mathcal{F}$ or there exists a packet in $\mathsf{ADV} \cap P$ with deadline in $(\delta, \gamma]$. ADV and $\mathcal{F}$ need to be changed, for which we first define a packet g. For $i = 0, \ldots, k$, let S'_i be the segment of P containing $d^t_{h_i}$, which ends at $\tau_i = \mathsf{nextts}^t(d^t_{h_i})$. (See Figure 2.9.)

Let g^* be the latest-deadline packet in $\mathsf{ADV} \cap P$ with deadline at most γ; it exists, otherwise the previous case applies — in particular, if there is no packet in $\mathsf{ADV} \cap P$ with deadline in $(\delta, \gamma]$, then $\varrho \in \mathcal{F}$ and the packet $F^{-1}(\varrho)$ from $\mathsf{ADV} \cap P$ in the pair with ϱ is a candidate for g as $\gamma \geq d_\varrho > \mathsf{prevts}^t(d_{F^{-1}(\varrho)})$ by Lemma 2.22(a). If $d^t_{g^*}$ is in a segment S'_i for some i and $h_i \in \mathsf{ADV}$, then let $g = h_i$; otherwise, let $g = g^*$. Note that if $h_k \in \mathsf{ADV}$, then $g = h_k$.

We will process segments S'_i in groups, where a group is an interval of indexes $1, \ldots, k$ of segments S'_i. Intuitively, we have a group for each $h_i \in \mathsf{ADV}$, which needs to be replaced in ADV as its deadline was decreased, a special last group, and possibly a special group at the beginning. Let $i_1 < i_2 < \cdots < i_\ell$ be the indexes of those h_i's ($i = 0, \ldots, k$) that are in ADV. Note that if $g = h_i$ for some i, then $i = i_\ell$ by the definitions of g^* and g.

First, suppose that $\ell > 0$. If $\ell > 1$, then for each $a = 1, \ldots, \ell - 1$, the interval $[i_a, i_{a+1} - 1]$ is a *middle group*. If $i_1 > 0$, meaning that $h_0 = p \notin \mathsf{ADV}$, then there is a special *initial group* $[0, i_1 - 1]$; it does not exist if $i_1 = 0$. Next, we assign the indexes in $[i_\ell, k]$ to one or two groups. If $g = h_{i_\ell}$ (in particular, if $i_\ell = k$), then $[i_\ell, k]$ is the *last group*. Otherwise, let α be the smallest i such that $\tau_i \geq d^t_g$. Then $[\alpha, k]$ is the *last group* and the interval $[i_\ell, \alpha - 1]$ is a new middle group; note that the new middle group is non-empty, since $\alpha = i_\ell$ implies $g = h_{i_\ell}$, thus $\alpha > i_\ell$. See Figure 2.13 for an illustration.

Otherwise, $\ell = 0$ (thus no h_i is in ADV) and then there are only at most two groups. There is the last group $[\alpha, k]$, where α is again the smallest $i \geq 0$ with $\tau_i \geq d^t_g$, and if $\alpha > 0$, we also have the initial group $[0, \alpha - 1]$.

Note that in each group $[a, b]$, packet h_a is in ADV, unless it is the initial group or it is the last group and $g \neq h_{i_\ell}$. Packets $h_{a+1}, \ldots, h_b$ are not in ADV for any group $[a, b]$.

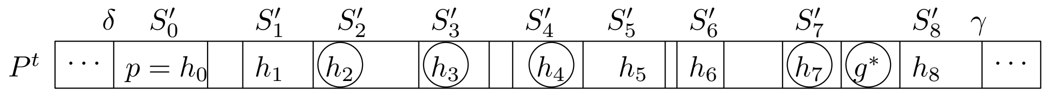


Figure 2.13: An example of creating the groups with $k = 8$. The circled packets are in ADV. Thus $[0, 1]$ is the initial group, $[2, 2]$, $[3, 3]$, $[4, 6]$, and $[7, 7]$ are the middle groups, and $[8, 8]$ is the last group.

To show (2.12), we split the cost of the changes and the adversary credit for replacing packets in ADV among groups in a natural way. Namely, for a group $[a, b]$, let $\Delta_{[a,b]}\Psi$ be the total change of the potential due to changes done when processing group $[a, b]$, let $\mathsf{advgain}^t_{[a,b]}$ be the adversary credit for the changes of ADV in segments $S'_a, \ldots, S'_b$, and let $\Delta^t(w_{h_{a+1}}, \ldots, w_{h_{b+1}})$ be the total amount by which the algorithm increases the weights of $h_{a+1}, \ldots, h_{b+1}$. Our goal is to prove that for each middle group $[a, b]$ and for the possible initial group $[a, b]$ (which has $a = 0$) it holds

$$\Delta_{[a,b]}\Psi - \phi \cdot \Delta^t(w_{h_{a+1}}, \ldots, w_{h_{b+1}}) - \mathsf{advgain}^t_{[a,b]} \geq -\frac{h^t_a}{\phi^2} + \frac{h^t_{b+1}}{\phi^2} . \qquad (2.19)$$

Similarly, for the last group $[a, k]$, which is defined in all cases, we show

$$\Delta_{[a,k]}\Psi - \phi \cdot \Delta^t(w_{h_{a+1}}, \ldots, w_{h_k}) - \mathsf{advgain}^t_{[a,k]} \geq -\frac{h^t_a}{\phi^2} - \frac{\varrho^t}{\phi} + \mu_k . \qquad (2.20)$$

(Note that the right-hand side of (2.20) may be positive.) The sum of (2.19) over all middle groups and the possible initial group plus (2.20) equals exactly (2.12).

We process the groups in the reverse order of time, i.e., from the last one to the first one, which may be of any type. After processing a group $[a, b]$, we maintain that h_a is not in ADV (even though it may have been in ADV beforehand).

Regarding invariant (P), we divide the changes of pslack in $(\delta, \gamma]$ into groups in a natural way. After processing a group $[a, b]$, we show that invariant (P) holds for slots in segments $S'_a, \ldots, S'_b$ and that invariant (P) is not violated for other slots, i.e., for a slot τ not in one of the segments in the group either $\#\mathsf{pairs}(\tau)$ does not increase, or $\#\mathsf{pairs}(\tau) \leq 0$ after processing the group. After we process all groups, invariant (P) holds for any slot, since by Lemma 2.18(d) the value of pslack changes only for slots in the first segment S_1 of P and in any of the segments $S'_0, \ldots, S'_k$ and since we have already shown that invariant (P) holds for S_1.

Last group. Let $[a, k]$ be the interval of indexes corresponding to the last group.

Let f^* be the earliest-deadline packet in $\mathcal{F}$ with deadline after δ. Such an f^* clearly exists if $\varrho \in \mathcal{F}$; otherwise, there exists a packet g' in $\mathsf{ADV} \cap P$ with deadline in $(\delta, \gamma]$ and the packet $F(g')$ is a candidate for f^* as $d_{F(g')} > \mathsf{prevts}^t(d^t_{g'}) \geq \delta$ by Lemma 2.22(a). Note that $f^{*t} \leq \varrho^t$ by the definition of ϱ.

If $\varrho \in \mathcal{F}$, let $f = \varrho$; otherwise, let $f = f^*$. We remove f from $\mathcal{F}$ and replace g in ADV by a new shadow packet s of weight $\mathsf{minwt}^t(d^t_g)$, which we add to the slot of g in ADV.

Calculation showing (2.20) *for the last group.* Apart from the changes in the paragraph above, we need to take into account the possible change of weights of $h_{a+1}, \dots, h_k$, which also increases the weight of the plan as any h_i is both in the old plan P and in the new plan Q by Lemma 2.18(a). (Increasing the weight of $\varrho = h_{k+1}$ has already been taken into account in (2.13).) We show $\Delta^t(w_{h_{a+1}}, \dots, w_{h_k}) \le \mu_a - \mu_k$. If there is no $i \in [a+1, k]$ such that $h^t_i < \mu_{i-1}$, then $\Delta^t(w_{h_{a+1}}, \dots, w_{h_k}) = 0 \le \mu_a - \mu_k$ as $\mu_a \ge \mu_k$. Otherwise, we use Lemma 2.25 with $a' = a+1$ and $b' = k$ to get $\Delta^t(w_{h_{a+1}}, \dots, w_{h_k}) \le \mu_a - h^t_k \le \mu_a - \mu_k$ by $h^t_k \ge \mu_k$.

The key observation in this case is that $g^t \le h^t_a$. This is trivial if $g = h_{i_\ell}$ and thus $i_\ell = a$. Otherwise, recall that $a = \alpha > i_\ell$ is the smallest i with $\tau_i \ge d^t_g$ and that h_a is the heaviest packet in plan P after τ_{a-1} and till γ. As g was in $\mathsf{ADV} \cap P$ and as $d^t_g \in (\tau_{a-1}, \gamma]$ by the definition of $a = \alpha$, we get that $g^t \le h^t_a$. Since $\tau_a \ge d^t_g$, we get $\mathsf{minwt}^t(d^t_g) \ge \mu_a$.

Thus for the changes in segments $S'_a, S'_{a+1}, \dots, S'_k$ it holds

$$\begin{aligned}
&\Delta_{[a,k]}\Psi - \phi \cdot \Delta^t(w_{h_{a+1}}, \dots, w_{h_k}) - \mathsf{advgain}^t_{[a,k]} \\
&\quad = \left(\frac{\Delta^t(w_{h_{a+1}}, \dots, w_{h_k})}{\phi} - \frac{f^t}{\phi} + \frac{g^t}{\phi} \right) - \phi \cdot \Delta^t(w_{h_{a+1}}, \dots, w_{h_k}) \\
&\qquad\quad - (g^t - \mathsf{minwt}^t(d^t_g)) \\
&\quad = -\frac{g^t}{\phi^2} - \frac{f^t}{\phi} - \Delta^t(w_{h_{a+1}}, \dots, w_{h_k}) + \mathsf{minwt}^t(d^t_g) \\
&\quad \ge -\frac{h^t_a}{\phi^2} - \frac{\varrho^t}{\phi} - (\mu_a - \mu_k) + \mu_a = -\frac{h^t_a}{\phi^2} - \frac{\varrho^t}{\phi} + \mu_k\,,
\end{aligned}$$

which shows (2.20).

Invariant (P) after processing the last group. We claim that after we process the last group, invariant (P) holds for slots in segments $S'_a, S'_{a+1}, \dots, S'_k$ and invariant (P) is not violated for another slot. (The proof is similar to the one in case L.1.B.)

Recall that by Lemma 2.18(d) the value of $\mathsf{pslack}(\tau)$ increases by one for $\tau \in [d^t_{h_i}, \tau_i)$, $i = a, \dots, k-1$. Moreover, if $d^t_{h_k} < d_\varrho$, the value of $\mathsf{pslack}(\tau)$ increases by one for $\tau \in [d^t_{h_k}, d_\varrho)$, and otherwise, it decreases by one for $\tau \in [d_\varrho, d^t_{h_k})$. For other slots in $(\delta, \gamma]$, the value of $\mathsf{pslack}(\tau)$ remains the same (or we take its change into account when processing the group that contains τ).

If $d_f < d_g$, then $\#\mathsf{pairs}(\tau)$ decreases by one for $\tau \in [d_f, d_g)$ and for other slots it remains the same. Otherwise, $d_f \ge d_g$ and $\#\mathsf{pairs}(\tau)$ increases by one for $\tau \in [d_g, d_f)$, while for other slots it does not change.

From the definitions of f^*, g^*, and invariant (P), we have that $\#\mathsf{pairs}(\tau) \le 0$ holds for $\tau \in (\delta, d_{f^*}) \cup [d_{g^*}, \gamma]$, even after the step. Thus the claim holds in this range, which includes slots outside $(\delta, \gamma]$ if $d_{f^*} > \gamma$ or if $d_{g^*} \le \delta$ (note that either of the two conditions implies $d_{f^*} > d_{g^*}$). So for the rest of the proof we can assume that $d_{f^*} < d_{g^*}$ and that $\tau \in [d_{f^*}, d_{g^*})$.

We only need to consider the case when either $\#\mathsf{pairs}(\tau)$ increases or $\mathsf{pslack}(\tau)$ decreases, because in other cases, the inequality $\mathsf{pslack}(\tau) \ge \#\mathsf{pairs}(\tau)$ is preserved after the step as these quantities change by at most 1.

The first case, when $\#\mathsf{pairs}(\tau)$ increases, happens when $d_g < d_f$ and $\tau \in [d_g, d_f)$. Since also $\tau \in [d_{f^*}, d_{g^*})$, this gives us that $g \ne g^*$ and $f \ne f^*$. Therefore $g = h_a$ and $f = \varrho$. If $a = k$, then $\mathsf{pslack}(\tau)$ also increases. Otherwise, $a < k$ and $d_{g^*} \le \tau_a$ by the definitions of g^*, g, and a, thus $\tau \in [d^t_{h_a}, \tau_a)$, which implies that $\mathsf{pslack}(\tau)$ increases as well.

The second case, when $\mathsf{pslack}(\tau)$ decreases, happens when $d_\varrho < d^t_{h_k}$ and $\tau \in [d_\varrho, d_{h_k})$. Since also $\tau \in [d_{f^*}, d_{g^*})$, we get $\tau \in [\max(d_\varrho, d_{f^*}), \min(d^t_{h_k}, d_{g^*}))$. Thus we only need to consider that case when d_{g^*} is in segment S'_k (that contains both d_ϱ and $d^t_{h_k}$), which implies $a = k$ and $g = h_k$. As $f \in \{f^*, \varrho\}$, we have that $\tau \in [d_f, d_g)$, so $\#\mathsf{pairs}(\tau)$ decreases as well. Thus the claim holds.

Middle group. Let $[a, b]$ be a middle group; recall that $h_a \in \mathsf{ADV}$. We have two subcases.

Case M.i: There is $i \in [a, b]$ such that $h^t_{i+1} < \mu_i$, i.e., the algorithm increases the weight of h_{i+1}. Let $F(h_a)$ be the packet that is in a pair with h_a. We remove $F(h_a)$ from $\mathcal{F}$ and replace h_a in ADV by a new shadow packet s of weight $\mu_a = \mathsf{minwt}^t(\tau_a)$, added to the slot of h_a in ADV.

Calculation showing (2.19) *in case M.i.* We take into account the possible change of weights of $h_{a+1}, \dots, h_{b+1}$. By the case condition, there is $i \in [a, b]$ such that $h^t_{i+1} < \mu_i$, we thus use Lemma 2.25 with $a' = a + 1$ and $b' = b + 1$ to get $\Delta^t(w_{h_{a+1}}, \dots, w_{h_{b+1}}) \le \mu_a - h^t_{b+1}$.

Next, observe that $F(h_a)^t \le h^t_{b+1}$. Indeed, $F(h_a)$ is not in P and it was in a pair with h_a, thus $d^t_{F(h_a)} > \mathsf{prevts}^t(d^t_{h_a}) \ge \delta$ by Lemma 2.22(a). Thus $F(h_a)^t \le \varrho^t$ as ϱ is the heaviest pending packet not in P with deadline after δ. By Lemma 2.18(b) $\varrho^t \le h^t_{b+1}$, which implies $F(h_a)^t \le h^t_{b+1}$.

Then we prove (2.19) as follows:

$$\begin{aligned}
&\Delta_{[a,b]}\Psi - \phi \cdot \Delta^t(w_{h_{a+1}}, \dots, w_{h_{b+1}}) - \mathsf{advgain}^t_{[a,b]} \\
&\quad = \left(\frac{\Delta^t(w_{h_{a+1}}, \dots, w_{h_{b+1}})}{\phi} - \frac{F(h_a)^t}{\phi} + \frac{h^t_a}{\phi} \right) - \phi \cdot \Delta^t(w_{h_{a+1}}, \dots, w_{h_{b+1}}) \\
&\qquad - (h^t_a - \mu_a) \\
&\quad = -\frac{F(h_a)^t}{\phi} - \frac{h^t_a}{\phi^2} - \Delta^t(w_{h_{a+1}}, \dots, w_{h_{b+1}}) + \mu_a \\
&\quad \ge -\frac{F(h_a)^t}{\phi} - \frac{h^t_a}{\phi^2} - (\mu_a - h^t_{b+1}) + \mu_a \\
&\quad \ge -\frac{h^t_a}{\phi^2} + \frac{h^t_{b+1}}{\phi^2},
\end{aligned}$$

where the last inequality follows from $F(h_a)^t \le h^t_{b+1}$.

Invariant (P) in case M.i. We claim that after we process the middle group, invariant (P) holds for slots in segments $S'_a, S'_{a+1}, \dots, S'_b$ and it is not violated for another slot. Note that as $b < k$, Lemma 2.18(d) shows that the value of pslack remains the same or increases for slots in segments $S'_a, S'_{a+1}, \dots, S'_b$ (recall that changes of pslack values in another segment are taken into account when we process the group containing that segment). We use Lemma 2.24 to analyze how the values of $\#\mathsf{pairs}$ change. If $d^t_{F(h_a)} \le d^t_{h_a}$, then $\#\mathsf{pairs}(\tau)$ decreases by one for $\tau \in [d^t_{F(h_a)}, d^t_{h_a})$. Otherwise, $d^t_{F(h_a)} > d^t_{h_a}$ and $\#\mathsf{pairs}(\tau) \le 0$ for $\tau \in [d^t_{h_a}, d^t_{F(h_a)})$. For other slots, $\#\mathsf{pairs}(\tau)$ remains the same. Hence, the claim holds.

Case M.ii: Otherwise, the algorithm does not increase the weight of h_{i+1} for any $i \in [a, b]$, i.e., $\Delta^t(w_{h_{a+1}}, \dots, w_{h_{b+1}}) = 0$. We replace h_a in ADV by h_{a+1}, i.e., we put h_{a+1} on the slot of h_a in ADV. Note that the new deadline of h_{a+1} is τ_a and the new slot of h_{a+1} in ADV is not after τ_a.

We claim that h_{a+1} is not in ADV before the replacement, therefore it is not twice in ADV after the replacement. This is trivial if $b > a$, since then packets $h_{a+1}, \dots, h_b$ are not in ADV before processing the groups. Otherwise, we have $a = b$. Recall that we are processing groups from the last one to the first one, thus the group containing index $a + 1$ is already processed. Furthermore, we maintain that after processing a group $[a', b']$, packet $h_{a'}$ is not in ADV, which shows the claim.

Calculation showing (2.19) *in case M.ii.* We bound the cost of changes in the middle group $[a, b]$ by

$$\begin{aligned}
\Delta_{[a,b]}\Psi - \phi \cdot \Delta^t(w_{h_{a+1}}, \dots, w_{h_{b+1}}) - \mathsf{advgain}^t_{[a,b]}& \\
&= \left(\frac{h^t_a}{\phi} - \frac{h^t_{a+1}}{\phi}\right) - \phi \cdot 0 - (h^t_a - h^t_{a+1}) \\
&= -\frac{h^t_a}{\phi^2} + \frac{h^t_{a+1}}{\phi^2} \\
&\geq -\frac{h^t_a}{\phi^2} + \frac{h^t_{b+1}}{\phi^2} ,
\end{aligned}$$

where that last inequality follows from $h^t_{a+1} \geq h^t_{b+1}$ as $a \leq b$. This shows (2.19).

Invariant (P) in case M.ii. We show that after we process the middle group, invariant (P) holds for slots in segments $S'_a, S'_{a+1}, \dots, S'_b$ and it is not violated for another slot. Note that $\#\mathsf{pairs}(\tau)$ increases by one for $\tau \in [d^t_{h_a}, \tau_a)$ as we replaced h_a by h_{a+1}. By Lemma 2.18(d) the value of $\mathsf{pslack}(\tau)$ increases by one for $\tau \in [d^t_{h_a}, \tau_a)$, thus invariant (P) holds for such τ. For other slots, the value of $\#\mathsf{pairs}$ stays the same and the value of pslack remains the same or increases by Lemma 2.18(d).

Initial group. If $i_1 > 0$ or if $\ell = 0$ and $\alpha > 0$, then there is the initial group $[0, b]$. Note that for any $i \in [0, b]$, $h_i \notin \mathsf{ADV}$. We do not change ADV or set $\mathcal{F}$, thus $\mathsf{advgain}^t_{[0,b]} = 0$ and $\#\mathsf{pairs}(\tau)$ remains the same for any slot τ. Invariant (P) holds for a slot $\tau \in S'_0 \cup \dots \cup S'_b$ as $b < k$ and as the value of pslack does not decrease by Lemma 2.18(d).

The only cost we need to calculate is that of changing the weights of $h_1, \dots, h_{b+1}$, denoted $\Delta^t(w_{h_1}, \dots, w_{h_{b+1}})$. First, suppose that the algorithm increases the weight of at least one of the packets $h_1, \dots, h_{b+1}$, i.e., there is $i \in [0, b]$ such that $h^t_{i+1} < \mu_i$. By Lemma 2.25 with $a' = 1$ and $b' = b + 1$ we have $\Delta^t(w_{h_1}, \dots, w_{h_{b+1}}) \leq \mu_0 - h^t_{b+1}$.

Then the calculation showing (2.19) is simple:

$$\begin{aligned}
\Delta_{[0,b]}\Psi - \phi \cdot \Delta^t(w_{h_1}, \dots, w_{h_{b+1}}) - \mathsf{advgain}^t_{[0,b]}& \\
&= \frac{1}{\phi}\Delta^t(w_{h_1}, \dots, w_{h_{b+1}}) - \phi \cdot \Delta^t(w_{h_1}, \dots, w_{h_{b+1}}) - 0 \\
&= -\Delta^t(w_{h_1}, \dots, w_{h_{b+1}}) \\
&\geq -\mu_0 + h^t_{b+1} \\
&\geq -\frac{p^t}{\phi^2} - \frac{\varrho^t}{\phi} + h^t_{b+1} \\
&\geq -\frac{p^t}{\phi^2} + \frac{h^t_{b+1}}{\phi^2} ,
\end{aligned}$$

where the penultimate inequality follows from $\mu_0 \leq p^t/\phi^2 + \varrho^t/\phi$ by (2.9) and the last inequality uses $\varrho^t \leq h^t_{b+1}$.

Otherwise, $\Delta^t(w_{h_1}, \dots, w_{h_{b+1}}) = 0$ and (2.19) holds, since its left-hand side is zero and the right-hand side is at most zero. This concludes the proof that the packet-scheduling inequality (2.4) holds in a leap step and also the proof of ϕ-competitiveness of Algorithm PlanM.

2.7 Algorithms with Lookahead

2.7.1 An Algorithm for 2-Bounded Instances with 1-Lookahead

In this section, we present an algorithm for 2-*bounded* Bounded-Delay Packet Scheduling *with 1-lookahead*, as defined in Section 2.1.

Consider some online algorithm $\mathcal{A}$ with 1-lookahead. 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 *canonical plan* in step t to be the canonical optimal schedule in the time interval $[t,\infty)$ that consists of pending and lookahead packets at time t. Note that this definition corresponds to the canonical realization of the plan as defined in Section 2.5. As in this section we work solely with the canonical plan, we refer to it just as the plan.

For 2-bounded instances, this plan uses slots t, $t+1$ and $t+2$ only. We will typically denote the packets in the plan scheduled in these slots by p_1, p_2, p_3, respectively. The canonical 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. A similar condition holds for p_2 and p_3.

Remark. Note that for general instances and higher lookahead, the structure of the plan gets more complicated. For example, if we define segments in the same way as in Section 2.5, not all packets from a segment can be scheduled in the first slot of the segment. Moreover, the front-adjusted (realization of the) plan is not unique and the definition of the substitute packet $\mathsf{sub}(P,p)$ needs to be changed to take into account also the release time of the substitute packet, which can be in future. However, in the 2-bounded case with 1-lookahead, the plan is quite simple and we actually do not need the tools from Section 2.5 to obtain our result.

Fix some parameter $\alpha > 1$. We now give a definition of our algorithm.

Pseudocode 5 Algorithm: CompareWithBias(α)

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

Note that if the algorithm schedules p_2, then p_1 must be expiring, as otherwise $w_{p_1} > w_{p_2}$ (by the 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 2.26. *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 the 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 assumption (A2) assures 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, relative to some time t (note that the notation in this section differs from the notation in other sections):

- e = packet scheduled at $t-1$ in ALG,
- f = packet scheduled at t in ALG,
- g = packet scheduled at $t+1$ in ALG,
- h = packet scheduled at $t+2$ in ALG,
- i = packet scheduled at $t-1$ in OPT,
- j = packet scheduled at t in OPT,
- k = packet scheduled at $t+1$ in OPT,
- ℓ = packet scheduled at $t+2$ in OPT.

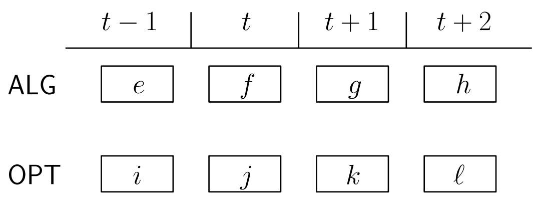


Figure 2.14: Packet definition.

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 $2\alpha \cdot w_j < w_f + w'_f$. 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.

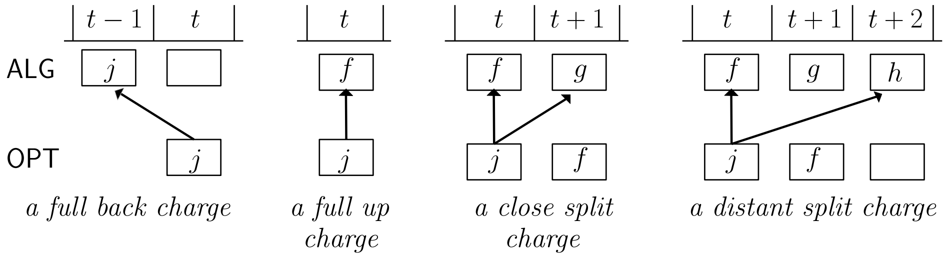


Figure 2.15: Full and split charges. Note that for split charges, f is scheduled in step $t+1$ in OPT, which follows from the fact that we do not charge j using a full up charge.

Parameters and constants. We set the parameter α and constants δ and R which we will use in the analysis so that they satisfy the following equalities:

$$2 - \delta - \frac{R - 1 + 2\delta}{\alpha} = R \tag{2.21}$$

$$1 - 2\delta + 2\alpha\delta = R \tag{2.22}$$

$$1 + \frac{1}{2\alpha} = R \tag{2.23}$$

By solving these equations we get $\alpha = \frac{1}{4}(\sqrt{13}+3) \approx 1.651$, $\delta = \frac{1}{6}(5-\sqrt{13}) \approx 0.232$, and $R = \frac{1}{2}(\sqrt{13}-1) \approx 1.303$. We will prove that the algorithm is R-competitive.

We also use the following properties of these constants:

$$2 - R - 3\delta = 0 \tag{2.24}$$

$$2 - R - 2\delta > 0 \tag{2.25}$$

$$1 - \delta - \frac{R - 1 + 2\delta}{2\alpha} > 0 \tag{2.26}$$

$$1 - \frac{R}{2\alpha} > 0 \tag{2.27}$$

$$3\alpha\delta < R \tag{2.28}$$

$$2 - \frac{R}{\alpha} < R \tag{2.29}$$

where (2.24) follows from (2.21) and (2.22) and the strict inequalities can be verified numerically.

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 receives 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.

Packet j scheduled in OPT at time t is charged according to the first rule below that applies. See Figures 2.15, and 2.16 for an illustration of different types of charges.

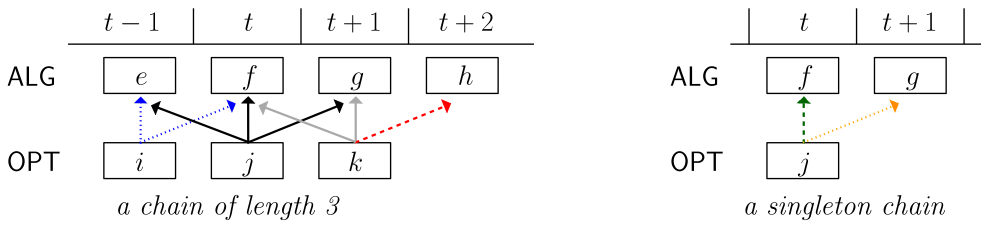


Figure 2.16: On the left, a chain of length 3 starting in step $t-1$ and ending in step $t+1$. The *chain beginning charges* are denoted by dotted (blue) lines, the *chain end charges* are denoted by gray lines and the *forward charge from a chain* is depicted by a dashed (red) arrow. Black arrows denote 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.

1. If j is scheduled in step $t-1$ in ALG (that is, $e = j$), 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.*

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

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 j 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 $\min(w_j, R \cdot w_f)$ to step t and $w_j - R \cdot w_f$ to step $t+1$ if $w_j > R \cdot w_f$. 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 $2\delta \cdot w_j$ to step t, and $(1-2\delta) \cdot w_j$ to step $t+1$. We call these charges *chain beginning charges.*

   (c) If step t is the last step of a long chain, charge $\delta \cdot w_j$ to step $t-1$, $(R-1+2\delta) \cdot w_f$ to step t, and $(1-\delta) \cdot w_j - (R-1+2\delta) \cdot w_f$ 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 $(1-\delta) \cdot w_j > (R-1+2\delta) \cdot w_f$, since $w_j > w_f$ and $1-\delta = R-1+2\delta$ which follows from (2.24).)

   (d) Otherwise, i.e., if step t is inside a long chain, charge $\delta \cdot w_j$ to step $t-1$, $\delta \cdot w_j$ to step t, and $(1-2\delta) \cdot w_j$ to step $t+1$. We call these charges *chain link charges.*

To estimate the competitive ratio we need to show that each step or a pair of steps does not receive too much charge. We start with a useful observation about plans of Algorithm CompareWithBias(α), that will be used multiple times in our proofs.

Lemma 2.27. *Consider a time t, where the algorithm has two pending packets a, b and a lookahead packet c with the following properties: $d_a = t$, $(r_b, d_b) = (t, t+1)$, $(r_c, d_c) = (t+1, t+2)$, and $w_a < \min(w_b, w_c)$. If the algorithm schedules packet a in step t, then the plan at time t is a, b, c, and $2\alpha \cdot w_a \geq w_b + w_c$.*

Proof. We claim that there is no pending or lookahead packet $q \notin \{b, c\}$ heavier than a. Suppose for a contradiction that such a q exists. Then a schedule containing packets q, b, c in some order is feasible and has larger profit than a, b, c. This implies that the plan does not contain a and thus a cannot be scheduled, contradicting the assumption of the lemma.

The schedule a, b, c is feasible and the claim above implies that it is optimal, thus it is the plan. It remains to show that $2\alpha \cdot w_a \geq w_b + w_c$, which follows easily by a contradiction: Otherwise $2\alpha \cdot w_a < w_b + w_c$ and CompareWithBias(α) would schedule b, contradicting the assumption of the lemma. □

Next, we will provide an analysis of full, split and chain charges, starting with full and split charges. We prove several lemmas from which the analysis follows. We fix some time slot t, and use the notation from Figure 2.14 for packets at time slots $t-1$, t, $t+1$ and $t+2$ in the schedule ALG of the algorithm and the optimal schedule OPT.

Analysis of full charges. Using Rules 1 and 2, if step t receives a full back charge, then the condition of Rule 2 guarantees that it will not receive a full up charge. This gives us the following observation.

Lemma 2.28. *Step t receives at most one full charge, i.e., a charge by Rule 1 or 2.*

Analysis of split charges. We now analyze close and distant split charges. The crucial property of split charges is that, similar to full charges, each step receives at most one split charge. Before we prove this, we establish several useful properties of split charges.

Lemma 2.29. *Let the plan at time t be p_1, p_2, p_3. If j is charged using a close or a distant split charge, then the following holds:*

(a) j is not scheduled by the algorithm in step $t-1$, i.e., j is pending for the algorithm in step t.

(b) $d_f = t+1$ and f is scheduled in step $t+1$ in OPT (that is, $k = f$). In particular, step t receives a full back charge.

(c) $d_j = t$ and $w_j \le w_{p_1}$.

(d) $p_2 = f$.

Proof. By Rule 1, packet j would be charged using a full back charge if it were scheduled in step $t-1$, implying (a). The case conditions for split charges in the charging scheme imply that OPT schedules f in step $t+1$ and $w_f > w_j$. Now (b) follows from the fact that we do not charge j using a full up charge.

To show (c), note that if j is not expiring, then j and f would have equal deadlines. As we also have $w_f > w_j$, f would be scheduled before j in OPT by the canonical ordering, a contradiction. The inequality $w_j \le w_{p_1}$ now follows from the definition of the plan.

It remains to prove (d). Towards contradiction, suppose that $f = p_1$. We know that j is expiring and thus it is not in the plan. If $d_{p_2} = t+1$ then the optimality of the plan implies $w_{p_2} > w_j$ (otherwise j, f, p_3 would be a better plan), so, since p_2 is not in OPT, we could improve OPT by scheduling f in step t and p_2 in step $t+1$.

Next, assume that $d_{p_2} = t+2$. The optimality of the plan implies that $w_{p_2} > w_j$ and $w_{p_3} > w_j$. Since both p_2, p_3 have deadline $t+2$, at least one of them is not scheduled in OPT. So OPT could be improved by scheduling f in step t and one of p_2 or p_3 in step $t+1$. In both cases we get a contradiction with the optimality of OPT. □

We show a useful lemma about a distant split charge from which we derive an upper bound on w_j, similar to the upper bound in the definition of the close split charge.

Lemma 2.30. *If j is charged using a distant split charge, then $w_g < w_{p_3}$ where p_3 is the third packet in the plan at time t, and $d_g = t+1$.*

Proof. Suppose that $w_g \ge w_{p_3}$. Then, from Lemma 2.29(d) and the choice of $p_2 = f$ in the algorithm, we have that $2\alpha w_{p_1} < w_{p_2} + w_{p_3} \le w_f + w_g$, so we would use the close split charge in step t, not the distant one. Thus $w_g < w_{p_3}$, as claimed.

To prove the second part, if we had $d_g = t+2$ then, since the algorithm chose g in step $t+1$ and also $d_{p_3} = t+2$, we would have $w_g \ge w_{p_3}$ – a contradiction. □

Lemma 2.31. *If j is charged using a distant split charge, then $2\alpha \cdot w_j < w_f + w_h$. (Recall that h is the packet scheduled in step $t+2$ in ALG.)*

Proof. Let p_1, p_2, p_3 be the plan in step t. By Lemma 2.29(d) we have that $f = p_2$. Thus $2\alpha \cdot w_{p_1} < w_{p_2} + w_{p_3}$ by the definition of the algorithm. By Lemma 2.29(c), j is expiring and $w_j \le w_{p_1}$. As $g \ne p_3$ by Lemma 2.30, the algorithm has p_3 pending in step $t+2$ where it is expiring, implying that $w_{p_3} \le w_h$. Putting it all together, we get $2\alpha w_j \le 2\alpha w_{p_1} < w_{p_2} + w_{p_3} \le w_f + w_h$. □

For a split charge from j in step t, let t' be the other step that receives the split charge from j; that is, $t' = t+1$ for a close split charge and $t' = t+2$ for a distant split charge. We now show that split-charge pairs are pairwise disjoint.

Lemma 2.32. *If j is charged using a split charge to a pair of steps t and t', then neither of t and t' is involved in another pair that receives a split charge from a packet $j' \neq j$.*

Proof. No matter which split charge we use for j, using Lemma 2.29(b), step $t+1$ does not receive a split charge from $k = f$. By a similar argument, since j is not scheduled in step $t-1$ in ALG, step t does not receive a close split charge from the packet scheduled in step $t-1$ in OPT.

It remains to prove that if j is charged using a distant split charge, then the packet ℓ scheduled in step $t+2$ in OPT is not charged using a split charge. (This also ensures that step t does not receive a distant split charge from a packet scheduled in step $t-2$ in OPT.)

For a contradiction, suppose that packet ℓ is charged using a split charge. Let p_1, p_2, p_3 be the plan in step t. Recall that g and h are the packets scheduled in steps $t+1$ and $t+2$ in ALG.

From Lemma 2.30, step $t+1$ does not receive a full back charge. Since we did not apply the close split charge for j in Rule 3, we must have

$$2\alpha(w_{p_1} - w_g) \geq w_f + w_g \geq w_f. \tag{2.30}$$

By Lemma 2.29(b) applied to step $t+2$, we get $d_h = t+3$. Since $d_{p_3} = t+2$, we get $w_{p_3} < w_h$. We now use Lemma 2.27 for step $t+1$ with $a = g, b = p_3$, and $c = h$. We note that all the assumptions of the lemma are satisfied: we have $d_g = t+1$, $(r_{p_3}, d_{p_3}) = (t+1, t+2)$, $(r_h, d_h) = (t+2, t+3)$, and $w_g < w_{p_3} < w_h$. This gives us that $2\alpha w_g \geq w_{p_3} + w_h > w_{p_3}$.

Since the algorithm schedules $f = p_2$ in step t, we have $2\alpha w_{p_1} < w_f + w_{p_3}$. Subtracting the inequality derived in the previous paragraph, we get $2\alpha(w_{p_1} - w_g) < (w_f + w_{p_3}) - w_{p_3} = w_f$ – a contradiction with (2.30). This completes the proof. □

The lemmas above allow us to estimate the total of full and split charges.

Lemma 2.33. *If j is charged using a split charge to a pair of steps t and t', then the total of full and split charges to steps t and t' does not exceed $R\cdot(w_f + w_{f'})$ where f' is the packet scheduled in step t' in ALG.*

Proof. Each of steps t and t' may receive a full charge, but each step at most one full charge from a packet of smaller or equal weight by Lemma 2.28 and charging rules.

First suppose that we use a distant split charge, or we use a close split charge and $2\alpha\cdot w_{p_1} < w_f + w_{f'}$. Then we have $2\alpha\cdot w_j < w_f + w_{f'}$ by Lemma 2.31 for a distant split charge or by $w_j \leq w_{p_1}$ from Lemma 2.29(c) for a close split charge. Thus the total of full and split charges to steps t and t' is upper bounded by

$$w_f + w_{f'} + w_j < w_f + w_{f'} + \frac{w_f + w_{f'}}{2\alpha} = \left(1 + \frac{1}{2\alpha}\right)\cdot(w_f + w_{f'}) = R\cdot(w_f + w_{f'})$$

where we used (2.23) in the last step.

Otherwise, i.e., if we use a close split charge and $2\alpha\cdot w_{p_1} \geq w_f + w_{f'}$, then step $t' = t+1$ does not get a full back charge and we have $2\alpha\cdot(w_{p_1} - w_{f'}) < w_f + w_{f'}$ by Rule 3. By Lemma 2.29(c) we have $w_j \leq w_{p_1}$ and $2\alpha\cdot(w_j - w_{f'}) < w_f + w_{f'}$. Also, step $t+1$ does not receive a full up charge by Lemma 2.29(b). We thus bound the total of full and split charges to steps t and $t+1$ by

$$w_f + w_j < w_f + \frac{w_f + (2\alpha+1)\cdot w_{f'}}{2\alpha} = \left(1 + \frac{1}{2\alpha}\right)\cdot(w_f + w_{f'}) = R\cdot(w_f + w_{f'})$$

using (2.23) in the last step again. □

Analysis of chain charges. We now analyze chaining steps starting with a lemma below consisting of several useful observations. In particular, Part (c) motivates the name "chaining" for such steps.

Lemma 2.34. *If step t is a chaining step, then the following holds:*
(a) $d_j = t + 1$,
(b) $d_f = t$.
Moreover, if step $t + 1$ is also a chaining step, then
(c) j is scheduled by the algorithm in step $t + 1$, i.e., $g = j$,
(d) $2\alpha \cdot w_f \geq w_j + w_k$ (recall that k is the packet scheduled in step $t + 1$ in OPT).

Proof. Recall that Algorithm CompareWithBias(α) never schedules a packet lighter than the heaviest expiring packet. As in step t it schedules f with $w_f < w_j$ (by Rule 5 for chain charges) and j is pending (otherwise we use Rule 1), (a) follows. Furthermore, it follows that f is expiring in step t, because otherwise, the algorithm would schedule j (or another packet of weight at least w_j), since both would have the same deadline and j is heavier. Thus (b) holds as well.

Now assume that step $t+1$ is also in the chain and for a contradiction suppose that $g \neq j$. Since j is expiring and pending for the algorithm in step $t+1$, we have $w_g > w_j$ and $w_k > w_g$ as step $t + 1$ is in the chain.

Summarizing, the algorithm sees all packets f, j, g, k in step t (some are pending and some may be lookahead packets), and they are all distinct packets with $w_f < w_j < w_g < w_k$, $d_f = t$, $(r_j, d_j) = (t, t + 1)$, and both g and k can be feasibly scheduled at time $t+1$. Thus, independently of the release times and deadlines of g and k, the plan at time t containing f would not be optimal – a contradiction. This proves that (c) holds.

Finally, we show (d). Since f is expiring in step t by (b) and both j and k are considered for the plan at time t and satisfy $(r_j, d_j) = (t, t+1)$, $(r_k, d_k) = (t+1, t+2)$ by (a), $w_f < w_j < w_k$, we use Lemma 2.27 with $a = f, b = j$, and $c = k$ and get the inequality in (d). ☐

First, we show that chaining steps do not receive charges of other types.

Lemma 2.35. *If step t is a chanining step, then t does not receive a full charge or a split charge.*

Proof. By Lemma 2.34, f is expiring, thus step t does not receive a full back charge. As $w_j > w_f$, the step also does not get a full up charge or a split charge from step t. So it remains to show that f does not receive a split charge.

First observe that step t cannot receive a close split charge from step $t-1$ in OPT, because j is pending in step t in ALG, while Lemma 2.29(b) states that a split charge from step $t-1$ would require j to be scheduled at time $t-1$ in ALG.

Finally, we show that step t does not receive a distant split charge. For a contradiction, suppose that step t receives a distant split charge from the packet x scheduled in step $t-2$ in OPT. Let p_1, p_2, p_3 be the plan in step $t-2$. According to Lemma 2.29(d) and (b), p_2 is scheduled in step $t-2$ in ALG and in step $t-1$ in OPT. Moreover, by Lemma 2.29(c), x is pending and expiring in step $t-2$ and $w_{p_1} \geq w_x$. As the algorithm scheduled p_2 in step $t-2$ we get $r_{p_2} = t-2$ and $w_{p_1} < w_{p_3}$.

Observe that p_3 is not scheduled in OPT, since it is expiring in step t and j is not expiring, by Lemma 2.34(a). Thus we could increase the weight of OPT if we scheduled p_2 in step $t-2$ instead of x and p_3 in step $t-1$. This contradicts the optimality of OPT. ☐

We now analyze how much charge does each chaining step get.

Lemma 2.36. *If step t is a chaining step, then it receives a charge of at most $R \cdot w_f$.*

Proof. By Lemma 2.35, step t does not receive any full or split charges; therefore we just need to prove that the total of chain charges to step t does not exceed $R \cdot w_f$.

Case 1: t is the last step of a chain. If t is the only step in the chain then Rule 5a implies directly that the charge to t is at most $R \cdot w_f$. Otherwise, Lemma 2.34(c) implies that f is scheduled in step $t-1$ in OPT, and thus the charge from step $t-1$ is $(1-2\delta) \cdot w_f$. The charge from step t is $(R-1+2\delta) \cdot w_f$ by Rule 5c. So the total charge is at most $R \cdot w_f$.

Case 2: t is not the last step of a chain. Since step $t+1$ is also in the chain, by Lemma 2.34(c) we have that j is scheduled in step $t+1$ in ALG and OPT has a packet k with $w_k > w_j$ in step $t+1$. From Lemma 2.34(d) we know that $2\alpha \cdot w_f \geq w_j + w_k$.

There are two sub-cases. If t is the first step of the chain, then the charge to t is at most

$$2\delta \cdot w_j + \delta \cdot w_k \leq \tfrac{3}{2}\delta \cdot (w_j + w_k) \leq 3\alpha\delta \cdot w_f < R \cdot w_f,$$

where the last inequality follows from (2.28). Otherwise, using Lemma 2.34(c), f is scheduled in step $t-1$ in OPT, so the total charge to step t is at most

$$(1-2\delta) \cdot w_f + \delta \cdot w_j + \delta \cdot w_k \leq (1-2\delta) \cdot w_f + 2\alpha\delta \cdot w_f = R \cdot w_f$$

where the last equality follows from (2.22). □

Analysis of forward charges from chains. We now show that a forward charge from a chain does not cause an overload on the step just after the chain which may also get both a full charge and a split charge. (This is the only case when a step receives charges of all three types.)

For the following lemmas, we assume that step $t-1$ is a chaining step. Recall that i is the packet scheduled in step $t-1$ in OPT and e is the packet scheduled in step $t-1$ in ALG. First, we prove some useful observations.

Lemma 2.37. *If step t receives a forward charge from a chain, then the following holds*
(a) $j \neq e$ (that is, j is not charged using a full back charge),
(b) $w_f \geq w_j$,
(c) $w_f \geq w_i$.
Moreover, if step t is not in a split-charge pair:
(d) j is charged using a full up charge to step t,
(e) step t does not receive a full back charge.

Proof. Part (a) holds because e is expiring in step $t-1$, by Lemma 2.34(b). Part (b) follows from (a) and the fact that step t is not chaining.

To show (c), Lemma 2.34(a) implies that $d_i = t$. Also, since $i \neq e$, i is pending in ALG in step t. Now (c) follows, because each packet scheduled by the algorithm is at least as heavy as the heaviest expiring packet.

Part (d) follows from (a) and (b) and the assumption that t is not in a split-charge pair. Part (e) follows from (d) and Lemma 2.28. □

Note that f may be the same packet as i or j. We start with the case in which f is not in a split-charge pair.

Lemma 2.38. *If step t receives a forward charge from a chain C and t is not in a split-charge pair, then the total charge to step t is at most $R \cdot w_f$.*

Proof. The proof is by case analysis, depending on the relative weights of j and e, and on whether C is a singleton or a long chain. In all cases we use Lemma 2.37 and the charging rules to show upper bounds on the total charge.

Case 1: $w_j < w_e$.

Case 1.1: The chain C is long. The charge to step t is then at most

$$\begin{aligned} w_j + (1-\delta)\cdot w_i - (R-1+2\delta)\cdot w_e &< w_j + (1-\delta)\cdot w_i - (R-1+2\delta)\cdot w_j \\ &= (2-R-2\delta)\cdot w_j + (1-\delta)\cdot w_i \\ &\le (2-R-2\delta)\cdot w_f + (1-\delta)\cdot w_f \qquad (2.31) \\ &= (3-R-3\delta)\cdot w_f = w_f\,. \qquad (2.32) \end{aligned}$$

To justify inequality (2.31), note that $2-R-2\delta \ge 0$ by (2.25) and $1-\delta \ge 0$ by the choice of δ, so we can apply inequalities $w_j \le w_f$ and $w_i \le w_f$ from Lemma 2.37(b) and (c). The last step (2.32) follows from equation (2.24).

Case 1.2: The chain C is singleton. We assume that $w_i > R\cdot w_e$, otherwise there is no forward charge from the chain. Then the charge to step t is

$$w_j + w_i - R\cdot w_e \le w_j + w_i - R\cdot w_j \le w_i \le w_f\,,$$

where in the last step we used Lemma 2.37(c).

Case 2: $w_j > w_e$. We claim first that j is not expiring in step t, that is $d_j = t+1$. Indeed, if we had $d_j = t$, then in step $t-1$ the algorithm would have pending packets e and i, plus packet j (pending or lookahead), that need to be scheduled in slots $t-1$ and t. Since $w_e < w_i$ (because step $t-1$ is a chaining step) and $w_e < w_j$ (by the case assumption), packet e could not be in the plan in step $t-1$ which is a contradiction. Thus $d_j = t+1$.

Recall that e is expiring in step $t-1$ by Lemma 2.34(b) and both i and j are considered for the plan in step $t-1$. Moreover, we know that $w_i > w_e$, $w_j > w_e$, $(r_i, d_i) = (t-1, t)$ (by Lemma 2.34(a)), and $(r_j, d_j) = (t, t+1)$. We thus use Lemma 2.27 for step $t-1$ with $a = e, b = i$, and $c = j$, to get that $2\alpha\cdot w_e \ge w_i + w_j$.

Case 2.1: The chain C is long. The charge to step t is

$$\begin{aligned} &w_j + (1-\delta)\cdot w_i - (R-1+2\delta)\cdot w_e \\ &\quad\le w_j + (1-\delta)\cdot w_i - (R-1+2\delta)\cdot \frac{w_i + w_j}{2\alpha} \\ &\quad= \left(1 - \frac{R-1+2\delta}{2\alpha}\right)\cdot w_j + \left(1-\delta-\frac{R-1+2\delta}{2\alpha}\right)\cdot w_i \\ &\quad\le \left(1 - \frac{R-1+2\delta}{2\alpha}\right)\cdot w_f + \left(1-\delta-\frac{R-1+2\delta}{2\alpha}\right)\cdot w_f \qquad (2.33) \\ &\quad= \left(2-\delta-\frac{R-1+2\delta}{\alpha}\right)\cdot w_f = R\cdot w_f\,. \qquad (2.34) \end{aligned}$$

To justify inequality (2.33), we note that $1-\delta-(R-1+2\delta)/(2\alpha) \ge 0$ by (2.26), so we can again apply inequalities $w_j \le w_f$ and $w_i \le w_f$ from Lemma 2.37(b) and (c). In the last step (2.34) we used equation (2.21).

Case 2.2: The chain C is singleton. We assume that $w_i > R\cdot w_e$, otherwise there is no forward charge from the chain. Then the charge to step t is

$$\begin{aligned} w_j + w_i - R\cdot w_e &\le w_j + w_i - R\cdot \frac{w_i + w_j}{2\alpha} \\ &= \left(1 - \frac{R}{2\alpha}\right)\cdot(w_i + w_j) \\ &\le \left(1 - \frac{R}{2\alpha}\right)\cdot(2w_f) \qquad (2.35) \\ &= \left(2 - \frac{R}{\alpha}\right)\cdot w_f < R\cdot w_f\,. \qquad (2.36) \end{aligned}$$

Inequality (2.35) is valid, because $w_i \le w_f$ and $w_j \le w_f$, by Lemma 2.37, and $1 - R/(2\alpha) \ge 0$ by (2.27). In step (2.36) we used (2.29). □

We now analyze how the forward charge from a chain combines with split charges. First, we observe that only the first step from a split-charge pair may receive a forward charge from a chain.

Lemma 2.39. *If j is charged using a split charge to a pair of steps t and t' (where t' is $t+1$ or $t+2$), then t' does not receive a forward charge from a chain.*

Proof. By Lemma 2.29(b) we have $k = f$, which implies that steps t and $t+1$ are not chaining steps. □

Lemma 2.40. *If j is charged using a split charge to a pair of steps t and t', f' is the packet scheduled in t' in* ALG*, and step t receives a forward charge from a chain C, then the total charge to steps t and t' is at most $R\cdot(w_f + w_{f'})$.*

Proof. First, we note that j is expiring in step t by Lemma 2.29(c). Furthermore, i is expiring in step t by Lemma 2.34(a) and f is not expiring in step t by Lemma 2.29(b), so $f \neq i$.

We claim $w_j < w_e$. Indeed, if $w_j > w_e$, then in step $t-1$ the algorithm would have pending packets e and i, plus packet j (pending or lookahead), that need to be scheduled in slots $t-1$ and t. Since $w_e < w_i$ (because step $t-1$ is a chaining step) and $w_e < w_j$, packet e could not be in the plan in step $t-1$ which is a contradiction. Therefore $w_j < w_e$.

Let p_1, p_2, p_3 be the plan at time t. We split the proof into two cases, both having two subcases, one for long chains and one for singleton chains.

Case 1: j is charged using a distant split charge or f' gets a full back charge.

We claim that $2\alpha\cdot w_i < w_f + w_{f'}$. Indeed, since i is expiring and pending in step t by Lemma 2.34(a), we have $w_i \le w_{p_1}$. As the algorithm scheduled $f = p_2$ by Lemma 2.29(d), we get $2\alpha\cdot w_{p_1} < w_f + w_{p_3}$. To prove the claim, it remains to show $w_{f'} \ge w_{p_3}$.

If j is charged using a distant split charge, then by Lemma 2.30 we have $w_g < w_{p_3}$ and in particular, $g \neq p_3$. Thus p_3 is pending and expiring in step $t{+}2$, hence $w_{p_3} \le w_{f'}$. Otherwise, if j is charged using a close split charge, then $f' = g$ gets a full back charge. Hence $d_{f'} = t+2$. Since also $d_{p_3} = t+2$ and the algorithm chooses the heaviest such packet, we have $w_{p_3} \le w_{f'}$.

The claim follows, since

$$2\alpha\cdot w_i \le 2\alpha\cdot w_{p_1} < w_f + w_{p_3} \le w_f + w_{f'}\,. \qquad (2.37)$$

Case 1.1: The chain C is long. The total charge to steps t and t', consisting of the full charge to f, a possible full charge to f', the split charge, and the forward charge from the chain, is at most

$$\begin{aligned}
w_f + w_{f'} + w_j + (1-\delta)\cdot w_i - (R-1+2\delta)\cdot w_e \\
&\le w_f + w_{f'} + (2-R-2\delta)\cdot w_e + (1-\delta)\cdot w_i \\
&< w_f + w_{f'} + (2-R-3\delta)\cdot w_i + w_i && (2.38)\\
&= w_f + w_{f'} + w_i && (2.39)\\
&< w_f + w_{f'} + \frac{w_f + w_{f'}}{2\alpha} && (2.40)\\
&= R\cdot(w_f + w_{f'})\,.
\end{aligned}$$

We can use $w_e < w_i$ in (2.38), because $2-R-2\delta \ge 0$ by (2.25). Equality (2.39) follows from $2-R-3\delta = 0$ by (2.24) and inequality (2.40) from (2.37). In the last step we use (2.23).

<u>Case 1.2</u>: The chain C is singleton. We suppose that $w_i > R\cdot w_e$, otherwise there is no forward charge from the chain. We upper bound the total charge to steps t and t' by

$$\begin{aligned}
w_f + w_{f'} + w_j + w_i - R\cdot w_e &\le w_f + w_{f'} + (1-R)\cdot w_e + w_i \\
&< w_f + w_{f'} + w_i \\
&< w_f + w_{f'} + \frac{w_f + w_{f'}}{2\alpha} && (2.41)\\
&= R\cdot(w_f + w_{f'})\,,
\end{aligned}$$

where we apply Equation 2.37 in (2.41), and we use (2.23) in the last step.

<u>Case 2</u>: j is charged using a close split charge and $f' = g$ does not get a full back charge. Observe that in this case g does not receive any full charge as $k = f$ is charged by a full back charge. We have (i) $2\alpha\cdot w_{p_1} < w_f + w_g$, or (ii) $2\alpha\cdot(w_{p_1} - w_g) < w_f + w_g$ by the definition of the close split charge. We suppose that we have (ii), since (i) is stronger than (ii).

Since i is expiring and pending in step t by Lemma 2.34(a), we have $w_i \le w_{p_1}$. Hence $2\alpha\cdot(w_i - w_g) < w_f + w_g$. This is equivalent to

$$w_i < \frac{w_f + (2\alpha+1)\cdot w_g}{2\alpha}\,. \tag{2.42}$$

<u>Case 2.1</u>: The chain C is long. The total charge to steps t and $t' = t+1$ is

$$\begin{aligned}
w_f + w_j + (1-\delta)\cdot w_i - (R-1+2\delta)\cdot w_e \\
&\le w_f + (2-R-2\delta)\cdot w_e + (1-\delta)\cdot w_i \\
&< w_f + (2-R-3\delta)\cdot w_i + w_i && (2.43)\\
&= w_f + w_i && (2.44)\\
&< w_f + \frac{w_f + (2\alpha+1)\cdot w_{f'}}{2\alpha} && (2.45)\\
&= w_f + w_{f'} + \frac{w_f + w_{f'}}{2\alpha} \\
&= R\cdot(w_f + w_{f'})\,,
\end{aligned}$$

We can use $w_e < w_i$ in (2.43), because $2-R-2\delta \ge 0$ by (2.25). Then we use $2-R-3\delta = 0$ by (2.24) in (2.44), Equation 2.42 in (2.45), and Equation 2.23 in the last step.

<u>Case 2.2</u>: The chain C is singleton. We again suppose that $w_i > R\cdot w_e$, as otherwise there is no forward charge from the chain. We upper bound the total charge to steps t

and $t+1$ by

$$\begin{aligned} w_f + w_j + w_i - R\cdot w_e &\le w_f + (1-R)\cdot w_e + w_i \\ &< w_f + w_i \\ &< w_f + \frac{w_f + (2\alpha+1)\cdot w_{f'}}{2\alpha} \qquad (2.46) \\ &= w_f + w_{f'} + \frac{w_f + w_{f'}}{2\alpha} \\ &= R\cdot(w_f + w_{f'})\,, \end{aligned}$$

where we apply (2.42) in inequality (2.46), and (2.23) in the last step. □

We now summarize our analysis of CompareWithBias(α). If t is not in a split-charge pair, we show upper bounds on the total charge to step t. For each split-charge pair (t, t'), we show upper bounds on the total charge to both steps t and t'. This is sufficient since split-charge pairs are pairwise disjoint by Lemma 2.32, thus summing all the bounds gives the result in Theorem 2.26.

For each step t, we distinguish three cases according to whether t is in a split-charge pair and whether t is a chaining step. In all cases, let f be the packet scheduled at time t in ALG and let j be the packet scheduled at time t in OPT.

Case 1: Step t is not chaining and it is not in a split-charge pair. Then t receives at most one full charge from a packet p such that $w_p \le w_f$ (by Lemma 2.28 and charging rules) and possibly a forward charge from a chain C; then Lemma 2.38 shows that the sum of a forward charge from a chain and a full charge is at most $R\cdot w_f$.

Case 2: Step t is a chaining step. Then it does not receive a split charge or a full charge, by Lemma 2.35. Lemma 2.36 implies that step t receives a charge of at most $R\cdot w_f$.

Case 3: (t, t') is a split-charge pair, i.e., t is the first step of the split-charge pair and $t' = t+1$, or $t' = t+2$. Thus j is charged using a split charge. Let f' be the packet scheduled in step t' in ALG.

By Lemma 2.39 step t' does not receive a forward charge from a chain. If step t also does not receive a forward charge from a chain, then the total charge to steps t and t' is at most $R\cdot(w_f + w_{f'})$ by Lemma 2.33. Otherwise, step t receives a forward charge from a chain and we apply Lemma 2.40 to show that the total charge to steps t and t' is again at most $R\cdot(w_f + w_{f'})$.

2.7.2 A Lower Bound for 2-bounded Instances with Lookahead

In this section, we prove that there is no deterministic online algorithm for Bounded-Delay Packet Scheduling with ℓ-lookahead that has competitive ratio smaller than $R := \frac{1}{2(\ell+1)}(1+\sqrt{5+8\ell+4\ell^2})$ for any $\ell \ge 0$, even for 2-bounded instances. We note that $R > 1$ for any $\ell \ge 0$, that R tends to 1 as ℓ goes to infinity, and that R is the positive root of the quadratic equation

$$(\ell+1)R^2 - R - (\ell+1) = 0\,. \qquad (2.47)$$

The idea of our proof is similar to the proof of the lower bound of ϕ for Bounded-Delay Packet Scheduling [Haj01, AMZ03, CF03] and, indeed, for $\ell = 0$ our lower bound is equal to ϕ. For the case of 1-lookahead we obtain a lower bound of $\frac{1}{4}(1+\sqrt{17}) \approx 1.281$.

Theorem 2.41. *Let $\ell \ge 0$ be an integer and $R = \frac{1}{2(\ell+1)}(1+\sqrt{5+8\ell+4\ell^2})$. For each $\varepsilon > 0$, no deterministic online algorithm for Bounded-Delay Packet Scheduling with ℓ-lookahead can be $(R-\varepsilon)$-competitive, even for 2-bounded instances.*

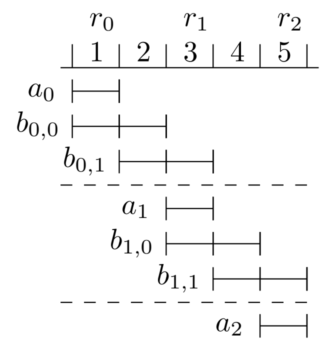


Figure 2.17: The instance for $\ell = 1$ and $k = n = 2$. Each packet has a row which shows slots between its release time and deadline. Packets from different phases are separated by a dashed line.

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 Bounded-Delay Packet Scheduling with ℓ-lookahead, parameterized 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 $k \leq n$. The number k of phases is determined by the adversary based on the behavior of $\mathcal{A}$. Each phase (except phase n) will involve $\ell + 2$ packets. The weights of these packets will grow roughly exponentially from one phase to next.

The adversary strategy is as follows. We start with phase 0. Suppose that some phase i, where $0 \leq i < n$, has been reached. Let $r_i = (\ell + 1)i + 1$ be the first slot of phase i. In phase i the adversary releases the following $\ell + 2$ packets:

- A packet a_i with weight w_i, release time r_i and deadline r_i, i.e., a tight packet.
- Packets $b_{i,j}$ for $j = 0, \ldots, \ell$ with weight w_{i+1}, release time $r_i + j$ and deadline $r_i + j + 1$.

(The weights w_i will be specified later.) Now, if $\mathcal{A}$ schedules an expiring packet in step r_i (a tight packet a_i or $b_{i-1,\ell}$, 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 $b_{i,0}$, in which case the adversary lets $k = i$ and the game ends. Note that in steps $r_i + 1, \ldots, r_i + \ell$ the algorithm may schedule only $b_{i,j}$ (for some j) of weight w_{i+1}. Also, importantly, in step r_i the algorithm cannot yet see whether the packets from phase $i + 1$ will arrive or not.

If phase $i = n$ is reached, then $k = n$, and in phase n the adversary releases a single tight packet a_n with weight w_n and release time and deadline r_n. See Figure 2.17 for an illustration.

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 r_i for $i < k$ algorithm $\mathcal{A}$ scheduled an expiring packet of weight w_i and in step r_k it scheduled packet $b_{k,0}$ of weight w_{k+1}. In steps $r_i + 1, \ldots, r_i + \ell$ for $i < k$ it scheduled packets of weight w_{i+1}. Finally, in phase k the algorithm scheduled $\ell + 1$ packets of weight w_{k+1}, including $b_{k,0}$. Overall, $\mathcal{A}$ scheduled packets of total weight $S_{k-1} + \ell \cdot (S_k - w_0) + (\ell + 1) \cdot w_{k+1} = (\ell + 1) \cdot S_{k+1} - w_k - \ell \cdot w_0$.

The adversary schedules packets of weight w_{i+1} in steps $r_i, \ldots, r_i + \ell$ for $i < k$ and all packets from phase k in steps $r_k, \ldots r_k + \ell + 1$. In total, the optimum has a schedule of weight $(\ell + 1) \cdot (S_k - w_0) + w_k + (\ell + 1) \cdot w_{k+1} = (\ell + 1) \cdot S_{k+1} + w_k - (\ell + 1) \cdot w_0$.

The ratio is

$$R_k = \frac{(\ell+1)\cdot S_{k+1} + w_k - (\ell+1)\cdot w_0}{(\ell+1)\cdot S_{k+1} - w_k - \ell\cdot w_0}.$$

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

The adversary schedules packets of weight w_{i+1} in steps $r_i, \ldots, r_i+\ell$ for $i < n$ and a packet of weight w_n in the last step r_n which adds up to $(\ell+1)\cdot S_n + w_n - (\ell+1)\cdot w_0$. The ratio is

$$\widehat{R}_n = \frac{(\ell+1)\cdot S_n + w_n - (\ell+1)\cdot w_0}{(\ell+1)\cdot S_n - \ell\cdot w_0}.$$

We normalize the instances so that $w_0 = 1$. It remains to show that we can set the weights so that $R_k \geq R - \varepsilon$ for all $k \geq 0$ and $\widehat{R}_n \geq R - \varepsilon$.

We first define a sequence of weights, parametrized by some parameter $\delta \geq 0$, such that $R_k = R$ for all $k \geq 1$. Using $w_k = S_k - S_{k-1}$ for $k \geq 1$ and $w_0 = 1$, the condition $R_k = R$ for $k \geq 1$ is rewritten as

$$R = \frac{(\ell+1)\cdot S_{k+1} + S_k - S_{k-1} - (\ell+1)}{(\ell+1)\cdot S_{k+1} - S_k + S_{k-1} - \ell},$$

or equivalently as

$$(\ell+1)(R-1)S_{k+1} - (R+1)S_k + (R+1)S_{k-1} = \ell R - (\ell+1). \tag{2.48}$$

By (2.47) we get that $(\ell+1)(R-1) = R/(R+1)$ and similarly $\ell R-(\ell+1) = -R^2/(R+1)$. Substituting and multiplying by $R+1$, we obtain that (2.48) is equivalent to

$$R\cdot S_{k+1} - (R+1)^2 S_k + (R+1)^2 S_{k-1} = -R^2. \tag{2.49}$$

To define our instance, we set $w_0 = 1$ and for $i = 1, 2, \ldots$,

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

where

$$\alpha = 1 + \frac{1}{R} = \frac{R+1}{R}, \quad \beta = R+1, \quad \gamma = R,$$

and $\delta > 0$ is a parameter to be chosen later. Summing the geometric sequences in $S_k = \sum_{i=0}^{k} w_i$, we obtain that, for $k = 0, \ldots, n$,

$$S_k = (\gamma+1)\alpha^k + \delta(\beta^k - \alpha^k) - \gamma. \tag{2.50}$$

It can be verified that (2.49) holds and thus, for any choice of δ and any $k \geq 1$, we have $R_k = R$. In fact, (2.50) describes a general solution of the linear recurrence (2.49) that satisfies one initial condition $S_0 = w_0 = 1$, as α, β are the two roots of $Rx^2 - (R+1)^2 x + (R+1)^2$ which is the characteristic polynomial of the recurrence, and $S_0 = S_1 = \cdots = S_n = -\gamma$ is a particular solution of the recurrence; furthermore, for $\delta = 0$ (2.50) gives a particular solution satisfying $S_0 = 1$ and changing δ does not change S_0.

We now show that for $\delta = 0$ the solution would satisfy $R_0 = R$. We first calculate w_1:

$$w_1 = (\gamma+1)(\alpha-1) = (R+1)\cdot\frac{1}{R} = \alpha.$$

By (2.47) we have $(\ell+1)\alpha = 1/(R-1)$. Using this we can calculate R_0 as

$$R_0 = \frac{(\ell+1)w_1+1}{(\ell+1)w_1} = 1 + \frac{1}{(\ell+1)\alpha} = R\,.$$

By continuity of the dependence of w_1 and R_0 on δ, for a sufficiently small $\delta > 0$ we have $R_0 \geq R - \varepsilon$. We fix such a $\delta > 0$.

Since $1 < \alpha < \beta$, for $n \to \infty$, the dominating term in S_n is $\delta\beta^n$ and w_0 is negligible compared to S_n. Thus we obtain

$$\begin{aligned}\lim_{n\to\infty} \widehat{R}_n = \lim_{n\to\infty} \frac{(\ell+1)S_n + S_n - S_{n-1}}{(\ell+1)S_n} &= \lim_{n\to\infty} \frac{(\ell+2)\delta\beta^n - \delta\beta^{n-1}}{(\ell+1)\delta\beta^n} \\ &= \frac{(\ell+2)\beta - 1}{(\ell+1)\beta} = R\,.\end{aligned}$$

The last equality follows from (2.47). Actually, this is the equation that defines R as the optimal ratio for our construction (if β is expressed in terms of R as the root of the characteristic polynomial). Consequently, for some sufficiently large n, we have that $\widehat{R}_n \geq R - \varepsilon$. Fix this n.

Summarizing, we showed that for any $\varepsilon > 0$ the adversary can choose $\delta > 0$ and n, and an instance with up to n phases, such that for this instance we have $R_0 \geq R - \varepsilon$, $R_k = R$ for all $k \geq 1$, and $\widehat{R}_n \geq R - \varepsilon$. This implies that the competitive ratio of $\mathcal{A}$ is at least R, completing the proof. ☐

2.7.3 Lower Bounds for Randomized Algorithms with Lookahead

Finally, we show that if packets have large spans, then any constant lookahead does not help to achieve a ratio close to 1 even for randomized algorithms. Our lower bound of 1.25 against the oblivious adversary is a straightforward generalization of the one by Chin and Fung [CF03] for 2-bounded instances without lookahead.

Theorem 2.42. *For any fixed $\ell \geq 0$, there is no better than 1.25-competitive randomized algorithm with ℓ-lookahead against the oblivious adversary, even on agreeable instances.*

Proof. We use the easy direction of the Yao's minimax principle for online algorithms (see e.g. [BE98]), which states that the expected ratio of a randomized algorithm on the worst-case instance is no better than the expected ratio of the best deterministic algorithm on an input drawn from the worst-case probability distribution of instances.

Let $n, k \gg \ell$ be large integers. We define a set of $n+1$ instances as follows (each instance is a set of packets, with each packet p specified by a triple (r_p, d_p, w_p)):

$$\begin{aligned}&J_0 = \{a_{0,m} = (1,k,1), b_{0,m} = (1,2k,2) \mid m = 1,\dots,k\},\\ &J_i = J_{i-1} \cup \{a_{i,m} = (ik+1,(i+1)k,2^i), b_{i,m} = (ik+1,(i+2)k,2^{i+1}) \mid m = 1,\dots,k\}\\ &\qquad \text{for } i = 1,\dots,n-1,\\ &J_n = J_{n-1} \cup \{a_{n,m} = (nk+1,(n+1)k,2^n) \mid m = 1,\dots,k\}\,.\end{aligned}$$

In words, in instance J_i, $i = 0,\dots,n-1$, in step $ik+1$ the adversary releases k packets $a_{i,1},\dots,a_{i,k}$ of span k and k packets $b_{i,1},\dots,b_{i,k}$ of span $2k$ in addition to packets from instance J_{i-1}. In instance J_n it additionally releases k packets $a_{n,1},\dots,a_{n,k}$ of span k only. See Figure 2.18 for an example. Clearly, all instances J_i have agreeable deadlines. For $i = 0,\dots,n$, we call the interval $[ik+1,(i+1)k]$ of slots the i-th *phase*. Note that in each phase except the last one, pending packets are of two types: lighter packets expiring in that phase and heavier packets expiring in the next phase.

We define the probability distribution of the instances J_i such that for $i = 0, \dots, n-1$, instance J_i is drawn with probability $p_i = 1/2^{i+1}$ and instance J_n is drawn with probability $p_n = 1/2^n$. Clearly, $\sum_i p_i = 1$.

First, we analyze the offline optimum profits. Observe that on instance J_i, $i = 1, \dots, n-1$, in the j-th phase for $j = 0, \dots, i-1$ the adversary schedules k heavier packets (of span $2k$), which are expiring in the next phase, while in each of the last two phases it transmits k packets expiring in the phase. Thus its profit is $\mathsf{OPT}(J_i) = \sum_{j=0}^{i-1} k \cdot 2^{j+1} + k \cdot 2^i + k \cdot 2^{i+1} = k \cdot (2^{i+2} + 2^i - 2)$. Similarly, it holds that $\mathsf{OPT}(J_n) = \sum_{j=0}^{n-1} k \cdot 2^{j+1} + k \cdot 2^n = k \cdot (2^{n+1} + 2^n - 2)$. It follows that the expected optimum profit is

$$\begin{aligned}
\mathsf{E}[\mathsf{OPT}] &= \sum_{i=0}^{n} p_i \cdot \mathsf{OPT}(J_i) = \sum_{i=0}^{n-1} \frac{k \cdot (2^{i+2} + 2^i - 2)}{2^{i+1}} + \frac{k \cdot (2^{n+1} + 2^n - 2)}{2^n} \\
&= k \cdot \left(\sum_{i=0}^{n-1} 2 + \frac{1}{2} - \frac{1}{2^i} \right) + k \cdot \left(2 + 1 - \frac{1}{2^{n-1}} \right) \\
&= k \cdot \left(\frac{5}{2}n - 2 + \frac{1}{2^{n-1}} + 3 - \frac{1}{2^{n-1}} \right) = k \cdot \left(\frac{5}{2}n + 1 \right) .
\end{aligned}$$

Fix a deterministic algorithm ALG. To analyze the expected profit of ALG, we first get rid of the influence of lookahead, which affects its behavior only in the last ℓ steps in each phase. Namely, we make the following assumption about the schedule $\mathsf{ALG}(J_i)$ without loss of generality: For each phase $j = 0, \dots, i-1$, in the last ℓ slots of the phase (when the algorithm knows that more packets will arrive) there are packets not expiring in the j-th phase only, i.e., the heaviest pending packets. Moreover, if $i < n$, in the last ℓ slots of the i-th phase (which is the penultimate phase) there are packets expiring in that phase only; this also trivially holds for the last phase.

Let $\alpha_i, i = 0, \dots, n-1$, be the number of lighter packets that the algorithm schedules in the first $k - \ell$ slots of the phase. We get the following expressions of the algorithm's profits on the instances:

$$\begin{aligned}
\mathsf{ALG}(J_i) &= \left(\sum_{j=0}^{i-1} \alpha_j \cdot 2^j + (k - \ell - \alpha_j + \ell) \cdot 2^{j+1} \right) + (\alpha_i + \ell) \cdot 2^i + k \cdot 2^{i+1} \\
&= \left(\sum_{j=0}^{i-1} k \cdot 2^{j+1} - \alpha_j \cdot 2^j \right) + \alpha_i \cdot 2^i + \ell \cdot 2^i + k \cdot 2^{i+1} \\
&\le k \cdot 2^{i+2} + \ell \cdot 2^i - \sum_{j=0}^{i-1} \alpha_j \cdot 2^j + \alpha_i \cdot 2^i \\
\mathsf{ALG}(J_n) &= \left(\sum_{j=0}^{n-1} \alpha_j \cdot 2^j + (k - \ell - \alpha_j + \ell) \cdot 2^{j+1} \right) + k \cdot 2^n \\
&= \left(\sum_{j=0}^{n-1} k \cdot 2^{j+1} - \alpha_j \cdot 2^j \right) + k \cdot 2^n \\
&\le k \cdot 2^{n+1} + k \cdot 2^n - \sum_{j=0}^{n-1} \alpha_j \cdot 2^j .
\end{aligned}$$

Our goal is to express $\mathsf{E}[\mathsf{ALG}] = \sum_i p_i \cdot \mathsf{ALG}(J_i)$. We now show that for any $j = 0, \dots, n-1$ the coefficient of α_j in $\mathsf{E}[\mathsf{ALG}]$ is 0 by simply summing for each i the

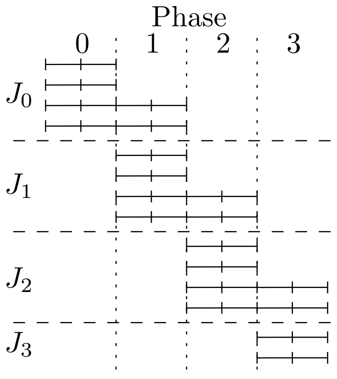


Figure 2.18: An illustration of the lower bound of 1.25 with $k = 2$ and $n = 3$. The dotted vertical lines split slots into phases.

coefficient of α_j in $\mathsf{ALG}(J_i)$ multiplied by p_i:

$$\frac{2^j}{2^{j+1}} - \sum_{i=j+1}^{n-1} \frac{2^j}{2^{i+1}} - \frac{2^j}{2^n} = 0$$

It follows that

$$\begin{aligned}\mathsf{E}\left[\mathsf{ALG}\right] &\leq \sum_{i=0}^{n-1} \frac{k \cdot 2^{i+2} + \ell 2^i}{2^{i+1}} + \frac{k \cdot 2^{n+1} + k \cdot 2^n}{2^n} \\ &= k \cdot n \cdot 2 + \ell \cdot n \cdot \frac{1}{2} + k \cdot 3 = k \cdot (2n+3) + \ell \cdot \frac{1}{2} n\,.\end{aligned}$$

We conclude that the ratio $\mathsf{E}\left[\mathsf{OPT}\right] / \mathsf{E}\left[\mathsf{ALG}\right]$ tends to 1.25 if n and k go to infinity. □

Note that in the case $\ell = 0$ (i.e., no lookahead), we can choose $k = 1$ and then the construction is the same as in [CF03].

2.8 Conclusions and Open Problems

Our main contribution is a ϕ-competitive algorithm for Bounded-Delay Packet Scheduling. It is based on a deep understanding of the plan and its updates after a packet arrives or after a packet is scheduled. The key idea is to maintain the slot-monotonicity property by increasing weights and decreasing deadlines of certain packets, which in turn can be used to amortize the adversary cost. However, maintaining the monotonicity property seems to require memory and we have some examples that the memoryless variant of our algorithm, called Plan(α), is not ϕ-competitive for any α. Hence, it is likely that we need a different approach to achieve ϕ-competitiveness without the help of memory. The best memoryless algorithm remains the 1.893-competitive algorithm due to Englert and Westermann [EW12] (which is very similar to Plan(α)). It would also be interesting to determine the competitive ratio of Plan(α) for the best choice of α.

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

We remark that a promising approach for improving the lower bound is to somehow try to simulate the lower bound of 1.633 for Item Collection from [BCD$^+$13a].

Randomized algorithms. The best randomized algorithm is RMix, which is $\frac{e}{e-1} \approx$ 1.582-competitive against both the oblivious and the adaptive adversary [CCF$^+$06, BCJ11]. The lower bounds are 1.25 against the oblivious adversary [CF03] and $\frac{4}{3}$ against the adaptive adversary [BCJ11]; both use 2-bounded instances only. The question is thus whether the techniques developed in this work and used to design our optimal deterministic algorithm can also be applied to improve the ratio of randomized algorithms.

Open Problem 2. *Design a better than $\frac{e}{e-1}$-competitive randomized algorithm, at least against the oblivious adversary, or improve one of the lower bounds.*

Special types of instances. Regarding deterministic algorithms, the optimal competitive ratio is resolved for most types of restricted instances, since the lower bound of ϕ holds already on 2-bounded instances. Moreover, the 2-uniform case is resolved as well (it is interesting that the optimal competitive ratio of memoryless deterministic algorithms is slightly larger than the optimal ratio of unrestricted deterministic algorithms). Thus the only missing piece in our understanding of special types of instances is to analyze s-uniform instances for $s > 2$. In particular, up to our knowledge, there is no lower bound for deterministic algorithms and the only result specifically for s-uniform instances with arbitrary s is the lower bound for randomized algorithms against the oblivious adversary by Chin *et al.* [CCF+06], which tends to 1.25 as s goes to infinity (this is an interesting coincidence with our lower bound for randomized algorithms with lookahead in Section 2.7.3).

Open Problem 3. *Design (deterministic or randomized) competitive algorithms for s-uniform instances, ideally working for any s, and construct lower bounds on the competitive ratio of algorithms on s-uniform instances.*

There are also many gaps for randomized algorithms, even on instances with agreeable deadlines or on 3-bounded instances; see Table 2.1.

Algorithms with lookahead. We initiated the study of the semi-online setting with ℓ-lookahead, in which the algorithm is aware of packets that arrive in the next ℓ steps. Generalizing the lower bound of ϕ, we constructed deterministic lower bounds for any ℓ that hold even on 2-bounded instances; the lower bound is equal to $\frac{1}{4}(1+\sqrt{17}) \approx 1.281$ for $\ell = 1$. We complemented the lower bound by a nearly optimal algorithm for 2-bounded instances with 1-lookahead, whose competitive ratio equals $\frac{1}{2}(\sqrt{13}-1) \approx 1.303$. The main question is whether lookahead helps on general instances. We conjecture that the answer is yes.

Open Problem 4. *Is there a better than ϕ-competitive deterministic algorithm with ℓ-lookahead for general instances, where ℓ is a constant?*

A related question is how the properties of the plan need to be adjusted so that one can use them in the setting with lookahead. At the beginning of Section 2.7.1, we briefly sketched that the situation is more complicated than without lookahead.

We also argued that if packets have large spans, then we get the same lower bound of 1.25 for randomized algorithms against the oblivious adversary as without lookahead. We leave open whether it is possible to generalize the lower bound of $\frac{4}{3}$ against the adaptive adversary by Bieńkowski *et al.* [BCJ11]. However, the straightforward extension used in Section 2.7.3, i.e., stretching each time slot to k slots, thus multiplying packet spans by k, and having k copies of each packet in the strategy, does not work.

It would be interesting to see whether a combination of lookahead and randomization can improve the competitive ratio, at least on some restricted instances. Finally, an interesting research direction is to apply lookahead on other buffer management models, such as the model with a FIFO buffer of limited capacity or Item Collection.

Instances with large spans only. Note that most lower bounds for deterministic or randomized algorithms use just 2-bounded instances, unless they are specifically for the s-uniform case. Overall, it seems that if packets have large spans (and none of them has a small span), then it is harder for the adaptive adversary to create constructions forcing a high ratio.

However, in the aforementioned lower bounds of 1.25 against the oblivious adversary (with lookahead in Section 2.7.3 or for s-uniform instances by Chin *et al.* [CCF$^+$06]) all packets have large spans. Thus the barrier of 1.25 holds even on instances with large spans only.

Open Problem 5. *Is there a better than ϕ-competitive deterministic algorithm for instances in which the span of each packet is at least s, for some $s > 1$? What lower bound can be proven using instances with large spans only for the deterministic case and for randomized algorithms against the adaptive adversary?*

Higher bandwidth. Kesselman *et al.* [KLM$^+$04] proposed Bounded-Delay Packet Scheduling in the setting with bandwidth m, i.e., m packets can be sent in each step. However, not much was shown about this more general setting and the only other paper that studied bandwidth higher than 1 is by Chin *et al.* [CCF$^+$06] who designed an algorithm with competitive ratio which tends to $\frac{e}{e-1} \approx 1.582$ for $m \to \infty$. They noted that the lower bound of 1.25 for randomized algorithms against the oblivious adversary applies to higher bandwidth as well and this remains the best lower bound even for the deterministic case for $m > 1$.

Open Problem 6. *For the case of higher bandwidth $m > 1$, design an improved (deterministic or randomized) competitive algorithm and construct better lower bounds for deterministic algorithms and for randomized algorithms against the adaptive adversary.*

Weights decreasing over time. In Section 2.4 we propose a generalization of Bounded-Delay Packet Scheduling in which the weight of each packet decreases over time (and thus no explicit deadline is needed). As we argued, the greedy algorithm remains 2-competitive even in the non-clairvoyant setting, where the algorithm knows only the current weights, and there is no deterministic non-clairvoyant algorithm with a competitive ratio below 2, thus the simple greedy algorithm is optimal in such a model. This naturally leads to the following question.

Open Problem 7. *Is there a better than 2-competitive randomized or clairvoyant algorithm for the model with weights decreasing over time?*

3. Packet Scheduling under Adversarial Jamming

This chapter focuses on deterministic online algorithms with speedup for the Packet Scheduling under Adversarial Jamming problem. In particular, we describe and analyze a universal algorithm for the problem. The outline of the chapter is as follows:

- We start by defining the problem formally in Section 3.1, where we also describe the terminology used, the resource augmentation of speedup, and other preliminaries.
- In Section 3.2 we continue by a survey of the previous work.
- In Section 3.3 we outline the results contained in this chapter.
- In Section 3.4 we describe our algorithm and prove a few of its crucial properties.
- Section 3.5 contains a few examples of the algorithm's behavior, which also provide some intuition what are the tight instances for the algorithm (indeed, we give matching upper bounds in the following sections).
- In Section 3.6 we present a universal framework for analyzing our algorithm locally that we subsequently apply on general instances and on a few special types of instances.
- In Section 3.7 we prove that the algorithm is 1-competitive with speedup 4 which is the main result of this chapter. The proof is by a more sophisticated non-local analysis.
- In Section 3.8 we prove a lower bound of $\phi+1 \approx 2.618$ on speedup of 1-competitive deterministic algorithms.
- Finally, in Section 3.9 we discuss possible future research directions.

All results in this chapter are contained in the following paper:

[BJSV18] Martin Böhm, Łukasz Jeż, Jiří Sgall, and Pavel Veselý. On packet scheduling with adversarial jamming and speedup. In *Proc. of the 15th Workshop on Approximation and Online Algorithms (WAOA'17)*, volume 10787 of *LNCS*, pages 190–206, 2018.

The algorithm together with the local analysis framework was devised by Martin Böhm, Łukasz Jeż, and Jiří Sgall.

3.1 Problem Definition and Preliminaries

Problem statement. We define the Packet Scheduling under Adversarial Jamming problem as follows: The instance consists of a set of packets and a set of times in which instantaneous *jamming errors* (or *faults*) occur. Each packet p is specified by its *release time* $r(p) \geq 0$ and its *size* $\ell(p) \geq 0$. We assume that there are only k different packet sizes $\ell_1 < \ell_2 < \cdots < \ell_k$, for some constant k, and these sizes are assumed to be constant as well. Time is continuous, begins at 0 and ends at the *time horizon* T, which is a part of the instance. (Alternatively, one can say that after T, the jamming errors are so frequent that it is not possible to transmit any packet.)

A *schedule* is an assignment of a subset S of packets to time intervals $[t, t')$ such that

(i) the time intervals assigned to two packets in S are disjoint,

(ii) each packet $p \in S$ is assigned one interval $[t, t+\ell(p))$ for some $r(p) \leq t \leq T-\ell(p)$, and

(iii) no time of a jamming error is contained in the interior of a time interval assigned to a packet in S.

The objective is to compute a schedule which maximizes the total size of the packets in S (the completed packets).

For the online setting, when arrivals of packets and jamming errors are unknown in advance, we extend the definition of the schedule. Namely, the *online schedule* is a set S of packets and a set $\mathcal{I}$ of time intervals $[t, t')$ such that each interval is assigned to one packet (not necessarily in S) and the following holds:

(i) any two time intervals in $\mathcal{I}$ are disjoint,

(ii) each packet $p \in S$ is assigned one *sufficient* interval $[t_p, t_p + \ell(p))$ for some $r(p) \leq t_p \leq T - \ell(p)$,

(iii) any packet p may be assigned several *short* intervals $[t_i, t'_i)$ such that $t'_i - t_i < \ell(p)$, $r(p) \leq t_i < t'_i \leq T$, and at time t'_i there is a jamming error,

(iv) if $p \in S$, all short intervals $[t_i, t'_i)$ assigned to p are before its sufficient interval, i.e., $t'_i \leq t_p$, and

(v) no time of a jamming error is contained in the interior of *any* time interval in $\mathcal{I}$.

Other terminology and notation. If a packet p is assigned a time interval $[t, t')$, then we say that p is *scheduled* or *starts* at time t and that p is *running* at any time in (t, t'). Moreover, if $t' = t + \ell(p)$, then *p is completed* at t'; otherwise, there is a fault at time t'. Note that at the completion time of a packet p the algorithm may start another packet, or the completion time may be the time of a jamming error and still, p is successfully transmitted. Similarly, the algorithm may start transmitting a packet at the time of a fault, meaning that we assume that the jamming error takes no time.

We say that a packet p is pending for the algorithm at time t if $r(p) \leq t$, p is not completed before or at t and not started before t and still running. (Notice that in the online setting a pending packet may be started before t, but its transmission is interrupted and stopped due to a jamming error.)

For a set of packets P, let $\ell(P)$ denote the total size of all the packets in P. Let $L_{\mathsf{ALG}}(i, Y)$ denote the total size of packets of size ℓ_i completed by an algorithm ALG during a time interval Y. Similarly, $L_{\mathsf{ALG}}(\geq i, Y)$ and $L_{\mathsf{ALG}}(< i, Y)$ denote the total size of packets of size at least ℓ_i and less than ℓ_i, respectively, completed by an algorithm ALG during a time interval Y; formally, we define $L_{\mathsf{ALG}}(\geq i, Y) = \sum_{j=i}^{k} L_{\mathsf{ALG}}(j, Y)$ and $L_{\mathsf{ALG}}(< i, Y) = \sum_{j=1}^{i-1} L_{\mathsf{ALG}}(j, Y)$. We use the notation $L_{\mathsf{ALG}}(Y)$ with a single parameter to denote the size $L_{\mathsf{ALG}}(\geq 1, Y)$ of packets of all sizes completed by ALG during Y and the notation L_{ALG} without parameters to denote the size of all packets of all sizes completed by ALG at any time.

Online algorithms. In the online variant of Packet Scheduling under Adversarial Jamming, at time t only the packets with release time up to t are revealed and an online algorithm is aware only of jamming errors till time t. At time t, if no packet is running, the algorithm may start any pending packet. Moreover, preemption is not allowed, meaning that the algorithm cannot interrupt or even stop the current transmission of a packet and choose another packet to transmit. If a jamming error occurs and a packet p is running (and it is not completed yet), then the current transmission is lost completely, that is, p may be retransmitted now or at any time later, but the retransmission needs to send the whole packet p. Furthermore, the algorithm does not know the time horizon T.

We analyze the performance of an online algorithm in the worst-case, i.e., we use competitive analysis; see Chapter 1 for an introduction to competitive analysis. In particular, we use the asymptotic competitive ratio in which the additive constant may depend on packet sizes $\ell_1, \dots, \ell_k$ and their number k. Thus an algorithm ALG is R-competitive if there exists a constant A, possibly dependent on k and $\ell_1, \dots, \ell_k$, such that for any instance and its optimal schedule OPT we have $L_{\mathsf{OPT}} \le R \cdot L_{\mathsf{ALG}} + A$. We remark that in our analyses we show only a crude bound of $\mathcal{O}(k \cdot \ell_k)$ on A.

Remark. We note that the absolute competitive ratio, i.e., with no additive term, is not suitable for our problem. Specifically, using an example we show that a deterministic online algorithm can be (constant) competitive only if the additive term in the definition of the competitive ratio depends on the values of the packet sizes, even if there are only two packet sizes. Suppose that a packet of size ℓ arrives at time 0. If the algorithm starts transmitting it immediately at time 0, then at time $\varepsilon > 0$ a packet of size $\ell - 2\varepsilon$ arrives, the next fault is at time $\ell - \varepsilon$ and then the schedule ends, i.e., the time horizon is at $T = \ell - \varepsilon$. Thus the algorithm does not complete the packet of size ℓ, while the adversary completes a slightly smaller packet of size $\ell - 2\varepsilon$. Otherwise, the algorithm is idle till some time $\varepsilon > 0$, no other packet arrives and the next fault is at time ℓ, which is also the time horizon. In this case, the packet of size ℓ is completed in the optimal schedule, while the algorithm completes no packet again.

Resource augmentation: speedup. We focus on algorithms with resource augmentation, namely, on online algorithms that transmit packets $s \ge 1$ times faster than the offline optimum solution they are compared against; such an algorithm is often said to be speed-s, running at speed s, or having a speedup of s. This means that such an algorithm needs time ℓ/s to transmit the whole packet of size ℓ, i.e., the sufficient time interval in the online schedule is $[t, t+\ell/s)$. We denote an algorithm ALG with speedup $s \ge 1$ by $\mathsf{ALG}(s)$.

Special types of instances. An instance is *divisible* (or the packet sizes are divisible) if ℓ_i divides ℓ_{i+1} for $i = 1, \dots, k-1$. More generally, an instance is α-separated if $\ell_{i+1} \ge \alpha \cdot \ell_i$ for $i = 1, \dots, k-1$. Note that all divisible instances are 2-separated, but not vice versa.

Offline optimal schedule. Anta *et al.* [AGK$^+$16] show that computing an offline optimal schedule is strongly NP-hard. The proof is by a straightforward reduction from the 3-partition problem in which a set of $3m$ numbers needs to be partitioned into m triplets that all have the same sum. However, the number of different packet sizes in the resulting instance is large and we assume that this number is a constant.

We remark that the offline setting without release times resembles bin packing with bins of different sizes (each interval between two consecutive faults corresponds to a bin). For bin packing with a constant number of item sizes, Goemans and Rothvoß [GR14] recently gave a polynomial-time algorithm, which works even for different bin sizes. Nevertheless, due to release times of packets, we cannot apply their technique directly. It is thus open whether there is a polynomial-time algorithm to find the optimum for a constant number of sizes.

3.2 Previous Work and Related Models

Packet Scheduling under Adversarial Jamming was proposed by Anta, Georgiou, Kowalski, Widmer, and Zavou [AGK$^+$16], but adversarial jamming appears also in earlier works on wireless networks; see e.g. [RSSZ13] and references thereof. Anta *et al.* [AGK$^+$16] restricted their work to two packet sizes only, for which they provide matching upper and lower bounds: If $\gamma > 1$ denotes the ratio of the two sizes, then the optimal asymptotic competitive ratio for deterministic algorithms is $(\gamma + \lfloor\gamma\rfloor)/\lfloor\gamma\rfloor$, which is always in the range $[2, 3)$ and equals 2 iff $\gamma \in \mathbb{N}$.

They note that their lower bound strategy for two sizes applies to randomized algorithms as well, even against the oblivious adversary, which would imply that randomization provides no advantage. However, their argument works in the adaptive adversary model only, since in the lower bound strategy, the adversary needs to make decisions based on the previous behavior of the algorithm that depends on random bits. To our best knowledge, randomized algorithms for our problem were never considered for the more common oblivious adversary model, where the adversary needs to fix the instance in advance and cannot change it according to the decisions of the algorithm.

Anta *et al.* [AGK$^+$16] also discuss a few variants of the problem. First, they prove that with the deferred feedback mechanism, which notifies the online algorithm of an error in the current transmission only when it ends, no (deterministic) algorithm is competitive as its throughput will be 0, while the adversary is able to send arbitrarily many packets. This holds even if all packets have the same size and their arrivals are stochastic (i.e., not controlled by the adversary). They thus focus on the instantaneous feedback mechanism, which notifies the algorithm of an error immediately.

Finally, [AGK$^+$16] contains results for stochastic packet arrivals, where the competitive ratio depends also on the arrival rate of smaller packets; in particular, their algorithm CSL-Preamble achieves the optimal ratio for a wide range of distributions in which the rate is small enough, but still only for two packet sizes.

Jurdziński, Kowalski, and Loryś [JKL15] extended the results in [AGK$^+$16] for adversarial packet arrivals by designing an optimal deterministic algorithm for any constant number of packet sizes, which are assumed to be constant as well (thus the model in [JKL15] is exactly our model). In particular, its competitive ratio is $\max_{1\leq j<i\leq k}(\gamma_{i,j} + \lfloor\gamma_{i,j}\rfloor)/\lfloor\gamma_{i,j}\rfloor$, where $\gamma_{i,j} = \ell_i/\ell_j > 1$ is the ratio of the i-th size to the j-th size, thus it matches the aforementioned lower bound in [AGK$^+$16]. Note that the above formula gives 2 if and only if the packet sizes are divisible in which case their algorithm is simpler, and they show that it can be generalized to the setting with more parallel channels, which suffer jamming errors independently. Moreover, for the divisible case, they show that speed 2 is sufficient for 1-competitiveness, using a different algorithm.

In another work, Anta *et al.* [AGKZ18] consider popular scheduling algorithms, namely Longest In System (LIS, also known as FIFO), Shortest In System (SIS or LIFO), Largest Processing Time (LPT), and Shortest Processing Time (SPT). They analyze their performance under speed augmentation with respect to three efficiency measures, which they call *completed load*, *pending load*, and *latency*. The first is precisely the objective of our model, the second is the total size of the available but not yet completed packets (which we minimize in turn), and finally, the last one is the maximum time elapsed from a packet's arrival till the end of its successful transmission. We note that a 1-competitive algorithm (possibly with an additive constant) for any of the first two objectives is also 1-competitive for the other, but there is no similar relation for larger ratios.

Regarding latency, LIS is 1-competitive with speedup $\varrho := \ell_k/\ell_1$, otherwise, it has an unbounded competitive ratio as well as all other three algorithms with any speedup. If we want to achieve 1-competitiveness for completed load or for pending load, then speedup ϱ (or a bit larger) is sufficient and necessary for any of these algorithms. Since ϱ can be arbitrarily large, none of these algorithms performs very well. Additionally, for deterministic algorithms and for work-conserving (randomized) algorithms which transmit a packet whenever one is pending, they provide an example showing that with speedup 1 it is not possible to be competitive, however, the additive constant cannot depend on packet sizes and their number.

Recently, Kowalski, Wong, and Zavou [KWZ17] studied the effect of speedup on latency and pending load objectives in the case of two packet sizes only. They use two conditions on the speedup, defined in [AGKZ15], and show that if both hold, then there is no 1-competitive deterministic algorithm for either objective (but speedup must be below 2), while if one of the conditions is not satisfied, such an algorithm exists.

Multiple channels or machines. The problem we study was generalized to multiple communication channels (or machines, depending on particular application). The standard assumption, in communication jargon, is that the jamming errors on each channel are independent and that any packet can be transmitted on at most one channel at any time.

As mentioned above, for divisible instances, Jurdziński *et al.* [JKL15] extended their (optimal) 2-competitive algorithm without speedup to an arbitrary number of channels.

The setting with speedup on general instances was studied by Anta *et al.* [AGKZ15], who consider the objectives of minimizing the number or the total size of pending packets. They investigate what speedup is necessary and sufficient for 1-competitiveness with respect to either objective. Recall that 1-competitiveness for minimizing the total size of pending packets is equivalent to 1-competitiveness for our objective of maximizing the total size of completed packets. In particular, for either objective, Anta *et al.* [AGKZ15] obtain a tight bound of 2 on speedup for 1-competitiveness for two packet sizes. Moreover, they claim a 1-competitive algorithm with speedup $\frac{7}{2}$ for a constant number of sizes and pending (or completed) load, but the proof is incorrect; see Section 3.5 for a (single-channel) counterexample.

Georgio and Kowalski [GK15] consider the same problem in a distributed setting, distinguishing between different information models. As communication and synchronization pose new challenges, they restrict their attention to jobs of unit size only and no speedup. On top of efficiency measured by the number of pending jobs, they also consider the standard (in distributed systems) notions of correctness and fairness.

Zavou and Anta [ZA16] studied distributed computation of tasks dynamically injected to a system with a shared repository of pending tasks. Their objective was completed load and in particular, they proved that the upper bounds by Jurdziński *et al.* [JKL15] for k different sizes without speedup are possible in their distributed setting, thus matching the lower bound by Anta *et al.* [AGK+16]. Additionally, they also show a few negative results for particular algorithms with speedup.

Finally, Garncarek, Jurdziński, and Loryś [GJL17] consider "synchronized" parallel channels that all suffer errors at the same time. Their work distinguishes between "regular" jamming errors and "crashes", which also cause the algorithm's state to reset, losing any information stored about the past events. They proved that for two packet sizes the optimum competitive ratio of deterministic algorithms tends to $\frac{4}{3}$ for the former and to $\phi = \frac{1}{2}(\sqrt{5}+1) \approx 1.618$ for the latter setting as the number of channels tends to infinity.

We remark that the aforementioned works [AGK+16, KWZ17, AGKZ15, AGKZ18, ZA16] are contained in the thesis of Zavou [Zav16].

3.3 Contributions

Our major contribution is a uniform algorithm, called *PrudentGreedy* (PG) and described in Section 3.4. It works well in every setting which we show using a uniform analysis framework (Section 3.6). This contrasts with the results of Jurdziński *et* *al.* [JKL15] where each upper bound was attained by a dedicated algorithm with independently crafted analysis; in a sense, this means that their algorithms require the knowledge of speed they are running at. Moreover, algorithms in [JKL15] do require the knowledge of all admissible packet sizes. Our algorithm has the advantage that it is completely oblivious, i.e., requires no such knowledge. Furthermore, our algorithm is more appealing as it is significantly simpler and "work-conserving" or "busy", i.e., transmitting some packet whenever there is one pending, which is desirable in practice. In contrast, algorithms in [JKL15] can be unnecessarily idle if there is a small number of pending packets.

Our main result concerns the analysis of the general case with speedup where we show that speed 4 is sufficient for our algorithm PG to be 1-competitive; the proof is by a complex (non-local) charging argument described in Section 3.7.

However, we start by formulating a simpler local analysis framework in Section 3.6 which is very universal as we demonstrate by applying it to several settings. In particular, we prove that on general instances, PG achieves the optimal competitive ratio of 3 without speedup and we also get a trade-off between the competitive ratio and the speedup for speeds in $[1, 4)$; see Figure 3.1 for a graph of our bounds on the competitive ratio depending on the speedup.

To recover the 1-competitiveness at speed 2 and also 2-competitiveness at speed 1 for divisible instances, we have to modify our algorithm slightly as otherwise, we can guarantee 1-competitiveness for divisible instances only at speed 2.5 (see Section 3.6.2). This is to be expected as divisible instances are a very special case. The definition of the modified algorithm for divisible instances and its analysis by our local analysis framework is in Section 3.6.3.

On the other hand, we prove that our original algorithm is 1-competitive on far broader class of "well-separated" instances at sufficient speed: If the ratio between two successive packet sizes (in their sorted list) is no smaller than $\alpha \geq 1$, our algorithm is 1-competitive if its speed is at least S_α which is a non-increasing function of α such that $S_1 = 6$ and $\lim_{\alpha\to\infty} S_\alpha = 2$; see Section 3.6.2 for the precise definition of S_α. (Note that speed 4 is sufficient for 1-competitiveness, but having $S_1 = 6$ reflects the limits of the local analysis.)

In Section 3.5 we demonstrate that the analyses of our algorithm are mostly tight, i.e., that (a) on general instances, the algorithm is no better than $(1+2/s)$-competitive for $s < 2$ and no better than $4/s$-competitive for $s \in [2, 4)$, (b) on divisible instances, it is no better than $\frac{4}{3}$-competitive for $s < 2.5$, and (c) it is at least 2-competitive for $s < 2$, even for two divisible packet sizes (example (c) is in Section 3.6.3). See Figure 3.1 for a graph of our bounds. Note that we do not obtain tight bounds for $s \in [2, 4)$, but we conjecture that using an appropriately adjusted non-local analysis of Theorem 3.14 (which shows 1-competitiveness for $s = 4$), it is possible to show that the algorithm is $4/s$-competitive for $s \in [2, 4)$.

In Section 3.8 we complement these results with two lower bounds on the speed that is sufficient to achieve 1-competitiveness by a deterministic algorithm. The first one proves that even for two divisible packet sizes, speed 2 is required to attain 1-competitiveness, establishing optimality of our modified algorithm and that of Jurdziński *et al.* [JKL15] for the divisible case. The second lower bound strengthens the previous construction by showing that for non-divisible instances with more packet sizes, speed $\phi + 1 \approx 2.618$ is needed for 1-competitiveness. Both results hold even if all packets are released simultaneously.

We remark that Sections 3.6, 3.7, and 3.8 are independent on each other and can be read in any order. In particular, the reader may safely skip proofs for special instances in Section 3.6 (e.g., the divisible instances), and proceed to Section 3.7 with the main

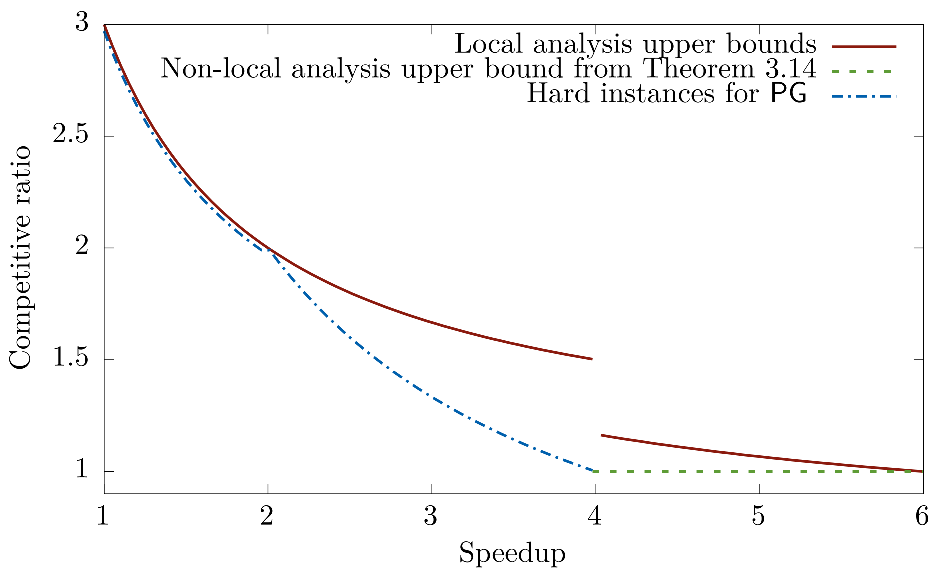


Figure 3.1: A graph of our upper and lower bounds on the competitive ratio of Algorithm $\mathsf{PG}(s)$, depending on the speedup s. The upper bounds are from Theorems 3.8 and 3.14 and the lower bounds are by hard instances from Section 3.5.

result, which is 1-competitiveness with speedup 4.

3.4 Algorithm PrudentGreedy (PG)

The general idea of the algorithm is that after each error, we start by transmitting packets of small sizes, only increasing the size of packets after a sufficiently long period of uninterrupted transmissions. It turns out that the right tradeoff is to transmit a packet only if it would have been transmitted successfully if started just after the last error. It is also crucial that the initial packet after each error has the right size, namely to ignore small packet sizes if the total size of remaining packets of those sizes is small compared to a larger packet that can be transmitted. In other words, the size of the first transmitted packet is larger than the *total* size of all pending smaller packets and we choose the largest such size. This guarantees that if no error occurs, all currently pending packets with size equal to or larger than the size of the initial packet are eventually transmitted before the algorithm starts a smaller packet.

We now give the description of our algorithm *PrudentGreedy* (PG) for general packet sizes, noting that the other algorithm for divisible sizes differs only slightly. We divide the run of the algorithm into phases. Each phase starts by an invocation of the initial step in which we need to carefully select a packet to transmit as discussed above. The phase ends by a fault, or when there is no pending packet, or when there are pending packets only of sizes larger than the total size of packets completed in the current phase. The periods of idle time, when no packet is pending, do not belong to any phase.

Formally, throughout the algorithm, t denotes the current time. The time t_B denotes the start of the current phase; initially $t_B = 0$. We set $\mathrm{rel}(t) = s \cdot (t - t_B)$. Since the algorithm does not insert unnecessary idle time, $\mathrm{rel}(t)$ denotes the amount of transmitted packets in the current phase. Note that we use $\mathrm{rel}(t)$ only when there is no packet running at time t, so there is no partially executed packet. Thus $\mathrm{rel}(t)$ can be thought of as a measure of time relative to the start of the current phase (scaled by the speed of the algorithm). Note also that the algorithm can evaluate $\mathrm{rel}(t)$ without knowing the speedup, as it can simply observe the total size of the transmitted packets. Let $P^{<i}$ denote the set of pending packets of sizes $\ell_1, \ldots, \ell_{i-1}$ at any given time.

```
Algorithm PrudentGreedy (PG):
  (1) If no packet is pending, stay idle until the next release time.
  (2) Let $i$ be the maximal $i \leq k$ such that there is a pending packet of size $\ell_i$ and
      $\ell(P^{<i}) < \ell_i$. Schedule a packet of size $\ell_i$ and set $t_B = t$.
  (3) Choose the maximum $i$ such that
          (i) there is a pending packet of size $\ell_i$,
          (ii) $\ell_i \leq \mathrm{rel}(t)$.
      Schedule a packet of size $\ell_i$. Repeat Step (3) as long as such $i$ exists.
  (4) If no packet satisfies the condition in Step (3), go to Step (1).
```

We first note that the algorithm is well-defined, i.e., that it is always able to choose a packet in Step (2) if it has any packets pending, and that if it succeeds in sending it, the length of thus started phase can be related to the total size of the packets completed in it.

Lemma 3.1. *In Step (2),* PG *always chooses some packet if it has any pending. Moreover, if* PG *completes the first packet in the phase, then $L_{\mathsf{PG}(s)}((t_B, t_E]) > s\cdot(t_E - t_B)/2$, where t_B denotes the start of the phase and t_E its end (by a fault or Step (4)).*

Proof. For the first property, note that a pending packet of the smallest size is eligible. For the second property, note that there is no idle time in the phase and that only the last packet chosen by PG in the phase may not complete due to a jam. By the condition in Step (3), the size of this jammed packet is no larger than the total size of all the packets PG previously completed in this phase (including the first packet chosen in Step (2)), which yields the bound. ☐

The following lemma shows a crucial property of the algorithm, namely that if packets of size ℓ_i are pending, the algorithm schedules packets of size at least ℓ_i most of the time. Its proof also explains the reasons behind our choice of the first packet in a phase in Step (2) of the algorithm.

Lemma 3.2. *Let u be a start of a phase in $\mathsf{PG}(s)$ and $t = u + \ell_i/s$.*

(i) *If a packet of size ℓ_i is pending at time u and no fault occurs in (u, t), then the phase does not end before t.*

(ii) *Suppose that $v > u$ is such that any time in $[u, v)$ a packet of size ℓ_i is pending and no fault occurs. Then the phase does not end in (u, v) and $L_{\mathsf{PG}(s)}(< i, (u, v]) < \ell_i + \ell_{i-1}$. (Recall that $\ell_0 = 0$.)*

Proof. (i) Suppose for a contradiction that the phase started at u ends at time $t' < t$. We have $\mathrm{rel}(t') < \mathrm{rel}(t) = \ell_i$. Let ℓ_j be the smallest packet size among the packets pending at t'. As there is no fault, the reason for a new phase has to be that $\mathrm{rel}(t') < \ell_j$, and thus Step (3) did not choose a packet to be scheduled. Also note that any packet started before t' was completed. This implies, first, that there is a pending packet of size ℓ_i, as there was one at time u and there was insufficient time to complete it, so j is well-defined and $j \leq i$. Second, all packets of sizes smaller than ℓ_j pending at u were completed before t', so their total size is at most $\mathrm{rel}(t') < \ell_j$. However, this contradicts the fact that the phase started by a packet smaller than ℓ_j at time u, as a pending packet of the smallest size equal to or larger than ℓ_j satisfied the condition in Step (2) at time u and a packet of size ℓ_i is pending at u. (Note that it is possible that no packet of size ℓ_j is pending at u.)

(ii) By (i), the phase that started at u does not end before time t if no fault happens. A packet of size ℓ_i is always pending by the assumption of the lemma, and it is always a valid choice of a packet in Step (3) from time t on. Thus, the phase that started at u does not end in (u, v), and moreover only packets of sizes at least ℓ_i are started in $[t, v)$. It follows that packets of sizes smaller than ℓ_i are started only before time t and their total size is thus less than $\text{rel}(t) + \ell_{i-1} = \ell_i + \ell_{i-1}$. The lemma follows. □

3.5 Examples for PrudentGreedy

Before analyzing the algorithm, we show some hard instances and obtain lower bounds on the performance of the algorithm, in some cases matching our upper bounds from later sections. At the same time, the examples are instructive in understanding some of the choices both in the analysis and in the design of the algorithm.

General Packet Sizes

Speeds below 2 We show an instance on which the performance of PG(s) matches the bound, that we give in Theorem 3.8.

Remark. PG(s) has competitive ratio at least $1 + 2/s$ for $s < 2$.

Proof. Choose a large enough integer N. At time 0 the following packets are released: $2N$ packets of size 1, one packet of size 2 and N packet of size $4/s - \varepsilon$ for a small enough $\varepsilon > 0$ such that it holds that $2 < 4/s - \varepsilon$. These are all packets in the instance.

First there are N phases, each of length $4/s - \varepsilon$ and ending by a fault. OPT completes a packet of size $4/s - \varepsilon$ in each phase, while PG(s) completes 2 packets of size 1 and then it starts a packet of size 2 which is not finished.

Then there is a fault every 1 unit of time, so that OPT completes all packets of size 1, while the algorithm has no pending packet of size 1 and as $s < 2$ the length of the phase is not sufficient to finish a longer packet.

Overall, OPT completes packets of total size $2 + 4/s - \varepsilon$ per phase, while the algorithm completes packets of total size only 2 per phase. The ratio thus tends to $1 + 2/s$ as $\varepsilon \to 0$. □

Speeds between 2 and 4 Now we show an instance which proves that PG(s) is not 1-competitive for $s < 4$. In particular, this implies that the speed sufficient for 1-competitiveness in Theorem 3.14, which we prove later, cannot be improved.

Remark. PG(s) has competitive ratio at least $4/s > 1$ for $s \in [2, 4)$.

Proof. Choose a large enough integer y. There will be four packet sizes: $1, x, y$ and z such that $1 < x < y < z$, $z = x + y - 1$, and $x = y \cdot (s - 2)/2 + 2$; as $s \geq 2$ it holds that $x > 1$ and as $s < 4$ we have $x \leq y - 1$ for a large enough y.

We will have N phases again. At time 0 the adversary releases all $N(y-1)$ packets of size 1, all N packets of size y and a single packet of size z (never completed by either OPT or PG(s)), whereas the packets of size x are released one per phase.

In each phase PG(s) completes, in this order: $y - 1$ packets of size 1 and then a packet of size x, which has arrived just after the $y - 1$ packets of size 1 are completed. Next, it will start a packet of size z and fail due to a jam. We show that OPT will complete a packet of size y. To this end, it is required that $y < 2(x + y - 1)/s$, or equivalently $x > y \cdot (s - 2)/2 + 1$ which holds by the choice of x.

After these N phases, we will have jams every 1 unit of time, so that OPT can complete all the $N(y-1)$ packets of size 1, while PG(s) will be unable to complete any packet (of size y or larger). The ratio per phase is

$$\frac{\mathsf{OPT}}{\mathsf{PG}(s)} = \frac{y-1+y}{y-1+x} = \frac{2y-1}{y-1+\frac{y\cdot(s-2)}{2}+2} = \frac{2y-1}{\frac{y\cdot s}{2}+1}$$

which tends to $4/s$ as $y \to \infty$. □

This example also disproves the claim of Anta *et al.* [AGKZ15] that their (m,β)-LAF algorithm is 1-competitive at speed 3.5, even for one channel, i.e., $m=1$, where it behaves almost exactly as PG(s) — the sole difference is that LAF starts a phase by choosing a “random” packet. As this algorithm is deterministic, we understand this to mean “arbitrary”, so in particular the same as chosen by PG(s).

Divisible Case

We give an example that shows that PG is not very good for divisible instances, in particular, it is not 1-competitive for any speed $s < 2.5$.

Remark. PG(s) has competitive ratio at least $\frac{4}{3}$ on divisible instances if $s < 2.5$.

Proof. We use packets of sizes 1, ℓ, and 2ℓ and we take ℓ sufficiently large compared to the given speed or competitive ratio. There are many packets of size 1 and 2ℓ available at the beginning, the packets of size ℓ arrive at specific times where PG schedules them immediately.

The faults occur at times divisible by 2ℓ, so the optimum schedules one packet of size 2ℓ in each phase between two faults. We have N such phases, $N(2\ell-1)$ packets of size 1 and N packets of size 2ℓ available at the beginning. In each phase, $\mathsf{PG}(s)$ schedules $2\ell-1$ packets of size 1, then a packet of size ℓ arrives and is scheduled, and then a packet of size 2ℓ is scheduled. The algorithm would need speed $2.5-1/(2\ell)$ to complete it. So, for ℓ large, the algorithm completes only packets of total size $3\ell-1$ per phase. After these N phases, we have faults every 1 unit of time, so the optimum schedules all packets of size 1, but the algorithm has no packet of size 1 pending and it is unable to finish a longer packet. The optimum thus finishes all packets 2ℓ plus all small packets, a total of $4\ell-1$ per phase. Thus the ratio tends to $\frac{4}{3}$ as $\ell \to \infty$. □

3.6 Local Analysis and Results

In this section we formulate a general method for analyzing our algorithms by comparing locally within each phase the size of “large” packets completed by the algorithm and by the adversary. This method simplifies a complicated induction used in [JKL15], letting us obtain the same upper bounds of 2 and 3 on competitiveness for divisible and unrestricted packet sizes, respectively, at no speedup, as well as several new results for the non-divisible cases included in this section. In Section 3.7, we use a more complex charging scheme to obtain our main result. We postpone the use of local analysis for the divisible case to Section 3.6.3.

For the analysis, let $s \geq 1$ be the speedup. We fix an instance and its schedules for $\mathsf{PG}(s)$ and OPT.

3.6.1 Critical Times and Master Theorem

The common scheme is the following. We carefully define a sequence of critical times $C_k \leq C_{k-1} \leq \cdots \leq C_1 \leq C_0$, where $C_0 = T$ is the end of the schedule, satisfying two properties: (1) till time C_i the algorithm has completed almost all pending packets of size ℓ_i released before C_i and (2) in $(C_i, C_{i-1}]$, a packet of size ℓ_i is always pending. Properties (1) and (2) allow us to relate $L_{\mathsf{OPT}}(i, (0, C_i])$ and $L_{\mathsf{OPT}}(\geq i, (C_i, C_{i-1}])$, respectively, to their "PG counterparts". As each packet completed by OPT belongs to exactly one of these sets, summing the bounds gives the desired results; see Figure 3.2 for an illustration. These two facts together imply R-competitiveness of the algorithm for appropriate R and speed s.

We first define the notion of i-good times so that they satisfy property (1), and then choose the critical times among their suprema so that those satisfy property (2) as well.

Definition 3.3. *Let $s \geq 1$ be the speedup. For $i = 1, \ldots k$, time t is called i-good if one of the following conditions holds:*

(i) At time t, Algorithm $\mathsf{PG}(s)$ starts a new phase by scheduling a packet of size larger than ℓ_i, or

(ii) at time t, no packet of size ℓ_i is pending for $\mathsf{PG}(s)$, or

(iii) $t = 0$.

We define critical times $C_0, C_1, \ldots, C_k$ iteratively as follows:

- *$C_0 = T$, i.e., it is the end of the schedule.*
- *For $i = 1, \ldots, k$, C_i is the supremum of i-good times t such that $t \leq C_{i-1}$.*

Note that all C_i's are defined and $C_i \geq 0$, as time $t = 0$ is i-good for all i. The choice of C_i implies that each C_i is of one of the three types (the types are not disjoint):

- C_i is i-good and a phase starts at C_i (this includes $C_i = 0$),
- C_i is i-good and $C_i = C_{i-1}$, or
- there exists a packet of size ℓ_i pending at C_i, however, any such packet was released at C_i.

If the first two options do not apply, then the last one is the only remaining possibility (as otherwise some time in the non-empty interval $(C_i, C_{i-1}]$ would be i-good); in this case, C_i is not i-good, but only the supremum of i-good times.

First we bound the total size of packets of size ℓ_i completed before C_i; the proof actually only uses the fact that each C_i is the supremum of i-good times and justifies the definition above.

Lemma 3.4. *Let $s \geq 1$ be the speedup. Then, for any i, it holds $L_{\mathsf{OPT}}(i, (0, C_i]) \leq L_{\mathsf{PG}(s)}(i, (0, C_i]) + \ell_k$.*

Proof. If C_i is i-good and satisfies condition (i) in Definition 3.3, then by the description of Step (2) of the algorithm, the total size of pending packets of size ℓ_i is less than the size of the scheduled packet, which is at most ℓ_k and the lemma follows.

In all the remaining cases it holds that $\mathsf{PG}(s)$ has completed all the jobs of size ℓ_i released before C_i, thus the inequality holds trivially even without the additive term. □

Our remaining goal is to bound $L_{\mathsf{OPT}}(\geq i, (C_i, C_{i-1}])$. We divide $(C_i, C_{i-1}]$ into i-segments by the faults. We prove the bounds separately for each i-segment. One important fact is that for the first i-segment we use only a loose bound, as we can use the additive constant. The critical part is then the bound for i-segments started by a fault, this part determines the competitive ratio and is different for each case. We summarize the general method by the following definition and master theorem.

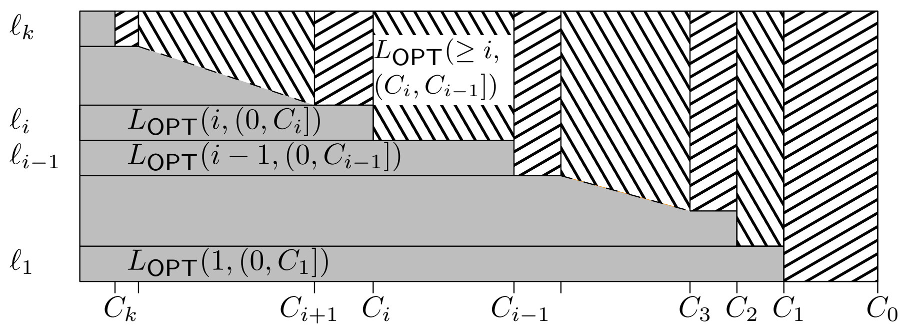

ℓ_k
ℓ_i
ℓ_{i-1}
ℓ_1
$L_{\mathsf{OPT}}(\geq i,$
$(C_i, C_{i-1}])$
$L_{\mathsf{OPT}}(i,(0,C_i])$
$L_{\mathsf{OPT}}(i-1,(0,C_{i-1}])$
$L_{\mathsf{OPT}}(1,(0,C_1])$
C_k
C_{i+1}
C_i
C_{i-1}
C_3
C_2
C_1
C_0

Figure 3.2: An illustration of dividing the schedule of OPT in the local analysis, i.e., dividing the (total size of) packets completed by OPT into $L_{\mathsf{OPT}}(i,(0,C_i])$ and $L_{\mathsf{OPT}}(\geq i,(C_i,C_{i-1}])$ for $i=1,\ldots,k$. Rows correspond to packet sizes and the X-axis to time. Gray horizontal rectangles thus correspond to $L_{\mathsf{OPT}}(i,(0,C_i])$, i.e., these rectangles represent the time interval $(0,C_i]$ and packets of size ℓ_i completed by OPT in $(0,C_i]$, whereas hatched rectangles correspond to $L_{\mathsf{OPT}}(\geq i,(C_i,C_{i-1}])$.

Definition 3.5. *The interval $(u,v]$ is called the initial i-segment if $u=C_i$ and v is either C_{i-1} or the first time of a fault after u, whichever comes first.*

The interval $(u,v]$ is called a proper i-segment if $u\in(C_i,C_{i-1})$ is a time of a fault and v is either C_{i-1} or the first time of a fault after u, whichever comes first.

Note that there is no i-segment if $C_{i-1}=C_i$.

Theorem 3.6 (Master Theorem). *Let $s\geq 1$ be the speedup. Suppose that for $R\geq 1$ both of the following hold:*

1. *For each $i=1,\ldots,k$ and each proper i-segment $(u,v]$ with $v-u\geq \ell_i$, it holds that*

$$(R-1)L_{\mathsf{PG}(s)}((u,v])+L_{\mathsf{PG}(s)}(\geq i,(u,v]) \;\geq\; L_{\mathsf{OPT}}(\geq i,(u,v])\,. \tag{3.1}$$

2. *For the initial i-segment $(u,v]$, it holds that*

$$L_{\mathsf{PG}(s)}(\geq i,(u,v]) \;>\; s(v-u)-4\ell_k\,. \tag{3.2}$$

Then $\mathsf{PG}(s)$ is R-competitive.

Proof. First note that for a proper i-segment $(u,v]$, u is a fault time. Thus if $v-u<\ell_i$, then $L_{\mathsf{OPT}}(\geq i,(u,v])=0$ and (3.1) is trivial. It follows that (3.1) holds even without the assumption $v-u\geq\ell_i$.

Now consider the initial i-segment $(u,v]$. We have $L_{\mathsf{OPT}}(\geq i,(u,v])\leq \ell_k+v-u$, as at most a single packet started before u can be completed. Combining this with (3.2) and using $s\geq 1$, we get $L_{\mathsf{PG}(s)}(\geq i,(u,v]) \;>\; s(v-u)-4\ell_k \;\geq\; v-u-4\ell_k \;\geq\; L_{\mathsf{OPT}}(\geq i,(u,v])-5\ell_k$.

Summing this with (3.1) for all proper i-segments and using $R\geq 1$ we get

$$\begin{aligned}(R-1)L_{\mathsf{PG}(s)}((C_i,C_{i-1}])+L_{\mathsf{PG}(s)}(\geq i,(C_i,C_{i-1}])+5\ell_k&\\ \geq L_{\mathsf{OPT}}(\geq i,(C_i,C_{i-1}])\,.&\end{aligned} \tag{3.3}$$

Note that for $C_i=C_{i-1}$, Equation (3.3) holds trivially.

To complete the proof, note that each completed job in the optimum contributes to exactly one among the $2k$ terms $L_{\mathsf{OPT}}(\geq i, (C_i, C_{i-1}])$ and $L_{\mathsf{OPT}}(i, (0, C_i])$; similarly for $L_{\mathsf{PG}(s)}$. Thus by summing both (3.3) and Lemma 3.4 for all $i = 1, \dots, k$ we obtain

$$\begin{aligned}
L_{\mathsf{OPT}} &= \sum_{i=1}^{k} L_{\mathsf{OPT}}\,(\geq i, (C_i, C_{i-1}]) + \sum_{i=1}^{k} L_{\mathsf{OPT}}(i, (0, C_i]) \\
&\leq \sum_{i=1}^{k} \Big((R-1) L_{\mathsf{PG}(s)}((C_i, C_{i-1}]) + \Big(L_{\mathsf{PG}(s)}(\geq i, (C_i, C_{i-1}]) + 5\ell_k \Big) \Big) \\
&\quad + \sum_{i=1}^{k} \Big(L_{\mathsf{PG}(s)}(i, (0, C_i]) + \ell_k \Big) \\
&\leq (R-1) L_{\mathsf{PG}(s)} + L_{\mathsf{PG}(s)} + 6k\ell_k \;=\; R \cdot L_{\mathsf{PG}(s)} + 6k\ell_k\,.
\end{aligned}$$

The theorem follows. □

3.6.2 Local Analysis of PrudentGreedy (PG)

The first part of the following lemma implies the condition (3.2) for the initial i-segments in all cases. The second part of the lemma is the base of the analysis of a proper i-segment, which is different in each situation.

Lemma 3.7. (i) *If $(u, v]$ is the initial i-segment, then $L_{PG(s)}(\geq i, (u, v]) > s(v - u) - 4\ell_k$.*

(ii) *If $(u, v]$ is a proper i-segment and $v - u \geq \ell_i$ then $L_{PG(s)}((u, v]) > s(v - u)/2$ and $L_{PG(s)}(\geq i, (u, v]) > s(v - u)/2 - \ell_i - \ell_{i-1}$. (Recall that $\ell_0 = 0$.)*

Proof. (i) If the phase that starts at u or contains u ends before v, let u' be its end; otherwise let $u' = u$. We have $u' \leq u + \ell_i/s$, as otherwise any packet of size ℓ_i, pending throughout the i-segment by definition, would be an eligible choice in Step (3) of the algorithm, and the phase would not end before v. Using Lemma 3.2(ii), we have $L_{\mathsf{PG}(s)}(< i, (u', v]) < \ell_i + \ell_{i-1} < 2\ell_k$. Since at most one packet at the end of the segment is unfinished, we have $L_{\mathsf{PG}(s)}(\geq i, (u, v]) \geq L_{\mathsf{PG}(s)}(\geq i, (u', v]) > s(v - u') - 3\ell_k \geq s(v - u) - 4\ell_k$.

(ii) Let $(u, v]$ be a proper i-segment. Thus u is a start of a phase that contains at least the whole interval $(u, v]$ by Lemma 3.2(ii). By the definition of C_i, u is not i-good, so the phase starts by a packet of size at most ℓ_i. If $v - u \geq \ell_i$ then the first packet finishes (as $s \geq 1$) and thus $L_{\mathsf{PG}(s)}((u, v]) > s(v - u)/2$ by Lemma 3.1. The total size of completed packets smaller than ℓ_i is less than $\ell_i + \ell_{i-1}$ by Lemma 3.2(ii), and thus $L_{\mathsf{PG}(s)}(\geq i, (u, v]) > s(v - u)/2 - \ell_i - \ell_{i-1}$. □

General Packet Sizes

The next theorem gives a tradeoff of the competitive ratio of $\mathsf{PG}(s)$ and the speedup s using our local analysis. While Theorem 3.14 shows that $\mathsf{PG}(s)$ is 1-competitive for $s \geq 4$, here we give a weaker result that reflects the limits of the local analysis. However, for $s = 1$ our local analysis is tight as already the lower bound from [AGK+16] shows that no algorithm is better than 3-competitive (for packet sizes 1 and $2 - \varepsilon$). Moreover, the bound for $s < 2$ is tight for our algorithm, as shown by the first example in Section 3.5. See Figure 3.1 for an illustration of our upper and lower bounds on the competitive ratio of $\mathsf{PG}(s)$ on general instances.

Theorem 3.8. *$PG(s)$ is R_s-competitive where:*

$R_s = 1 + 2/s$ for $s \in [1, 4)$,

$R_s = 2/3 + 2/s$ *for* $s \in [4, 6)$*, and*
$R_s = 1$ *for* $s \geq 6$.

Proof. Lemma 3.7(i) implies the condition (3.2) for the initial i-segments. We now prove (3.1) for any proper i-segment $(u, v]$ with $v - u \geq \ell_i$ and appropriate R. The bound then follows by the Master Theorem.

Since there is a fault at time u, we have $L_{\mathsf{OPT}}(\geq i, (u, v]) \leq v - u$.

For $s \geq 6$, Lemma 3.7(ii) implies

$$\begin{aligned} L_{\mathsf{PG}(s)}(\geq i, (u, v]) &> s(v - u)/2 - 2\ell_i \\ &\geq 3(v - u) - 2(v - u) = v - u \geq L_{\mathsf{OPT}}(\geq i, (u, v]), \end{aligned}$$

which is (3.1) for $R = 1$.

For $s \in [4, 6)$, by Lemma 3.7(ii) we have $L_{\mathsf{PG}(s)}((u, v]) > s(v - u)/2$ and by multiplying it by $(2/s - 1/3)$ we obtain

$$\left(\frac{2}{s} - \frac{1}{3}\right) \cdot L_{\mathsf{PG}(s)}((u, v]) > \left(1 - \frac{s}{6}\right)(v - u).$$

Thus to prove (3.1) for $R = 2/3 + 2/s$, it suffices to show that

$$L_{\mathsf{PG}(s)}(\geq i, (u, v]) > \frac{s}{6}(v - u),$$

as clearly $v - u \geq L_{\mathsf{OPT}}(\geq i, (u, v])$. The remaining inequality again follows from Lemma 3.7(ii), but we need to consider two cases:

If $(v - u) \geq \frac{6}{s}\ell_i$, then

$$L_{\mathsf{PG}(s)}(\geq i, (u, v]) > \frac{s}{2}(v - u) - 2\ell_i \geq \frac{s}{2}(v - u) - \frac{s}{3}(v - u) = \frac{s}{6}(v - u).$$

On the other hand, if $(v - u) < \frac{6}{s}\ell_i$, then using $s \geq 4$ as well,

$$L_{\mathsf{PG}(s)}(\geq i, (u, v]) > \frac{s}{2}(v - u) - 2\ell_i \geq 0,$$

therefore $\mathsf{PG}(s)$ completes a packet of size at least ℓ_i which implies

$$L_{\mathsf{PG}(s)}(\geq i, (u, v]) \geq \ell_i > \frac{s}{6}(v - u),$$

concluding the case of $s \in [4, 6)$.

For $s \in [1, 4)$, by Lemma 3.7(ii) we get $(2/s) \cdot L_{\mathsf{PG}(s)}((u, v]) > v - u \geq L_{\mathsf{OPT}}(\geq i, (u, v])$, which implies (3.1) for $R = 1 + 2/s$. □

Well-separated Packet Sizes

We can obtain better bounds on the speedup necessary for 1-competitiveness if the packet sizes are sufficiently different. Namely, we call the packet sizes $\ell_1, \dots, \ell_k$ α-*separated* if $\ell_i \geq \alpha \ell_{i-1}$ holds for $i = 2, \dots, k$.

Next, we show that for α-separated packet sizes, $\mathsf{PG}(S_\alpha)$ is 1-competitive for the following S_α. We define

$$\alpha_0 = \frac{1}{2} + \frac{1}{6}\sqrt{33} \approx 1.46\,, \text{ which is the positive root of } 3\alpha^2 - 3\alpha - 2.$$

$$\alpha_1 = \frac{3 + \sqrt{17}}{4} \approx 1.78\,, \text{ which is the positive root of } 2\alpha^2 - 3\alpha - 1.$$

$$S_\alpha = \begin{cases} \dfrac{4\alpha + 2}{\alpha^2} & \text{for } \alpha \in [1, \alpha_0], \\ 3 + \frac{1}{\alpha} & \text{for } \alpha \in [\alpha_0, \alpha_1), \text{ and} \\ 2 + \frac{2}{\alpha} & \text{for } \alpha \geq \alpha_1. \end{cases}$$

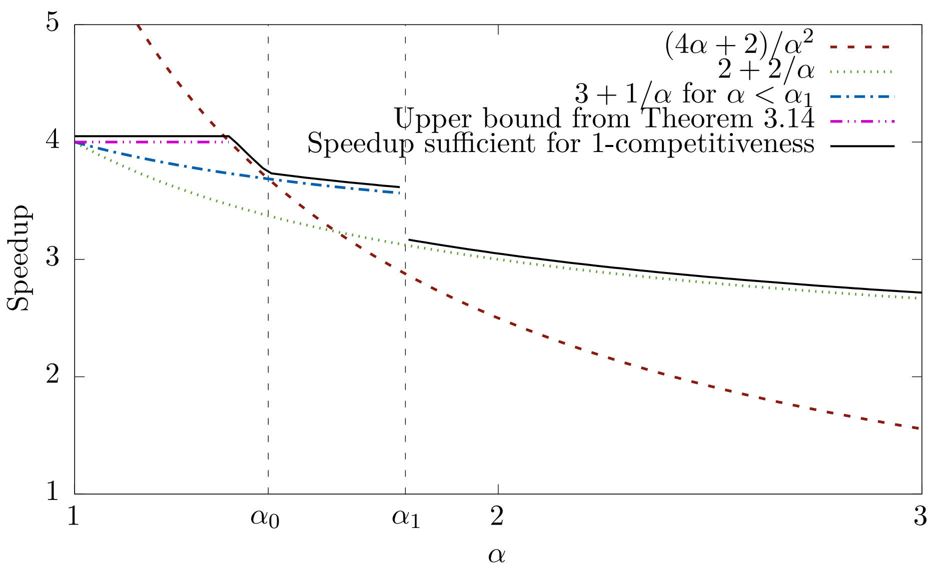


Figure 3.3: A graph of S_α and the bounds on the speedup that we use in Theorem 3.9. Note that in the graph we also use Theorem 3.14 for 1-competitiveness with speed 4 (for any α), but in the definition of S_α we do not take it into account.

See Figure 3.3 for a graph of S_α and all the bounds on it that we use. The value of α_0 is chosen as the point where $(4\alpha+2)/\alpha^2 = 3 + 1/\alpha$. The value of α_1 is chosen as the point from which the argument in case (viii) of the proof below works, which allows for a better result for $\alpha \ge \alpha_1$. If $s \ge S_\alpha$ then $s \ge (4\alpha+2)/\alpha^2$ and $s \ge 2 + 2/\alpha$ for all α and also $s \ge 3 + 1/\alpha$ for $\alpha < \alpha_1$; these facts follow from inspection of the functions and are useful for the analysis.

Note that S_α is decreasing in α, with a single discontinuity at α_1. We have $S_1 = 6$, matching the upper bound for 1-competitiveness using local analysis. We have $S_2 = 3$, i.e., $\mathsf{PG}(3)$ is 1-competitive for 2-separated packet sizes, which includes the case of divisible packet sizes (however, only $s \ge 2.5$ is needed in the divisible case, as we show later). The limit of S_α for $\alpha \to +\infty$ is 2. For $\alpha < (1+\sqrt{3})/2 \approx 1.366$, we get $S_\alpha > 4$, while Theorem 3.14 shows that $\mathsf{PG}(s)$ is 1-competitive for $s \ge 4$; the weaker result of Theorem 3.9 below reflect the limits of the local analysis.

Theorem 3.9. *Let $\alpha > 1$. If the packet sizes are α-separated, then $\mathsf{PG}(s)$ is 1-competitive for any $s \ge S_\alpha$.*

Proof. Lemma 3.7(i) implies (3.2). We now prove for any proper i-segment $(u, v]$ with $v - u \ge \ell_i$ that

$$L_{\mathsf{PG}(s)}(\ge i, (u,v]) \ge L_{\mathsf{OPT}}(\ge i, (u,v])\,, \tag{3.4}$$

which is (3.1) for $R = 1$. The bound then follows by the Master Theorem.

Let $X = L_{\mathsf{OPT}}(\ge i, (u,v])$. Note that $X \le v - u$.

Lemma 3.7(ii) together with $\ell_{i-1} \le \ell_i/\alpha$ gives $L_{\mathsf{PG}(s)}(\ge i, (u,v]) > M$ for $M = sX/2 - (1 + 1/\alpha)\ell_i$.

We use the fact that both X and $L_{\mathsf{PG}(s)}(\ge i, (u,v])$ are sums of some packet sizes ℓ_j, $j \ge i$, and thus only some of the values are possible. However, the situation is quite complicated, as for example ℓ_{i+1}, ℓ_{i+2}, $2\ell_i$, $\ell_i + \ell_{i+1}$ are possible values, but their ordering may vary.

We distinguish several cases based on X and α. We note in advance that the first five cases suffice for $\alpha < \alpha_1$; only after completing the proof for $\alpha < \alpha_1$, we analyze the additional cases needed for $\alpha \geq \alpha_1$.

Case i: $X = 0$. Then (3.4) is trivial.

Case ii: $X = \ell_i$. Using $s \geq 2 + 2/\alpha$, we obtain $M \geq (1 + 1/\alpha)\ell_i - (1 + 1/\alpha)\ell_i = 0$. Thus $L_{\mathsf{PG}(s)}(\geq i, (u, v]) > M \geq 0$ which implies $L_{\mathsf{PG}(s)}(\geq i, (u, v]) \geq \ell_i = X$ and (3.4) holds.

Case iii: $X = \ell_{i+1}$ and $\ell_{i+1} \leq 2\ell_i$. Using $s \geq (4\alpha + 2)/\alpha^2$ and $X = \ell_{i+1} \geq \alpha\ell_i$, we obtain

$$M \geq \frac{s\ell_{i+1}}{2} - \left(1 + \frac{1}{\alpha}\right)\ell_i \geq \left(2 + \frac{1}{\alpha}\right)\ell_i - \left(1 + \frac{1}{\alpha}\right)\ell_i = \ell_i\,.$$

Thus $L_{\mathsf{PG}(s)}(\geq i, (u, v]) > \ell_i$ which together with $\ell_{i+1} \leq 2\ell_i$ implies $L_{\mathsf{PG}(s)}(\geq i, (u, v]) \geq \ell_{i+1} = X$ and (3.4) holds.

Case iv: $X \geq \alpha^2\ell_i$. (Note that this includes all cases when a packet of size at least ℓ_{i+2} contributes to X.) We first show that $s \geq 2(1 + 1/\alpha^2 + 1/\alpha^3)$ by straightforward calculations with the golden ratio ϕ:

- If $\alpha \leq \phi$, we have

$$s \geq \frac{4\alpha + 2}{\alpha^2} = 2\left(\frac{2}{\alpha} + \frac{1}{\alpha^2}\right) \geq 2\left(1 + \frac{1}{\alpha^2} + \frac{1}{\alpha^3}\right)\,,$$

  where we use $2/\alpha \geq 1 + 1/\alpha^3$ or equivalently $\alpha^3 + 1 - 2\alpha^2 \leq 0$, which is true as

$$\alpha^3 + 1 - 2\alpha^2 = \alpha^3 - \alpha^2 + 1 - \alpha^2 = \alpha^2(\alpha - 1) - (\alpha + 1)(\alpha - 1) = (\alpha - 1)(\alpha^2 - \alpha - 1) \leq 0\,,$$

  where the last inequality holds for $\alpha \in (1, \phi)$.
- If on the other hand $\alpha \geq \phi$, then $s \geq 2(1 + 1/\alpha) \geq 2(1 + 1/\alpha^2 + 1/\alpha^3)$, as $1/\alpha \geq 1/\alpha^2 + 1/\alpha^3$ holds for $\alpha \geq \phi$.

Now we obtain

$$\begin{aligned} M - X &\geq \left(\frac{s}{2} - 1\right)X - \left(1 + \frac{1}{\alpha}\right)\ell_i \\ &\geq \left(1 + \frac{1}{\alpha^2} + \frac{1}{\alpha^3} - 1\right)X - \left(1 + \frac{1}{\alpha}\right)\ell_i \\ &\geq \left(\frac{1}{\alpha^2} + \frac{1}{\alpha^3}\right)\alpha^2\ell_i - \left(1 + \frac{1}{\alpha}\right)\ell_i = 0\ , \end{aligned}$$

and (3.4) holds.

Case v: $X \geq 2\ell_i$ and $\alpha < \alpha_1$. (Note that this includes all cases when at least two packets contribute to X, but we use it only if $\alpha < \alpha_1$.) Using $s \geq 3 + 1/\alpha$ we obtain

$$M - X \geq \left(\frac{1}{2}\left(3 + \frac{1}{\alpha}\right) - 1\right)X - \left(1 + \frac{1}{\alpha}\right)\ell_i \geq \frac{1}{2}\left(1 + \frac{1}{\alpha}\right)2\ell_i - \left(1 + \frac{1}{\alpha}\right)\ell_i = 0\ ,$$

and (3.4) holds.

Proof for $\alpha < \alpha_1$: We now observe that for $\alpha < \alpha_1$, we have exhausted all the possible values of X. Indeed, if (v) does not apply, then at most a single packet contributes to X, and one of the cases (i)-(iv) applies, as (iv) covers the case when $X \geq \ell_{i+2}$, and as $X = \ell_{i+1}$ is covered by (iii) or (v). Thus (3.4) holds and the proof is complete.

Proof for $\alpha \geq \alpha_1$: We now analyze the remaining cases for $\alpha \geq \alpha_1$.

Case vi: $X \geq (\alpha + 1)\ell_i$. (Note that this includes all cases when two packets not both of size ℓ_i contribute to X.) Using $s \geq 2 + 2/\alpha$ we obtain

$$M - X \geq \left(1 + \frac{1}{\alpha} - 1\right)X - \left(1 + \frac{1}{\alpha}\right)\ell_i \geq \frac{1}{\alpha}(\alpha + 1)\ell_i - \left(1 + \frac{1}{\alpha}\right)\ell_i = 0$$

and (3.4) holds.

Case vii: $X = n \cdot \ell_i < (\alpha+1)\ell_i$ for some $n = 2, 3, \ldots$. Since $\alpha > \alpha_1 > \phi$, we have $\ell_{i+1} > \ell_i + \ell_{i-1}$. This implies that the first packet of size at least ℓ_i that is scheduled in the phase has size equal to ℓ_i by the condition in Step (3) of the algorithm. Thus, if also a packet of size larger than ℓ_i contributes to $L_{\mathsf{PG}(s)}(\geq i, (u,v])$, we have

$$L_{\mathsf{PG}(s)}(\geq i, (u,v]) \geq \ell_{i+1} + \ell_i \geq (\alpha+1)\ell_i > X$$

by the case condition and (3.4) holds. Otherwise $L_{\mathsf{PG}(s)}(\geq i, (u,v])$ is a multiple of ℓ_i. Using $s \geq 2 + 2/\alpha$, we obtain

$$M \geq \left(1 + \frac{1}{\alpha}\right) n \cdot \ell_i - \left(1 + \frac{1}{\alpha}\right) \ell_i \geq (n-1)\left(1 + \frac{1}{\alpha}\right)\ell_i > (n-1)\ell_i\,.$$

This, together with divisibility by ℓ_i implies $L_{\mathsf{PG}(s)}(\geq i, (u,v]) \geq n \cdot \ell_i = X$ and (3.4) holds again.

Case viii: $X = \ell_{i+1}$ and $\ell_{i+1} > 2\ell_i$. We distinguish two subcases depending on the size of the unfinished packet of PG(s) in this phase.

If the unfinished packet has size at most ℓ_{i+1}, the size of the completed packets is bounded by

$$L_{\mathsf{PG}(s)}((u,v]) > sX - \ell_{i+1} = (s-1)\ell_{i+1} \geq \left(1 + \frac{2}{\alpha}\right)\ell_{i+1}\,,$$

using $s \geq 2 + 2/\alpha$. Since the total size of packets smaller than ℓ_i is less then $(1 + 1/\alpha)\ell_i$ by Lemma 3.2(ii), we obtain

$$L_{\mathsf{PG}(s)}(\geq i, (u,v]) - X > \frac{2\ell_{i+1}}{\alpha} - \left(1 + \frac{1}{\alpha}\right)\ell_i \geq 2\ell_i - \left(1 + \frac{1}{\alpha}\right)\ell_i > 0\,,$$

where the penultimate inequality uses $\ell_{i+1}/\alpha \geq \ell_i$. Thus (3.4) holds.

Otherwise the unfinished packet has size at least ℓ_{i+2} and, by Step (3) of the algorithm, also $L_{\mathsf{PG}(s)}((u,v]) > \ell_{i+2}$. We have $\ell_{i+2} \geq \alpha\ell_{i+1}$ and by the case condition $\ell_{i+1} > 2\ell_i$ we obtain

$$L_{\mathsf{PG}(s)}(\geq i, (u,v]) - X > (\alpha-1)\ell_{i+1} - \left(1 + \frac{1}{\alpha}\right)\ell_i > 2(\alpha-1)\ell_i - \left(1 + \frac{1}{\alpha}\right)\ell_i \geq 0\,,$$

as the definition of α_1 implies that $2(\alpha-1) \geq 1 + 1/\alpha$ for $\alpha \geq \alpha_1$. Thus (3.4) holds.

We now observe that we have exhausted all the possible values of X for $\alpha \geq \alpha_1$. Indeed, if at least two packets contribute to X, either (vi) or (vii) applies. Otherwise, at most a single packet contributes to X, and one of the cases (i)-(iv) or (viii) applies, as (iv) covers the case when $X \geq \ell_{i+2}$. Thus (3.4) holds and the proof is complete. □

Divisible Packet Sizes

Now, we turn briefly to even more restricted *divisible instances* considered by Jurdziński et al. [JKL15], which are a special case of 2-separated instances. Namely, we improve upon Theorem 3.9 in Theorem 3.10 presented below in the following sense: While the former guarantees that PG(s) is 1-competitive on (more general) 2-separated instances at speed $s \geq 3$, the latter shows that speed $s \geq 2.5$ is sufficient for (more restricted) divisible instances. Moreover, we note that that by an example in Section 3.5, the bound of Theorem 3.10 is tight, i.e., PG(s) is not 1-competitive for $s < 2.5$, even on divisible instances.

Theorem 3.10. *If the packet sizes are divisible, then PG(s) is 1-competitive for $s \geq 2.5$.*

Proof. Lemma 3.7(i) implies (3.2). We now prove (3.1) for any proper i-segment $(u, v]$ with $v - u \geq \ell_i$ and $R = 1$. The bound then follows by the Master Theorem. Since there is a fault at time u, we have $L_{\mathsf{OPT}}(\geq i, (u, v]) \leq v - u$.

By divisibility we have $L_{\mathsf{OPT}}(\geq i, (u, v]) = n\ell_i$ for some nonnegative integer n. We distinguish two cases based on the size of the last packet started by PG in the i-segment $(u, v]$, which is possibly unfinished due to a fault at v.

If the unfinished packet has size at most $n\ell_i$, then

$$L_{\mathsf{PG}(s)}(\geq i, (u, v]) > 5(v - u)/2 - \ell_i - \ell_{i-1} - n\ell_i \geq 5n\ell_i/2 - 3\ell_i/2 - n\ell_i \geq (n - 1)\ell_i$$

by Lemma 3.1 and Lemma 3.2(ii). Divisibility now implies $L_{\mathsf{PG}(s)}(\geq i, (u, v]) \geq n\ell_i = L_{\mathsf{OPT}}(\geq i, (u, v])$.

Otherwise, by divisibility the size of the unfinished packet is at least $(n + 1)\ell_i$ and the size of the completed packets is larger by the condition in Step (3) of the algorithm; here we also use the fact that PG(s) completes the packet started at u, as its size is at most $\ell_i \leq v - u$ (otherwise, u would be i-good, thus $C_i \geq u$ and $(u, v]$ is not a proper i-segment). Thus $L_{\mathsf{PG}(s)}(\geq i, (u, v]) > (n+1)\ell_i - 3\ell_i/2 \geq (n - 1/2)\ell_i$. Divisibility again implies $L_{\mathsf{PG}(s)}(\geq i, (u, v]) \geq n\ell_i = L_{\mathsf{OPT}}(\geq i, (u, v])$, which shows (3.1). □

3.6.3 Algorithm PG-DIV and its Analysis

We introduce our other algorithm PG-DIV designed for divisible instances. Actually, it is rather a fine-tuned version of PG, as it differs from it only in Step (3), where PG-DIV enforces an additional *divisibility condition*, set apart by italics in its formalization below. Then, using our framework of local analysis from this section, we give a simple proof that PG-DIV matches the performance of the algorithms from [JKL15] on divisible instances.

Algorithm PG-DIV:

```
(1) If no packet is pending, stay idle until the next release time.
(2) Let $i$ be the maximal $i \leq k$ such that there is a pending packet of size $\ell_i$ and
    $\ell(P^{<i}) < \ell_i$. Schedule a packet of size $\ell_i$ and set $t_B = t$.
(3) Choose the maximum $i$ such that
        (i) there is a pending packet of size $\ell_i$,
        (ii) $\ell_i \leq \mathrm{rel}(t)$ and
        (iii) $\ell_i$ divides rel($t$).
    Schedule a packet of size $\ell_i$. Repeat Step (3) as long as such $i$ exists.
(4) If no packet satisfies the condition in Step (3), go to Step (1).
```

Throughout the section we assume that the packet sizes are divisible. We note that Lemmas 3.1 and 3.4 and the Master Theorem apply to PG-DIV as well, since their proofs are not influenced by the divisibility condition. In particular, the definition of critical times C_i (Definition 3.3) remains the same. Thus, this section is devoted to leveraging divisibility to prove stronger stronger analogues of Lemma 3.2 and Lemma 3.7 (which are not needed to prove the Master Theorem) in this order. Once established, these are combined with the Master Theorem to prove that PG-DIV(2) is 1-competitive and PG-DIV(1) is 2-competitive. Recall that $\mathrm{rel}(t) = s \cdot (t - t_B)$ is the relative time after the start of the current phase t_B, scaled by the speed of the algorithm.

Lemma 3.11. (i) *If PG-DIV starts or completes a packet of size ℓ_i at time t, then ℓ_i divides rel(t).*

(ii) *Let t be a time with rel(t) divisible by ℓ_i and rel$(t) > 0$. If a packet of size ℓ_i is pending at time t, then PG-DIV starts or continues running a packet of size at least ℓ_i at time t.*

(iii) *If at the beginning of phase at time u a packet of size ℓ_i is pending and no fault occurs before time $t = u + \ell_i/s$, then the phase does not end before t.*

Proof. (i) follows trivially from the description of the algorithm.

(ii): If PG-DIV continues running some packet at t, it cannot be a packet smaller than ℓ_i by (i) and the claim follows. If PG-DIV starts a new packet, then a packet of size ℓ_i is pending by the assumption. Furthermore, it satisfies all the conditions from Step 3 of the algorithm, as rel(t) is divisible by ℓ_i and rel$(t) \geq \ell_i$ (from rel$(t) > 0$ and divisibility). Thus the algorithm starts a packet of size at least ℓ_i.

(iii): We proceed by induction on i. Assume that no fault happens before t. If the phase starts by a packet of size at least ℓ_i, the claim holds trivially, as the packet is not completed before t. This also proves the base of the induction for $i = 1$.

It remains to handle the case when the phase starts by a packet smaller than ℓ_i. Let $P^{<i}$ be the set of all packets of size smaller than ℓ_i pending at time u. By the Step (2) of the algorithm, $\ell(P^{<i}) \geq \ell_i$. We show that all packets of $P^{<i}$ are completed if no fault happens, which implies that the phase does not end before t.

Let j be such that ℓ_j is the maximum size of a packet in $P^{<i}$; note that j exists, as the phase starts by a packet smaller than ℓ_i. By the induction assumption, the phase does not end before time $t' = u + \ell_j/s$. From time t' on, the conditions in Step (3) guarantee that the remaining packets from $P^{<i}$ are processed from the largest ones, possibly interleaved with some of the newly arriving packets of larger sizes, as rel(τ) for the current time $\tau \geq t'$ such that a packet completes at τ is always divisible by the size of the largest pending packet from $P^{<i}$. This shows that the phase cannot end before all packets from $P^{<i}$ are completed if no fault happens. □

Now we prove a stronger analogue of Lemma 3.7.

Lemma 3.12. (i) *If $(u, v]$ is the initial i-segment, then*

$$L_{\textsf{PG-DIV}(s)}(\geq i, (u, v]) > s(v - u) - 3\ell_k\,.$$

(ii) *If $(u, v]$ is a proper i-segment and $v - u \geq \ell_i$ then*

$$L_{\textsf{PG-DIV}(s)}(\geq i, (u, v]) > s(v - u)/2 - \ell_i\,.$$

Furthermore, $L_{\textsf{PG-DIV}(s)}((u, v]) > s(v - u)/2$ and $L_{\textsf{PG-DIV}(s)}((u, v])$ is divisible by ℓ_i.

Proof. Suppose that time $t \in [u, v)$ satisfies that rel(t) is divisible by ℓ_i and rel$(t) > 0$. Then observe that Lemma 3.11(ii) together with the assumption that a packet of size ℓ_i is always pending in $[u, v)$ implies that from time t on only packets of size at least ℓ_i are scheduled, and thus the current phase does not end before v.

For a proper i-segment $(u, v]$, the previous observation for $t = u + \ell_i/s$ immediately implies (ii): Observe that $t \leq v$ by the assumption of (ii). Now $L_{\textsf{PG-DIV}(s)}(< i, (u, v])$ is either equal to 0 (if the phase starts by a packet of size ℓ_i at time u), or equal to ℓ_i (if the phase starts by a smaller packet). In both cases ℓ_i divides $L_{\textsf{PG-DIV}(s)}(< i, (u, v])$ and thus also $L_{\textsf{PG-DIV}(s)}((u, v])$. As in the analysis of PG, the total size of completed packets is more than $s(v - u)/2$ and (ii) follows.

For the initial i-segment $(u, v]$ we first observe that the claim is trivial if $s(v - u) \leq 2\ell_i$. So we may assume that $u + 2\ell_i/s \leq v$. Now we distinguish two cases:

1. The phase of u ends at some time $u' \leq u+\ell_i/s$: Then, by Lemma 3.11(iii) and the initial observation, the phase that immediately follows the one of u does not end in (u', v) and from time $u'+\ell_i/s$ on, only packets of size at least ℓ_i are scheduled. Thus $L_{\mathsf{PG\text{-}DIV}(s)}(< i, (u,v]) \leq 2\ell_i$.

2. The phase of u does not end by time $u + \ell_i/s$: Thus there exists $t \in (u, u + \ell_i/s]$ such that ℓ_i divides $\mathrm{rel}(t)$ and also $\mathrm{rel}(t) > 0$ as $t > u$. Using the initial observation for this t we obtain that the phase does not end in (u, v) and from time t on only packets of size at least ℓ_i are scheduled. Thus $L_{\mathsf{PG\text{-}DIV}(s)}(< i, (u,v]) \leq \ell_i$.

In both cases $L_{\mathsf{PG\text{-}DIV}(s)}(< i, (u,v]) \leq 2\ell_i$, furthermore only a single packet is possibly unfinished at time v. Thus $L_{\mathsf{PG\text{-}DIV}(s)}(\geq i, (u,v]) > s(v-u)-2\ell_i-\ell_k$ and (i) follows. □

Theorem 3.13. *Let the packet sizes be divisible. Then $\mathsf{PG\text{-}DIV}(1)$ is 2-competitive. Also, for any speed $s \geq 2$, $\mathsf{PG\text{-}DIV}(s)$ is 1-competitive.*

Proof. Lemma 3.12(i) implies (3.2). We now prove (3.1) for any proper i-segment $(u, v]$ with $v - u \geq \ell_i$ and appropriate R. The theorem then follows by the Master Theorem.

Since u is a time of a fault, we have $L_{\mathsf{OPT}}(\geq i, (u,v]) \leq v-u$. If $L_{\mathsf{OPT}}(\geq i, (u,v]) = 0$, (3.1) is trivial. Otherwise $L_{\mathsf{OPT}}(\geq i, (u,v]) \geq \ell_i$, thus $v - u \geq \ell_i$ and the assumption of Lemma 3.12(ii) holds.

For $s \geq 2$, Lemma 3.12(ii) implies

$$L_{\mathsf{PG\text{-}DIV}(s)}(\geq i, (u,v]) > s(v-u)/2 - \ell_i \geq v - u - \ell_i \geq L_{\mathsf{OPT}}(\geq i, (u,v]) - \ell_i\,.$$

Since both $L_{\mathsf{PG\text{-}DIV}(s)}(\geq i, (u,v])$ and $L_{\mathsf{OPT}}(\geq i, (u,v])$ are divisible by ℓ_i, this implies $L_{\mathsf{PG\text{-}DIV}(s)}(\geq i, (u,v]) \geq L_{\mathsf{OPT}}(\geq i, (u,v])$, i.e., (3.1) holds for $R = 1$.

For $s = 1$, Lemma 3.12(ii) implies

$$\begin{aligned} L_{\mathsf{PG\text{-}DIV}}((u,v]) + L_{\mathsf{PG\text{-}DIV}}(\geq i, (u,v]) &> (v-u)/2 + (v-u)/2 - \ell_i \\ &\geq v - u - \ell_i \geq L_{\mathsf{OPT}}(\geq i, (u,v]) - \ell_i\,. \end{aligned}$$

Since $L_{\mathsf{PG\text{-}DIV}}((u,v])$, $L_{\mathsf{PG\text{-}DIV}}(\geq i, (u,v])$, and $L_{\mathsf{OPT}}(\geq i, (u,v])$ are all divisible by ℓ_i, this implies $L_{\mathsf{PG\text{-}DIV}}((u,v]) + L_{\mathsf{PG\text{-}DIV}}(\geq i, (u,v]) \geq L_{\mathsf{OPT}}(\geq i, (u,v])$, i.e., (3.1) holds for $R = 2$. □

Example with Two Divisible Packet Sizes

We show that for our algorithms speed 2 is necessary if we want a ratio below 2, even if there are only two packet sizes in the instance. This matches the upper bound given in Theorem 3.8 for $\mathsf{PG}(2)$ and our upper bounds for $\mathsf{PG\text{-}DIV}(s)$ on divisible instances, i.e., ratio 2 for $s < 2$ and ratio 1 for $s \geq 2$. We remark that by Theorem 3.21, no deterministic algorithm can be 1-competitive with speed $s < 2$ on divisible instances, but this example shows a stronger lower bound for our algorithms, namely that their ratios are at least 2.

Remark. PG and $\mathsf{PG\text{-}DIV}$ have ratio no smaller than 2 when $s < 2$, even if packet sizes are only 1 and $\ell \geq \max\{s + \varepsilon,\ \varepsilon/(2 - s)\}$ for an arbitrarily small $\varepsilon > 0$.

Proof. We denote either algorithm by ALG. There will be N phases, that all look the same: In each phase, issue one packet of size ℓ and ℓ packets of size 1, and have the phase end by a fault at time $(2\ell - \varepsilon)/s \geq \ell$ which holds by the bounds on ℓ. Then ALG will complete all ℓ packets of size 1 but will not complete the one of size ℓ. By the previous inequality, OPT can complete the packet of size ℓ within the phase. Once all N phases are over, the jams occur every 1 unit of time, which allows OPT completing all $N\ell$ remaining packets of size 1. However, ALG is unable to complete any of the packets of size ℓ. Thus the ratio is 2. □

3.7 PrudentGreedy with Speed 4

In this section we prove that speed 4 is sufficient for PG to be 1-competitive. An example in Section 3.5 shows that speed 4 is also necessary for our algorithm.

Theorem 3.14. *PG(s) is 1-competitive for $s \geq 4$.*

Intuition For $s \geq 4$ we have that if at the start of a phase PG(s) has a packet of size ℓ_i pending and the phase has length at least ℓ_i, then PG(s) completes a packet of size at least ℓ_i. To show this, assume that the phase starts at time t. Then the first packet p of size at least ℓ_i is started before time $t + 2\ell_i/s$ by Lemma 3.2(ii) and by the condition in Step (3) it has size smaller than $2\ell_i$. Thus it completes before time $t + 4\ell_i/s \leq t + \ell_i$, which is before the end of the phase. This property does not hold for $s < 4$. It is important in our proof, as it shows that if the optimal schedule completes a job of some size, and such job is pending for PG(s), then PG(s) completes a job of the same size or larger. However, this is not sufficient to complete the proof by a local (phase-by-phase) analysis similar to the previous section, as the next example shows.

Assume that at the beginning, we release N packets of size 1, N packets of size $1.5 - 2\varepsilon$, one packet of size $3 - 2\varepsilon$ and a sufficient number of packets of size $1 - \varepsilon$, for a small $\varepsilon > 0$. Our focus is on packets of size at least 1. Supposing $s = 4$ we have the following phases:

- First, there are N phases of length 1. In each phase the optimum completes a packet of size 1, while among packets of size at least 1, PG(s) completes a packet of size $1.5 - 2\varepsilon$, as it starts packets of sizes $1 - \varepsilon$, $1 - \varepsilon$, $1.5 - 2\varepsilon$, $3 - 2\varepsilon$, in this order, and the last packet is jammed.
- Then there are N phases of length $1.5 - 2\varepsilon$ where the optimum completes a packet of size $1.5 - 2\varepsilon$ while among packets of size at least 1, the algorithm completes only a single packet of size 1, as it starts packets of sizes $1 - \varepsilon$, $1 - \varepsilon$, 1, $3 - 2\varepsilon$, in this order. The last packet is jammed, since for $s = 4$ the phase must have length at least $1.5 - \varepsilon$ to complete it.

In phases of the second type, the algorithm does not complete more (in terms of total size) packets of size at least 1 than the optimum. Nevertheless, in our example, packets of size $1.5 - 2\varepsilon$ were already finished by the algorithm, and this is a general rule. The novelty in our proof is a complex charging argument that exploits such subtle interaction between phases.

Outline of the proof We define critical times C'_i similarly as before, but without the condition that they should be ordered (thus either $C'_i \leq C'_{i-1}$ or $C'_i > C'_{i-1}$ may hold). Then, since the algorithm has nearly no pending packets of size ℓ_i just before C'_i, we can charge almost all adversary's packets of size ℓ_i started before C'_i to algorithm's packets of size ℓ_i completed before C'_i in a 1-to-1 fashion; we thus call these charges 1-to-1 charges. We account for the first few packets of each size completed at the beginning of ADV, the schedule of the adversary, in the additive constant of the competitive ratio, thereby shifting the targets of the 1-to-1 charges backward in time. This also resolves what to do with the yet uncharged packets pending for the algorithm just before C'_i.

After the critical time C'_i, packets of size ℓ_i are always pending for the algorithm, and thus (as we observed above) the algorithm schedules a packet of size at least ℓ_i

when the adversary completes a packet of size ℓ_i. It is actually more convenient not to work with phases, but partition the schedule into blocks inbetween successive faults. A block can contain several phases of the algorithm separated by an execution of Step (4); however, in the most important and tight part of the analysis the blocks coincide with phases.

In the crucial lemma of the proof, based on these observations and their refinements, we show that we can assign the remaining packets in ADV to algorithm's packets in the same block so that for each algorithm's packet q the total size of packets assigned to it is at most $\ell(q)$. However, we cannot use this assignment directly to charge the remaining packets, as some of the algorithm's big packets may receive 1-to-1 charges, and in this case the analysis needs to handle the interaction of different blocks. This very issue can be seen even in our introductory example.

To deal with this, we process blocks in the order of time from the beginning to the end of the schedule, simultaneously completing the charging to the packets in the current block of the schedule of PG(s) and possibly modifying ADV in the future blocks. In fact, in the assignment described above, we include not only the packets in ADV without 1-to-1 charges, but also packets in ADV with a 1-to-1 charge to a later block. After creating the assignment, if we have a packet q in PG that receives a 1-to-1 charge from a packet p in a later block of ADV, we remove p from ADV in that later block and replace it there by the packets assigned to q (that are guaranteed to be of smaller total size than p). After these swaps, the 1-to-1 charges together with the assignment form a valid charging that charges the remaining not swapped packets in ADV in this block together with the removed packets from the later blocks in ADV to the packets of PG(s) in the current block. This charging is now independent of the other blocks, so we can continue with the next block.

3.7.1 Blocks, Critical Times, 1-to-1 Charges and the Additive Constant

We now formally define the notions of blocks and (modified) critical times.

Definition 3.15. *Let $f_1, f_2, \ldots, f_N$ be the times of faults. Let $f_0 = 0$ and $f_{N+1} = T$ is the end of schedule. Then the time interval $(f_i, f_{i+1}]$, $i = 0, \ldots, N$, is called a block.*

Definition 3.16. *For $i = 1, \ldots k$, the critical time C'_i is the supremum of i-good times $t \in [0, T]$, where T is the end of the schedule and i-good times are as defined in Definition 3.3.*

All C'_i's are defined, as $t = 0$ is i-good for all i. Similarly to Section 3.6.1, each C'_i is of one of the following types: (i) C'_i starts a phase and a packet larger than ℓ_i is scheduled, (ii) $C'_i = 0$, (iii) $C'_i = T$, or (iv) just before time C'_i no packet of size ℓ_i is pending but at time C'_i one or more packets of size ℓ_i are pending; in this case C'_i is not i-good but only the supremum of i-good times. We observe that in each case, at time C'_i the total size of packets p of size ℓ_i pending for $\mathsf{PG}(s)$ and released before C'_i is less than ℓ_k.

Next we define the set of packets that contribute to the additive constant.

Definition 3.17. *Let the set A contain for each $i = 1, \ldots, k$:*
(i) *the first $\lceil 4\ell_k/\ell_i \rceil$ packets of size ℓ_i completed by the adversary, and*
(ii) *the first $\lceil 4\ell_k/\ell_i \rceil$ packets of size ℓ_i completed by the adversary after C'_i.*
If there are not sufficiently many packets of size ℓ_i completed by the adversary in (i) or (ii), we take all the packets in (i) or all the packets completed after C'_i in (ii), respectively.

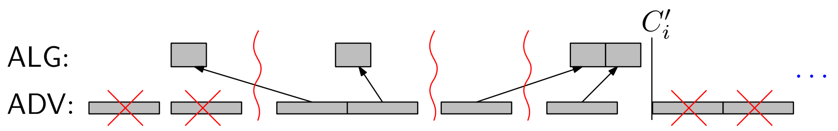


Figure 3.4: An illustration of back, up, and forward 1-to-1 charges for ℓ_i-sized packets (other packets are not shown). The winding lines depict the times of jamming errors, i.e., the beginnings and ends of blocks. Note that the packets in the algorithm's schedule are shorter, but wider, which illustrates that the algorithm runs the packets with a higher speed for a shorter time (the area thus corresponds to the amount of work done). Crossed packets are included in the set A (and thus contribute to the additive constant).

For each i, we put into A packets of size ℓ_i of total size at most $10\ell_k$. Thus we have $\ell(A) = \mathcal{O}(k\ell_k)$ which implies that packets in A can be counted in the additive constant.

We define 1-to-1 charges for packets of size ℓ_i as follows. Let p_1, p_2, . . . , p_n be all the packets of size ℓ_i started by the adversary before C_i' that are not in A. We claim that $\mathsf{PG}(s)$ completes at least n packets of size ℓ_i before C_i' if $n \geq 1$. Indeed, if $n \geq 1$, before time C_i' at least $n + \lceil 4\ell_k/\ell_i \rceil$ packets of size ℓ_i are started by the adversary and thus released; by the definition of C_i' at time C_i' fewer than ℓ_k/ℓ_i of them are pending for $\mathsf{PG}(s)$, one may be running and the remaining ones must be completed. We now charge each p_m to the mth packet of size ℓ_i completed by $\mathsf{PG}(s)$. Note that each packet started by the adversary is charged at most once and each packet completed by $\mathsf{PG}(s)$ receives at most one charge.

We call a 1-to-1 charge starting and ending in the same block an *up charge*, a 1-to-1 charge from a block starting at u to a block ending at $v' \leq u$ a *back charge*, and a 1-to-1 charge from a block ending at v to a block starting at $u' \geq v$ a *forward charge*; see Figure 3.4 for an illustration. A *charged packet* is a packet charged by a 1-to-1 charge. The definition of A implies the following two important properties.

Lemma 3.18. *Let p be a packet of size ℓ_i, started by the adversary at time t, charged by a forward charge to a packet q started by $\mathsf{PG}(s)$ at time t'. Then at any time $\tau \in [t - 3\ell_k, t')$, more than ℓ_k/ℓ_i packets of size ℓ_i are pending for $\mathsf{PG}(s)$.*

Proof. Let m be the number of packets of size ℓ_i that $\mathsf{PG}(s)$ completes before q. Then, by the definition of A, the adversary completes $m + \lceil 4\ell_k/\ell_i \rceil$ packets of size ℓ_i before p. As fewer than $3\ell_k/\ell_i$ of these packets are started in $(t - 3\ell_k, t]$, the remaining more than $m + \ell_k/\ell_i$ packets have been released before or at time $t - 3\ell_k$. As only m of them are completed by $\mathsf{PG}(s)$ before t', the remaining more than ℓ_k/ℓ_i packets are pending at any time $\tau \in [t - 3\ell_k, t')$. ☐

Lemma 3.19. *Let $p \notin A$ be a packet of size ℓ_i started by the adversary at time t that is not charged. Then $t - 4\ell_k \geq C_i'$ and thus at any $\tau \geq t - 4\ell_k$, a packet of size ℓ_i is pending for $\mathsf{PG}(s)$.*

Proof. Any packet of size ℓ_i started before $C_i' + 4\ell_k$ is either charged or put in A, thus $t - 4\ell_k \geq C_i'$. After C_i', a packet of size ℓ_i is pending by the definition of C_i'. ☐

3.7.2 Processing Blocks

Initially, let ADV be an optimal (adversary) schedule. First, we remove all packets in A from ADV. Then we process blocks one by one in the order of time. When we process a block, we modify ADV so that we (i) remove some packets from ADV, so that the total size of removed packets is at most the total size of packets completed by PG(s) in this block, and (ii) reschedule any remaining packet in ADV in this block to one of the later blocks, so that the schedule of remaining packets is still feasible. Summing over all blocks, (i) guarantees that PG(s) is 1-competitive with an additive constant $\ell(A)$.

When we reschedule a packet in ADV, we keep the packet's 1-to-1 charge (if it has one), however, its type may change due to rescheduling. Since we are moving packets to later times only, the release times are automatically respected. Also it follows that we can apply Lemmas 3.18 and 3.19 even to ADV after rescheduling.

After processing of a block, there will remain no charges to or from it. For the charges from the block, this is automatic, as ADV contains no packet in the block after we process it. For the charges to the block, this is guaranteed as in the process we remove from ADV all the packets in later blocks charged by back charges to the current block.

From now on, let $(u, v]$ be the current block that we are processing; all previous blocks ending at $v' \leq u$ are processed. As there are no charges to the previous blocks, any packet scheduled in ADV in $(u, v]$ is charged by an up charge or a forward charge, or else it is not charged at all. We distinguish two main cases of the proof, depending on whether PG(s) finishes any packet in the current block.

Main Case 1: Empty Block

The algorithm does not finish any packet in $(u, v]$. We claim that ADV does not finish any packet. The processing of the block is then trivial.

For a contradiction, assume that ADV starts a packet p of size ℓ_i at time t and completes it. The packet p cannot be charged by an up charge, as PG(s) completes no packet in this block. Thus p is either charged by a forward charge or not charged. Lemma 3.18 or 3.19 implies that at time t some packet of size ℓ_i is pending for PG(s).

Since PG does not idle unnecessarily, this means that some packet q of size ℓ_j for some j is started in PG(s) at time $\tau \leq t$ and running at t. As PG(s) does not complete any packet in $(u, v]$, the packet q is jammed by the fault at time v. This implies that $j > i$, as $\ell_j > s(v - \tau) \geq v - t \geq \ell_i$; we also have $t - \tau < \ell_j$. Moreover, q is the only packet started by PG(s) in this block, thus it starts a phase.

As this phase is started by packet q of size $\ell_j > \ell_i$, the time τ is i-good and $C'_i \geq \tau$. All packets ADV started before time $C'_i + 4\ell_k/s$ are charged, as the packets in A are removed from ADV and packets in ADV are rescheduled only to later times. Packet p is started before $v < \tau + \ell_j/s < C'_i + \ell_k/s$, thus it is charged. It follows that p is charged by a forward charge. We now apply Lemma 3.18 again and observe that it implies that at $\tau > t - \ell_j$ there are more than ℓ_k/ℓ_i packets of size ℓ_i pending for PG(s). This is in contradiction with the fact that at τ, PG(s) started a phase by q of size $\ell_j > \ell_i$.

Main Case 2: Non-empty Block

Otherwise, PG(s) completes a packet in the current block $(u, v]$.

Let Q be the set of packets completed by PG(s) in $(u, v]$ that do not receive an up charge. Note that no packet in Q receives a forward charge, as the modified ADV contains no packets before u, so packets in Q either get a back charge or no charge at all. Let P be the set of packets completed in ADV in $(u, v]$ that are not charged by an up charge. Note that P includes packets charged by a forward charge and uncharged packets, as no packets are charged to a previous block.

We first assign packets in P to packets in Q so that for each packet $q \in Q$ the total size of packets assigned to q is at most $\ell(q)$. Formally, we iteratively define a provisional assignment $f : P \to Q$ such that $\ell(f^{-1}(q)) \leq \ell(q)$ for each $q \in Q$.

Provisional assignment We maintain a set $O \subseteq Q$ of *occupied* packets that we do not use for a future assignment. Whenever we assign a packet p to $q \in Q$ and $\ell(q) - \ell(f^{-1}(q)) < \ell(p)$, we add q to O. This rule guarantees that each packet $q \in O$ has $\ell(f^{-1}(q)) > \ell(q)/2$.

We process packets in P in the order of decreasing sizes as follows. We take the largest unassigned packet $p \in P$ of size $\ell(p)$ (if there are more unassigned packets of size $\ell(p)$, we take an arbitrary one) and choose an arbitrary packet $q \in Q \setminus O$ such that $\ell(q) \geq \ell(p)$; we prove in Lemma 3.20 below that such a q exists. We assign p to q, that is, we set $f(p) = q$. Furthermore, as described above, if $\ell(q) - \ell(f^{-1}(q)) < \ell(p)$, we add q to O. We continue until all packets are assigned.

If a packet p is assigned to q and q is not put in O, it follows that $\ell(q) - \ell(f^{-1}(q)) \geq \ell(p)$. This implies that after the next packet p' is assigned to q, we have $\ell(q) \geq \ell(f^{-1}(q))$, as the packets are processed from the largest one and thus $\ell(p') \leq \ell(p)$. If follows that at the end we obtain a valid provisional assignment.

Lemma 3.20. *The assignment process above assigns all packets in P.*

Proof. For each size ℓ_j we show that all packets of size ℓ_j in P are assigned, which is clearly sufficient. We fix the size ℓ_j and define a few quantities.

Let n denote the number of packets of size ℓ_j in P. Let o denote the total *occupied size*, defined as $o = \ell(O) + \sum_{q \in Q \setminus O} \ell(f^{-1}(q))$ at the time just before we start assigning the packets of size ℓ_j. Note that the rule for adding packets to O implies that $\ell(f^{-1}(Q)) \geq o/2$. Let a denote the current total *available size* defined as $a = \sum_{q \in Q \setminus O : \ell(q) \geq \ell_j} (\ell(q) - \ell(f^{-1}(q)))$. We remark that in the definition of a we restrict attention only to packets of size $\geq \ell_j$, but in the definition of o we consider all packets in Q; however, as we process in the order of decreasing sizes, so far we have assigned packets from P only to packets of size $\geq \ell_j$ in Q.

First, we claim that it is sufficient to show that $a > (2n-2)\ell_j$ before we start assigning the packets of size ℓ_j. As long as $a > 0$, there is a packet $q \in Q \setminus O$ of size at least ℓ_j and thus we may assign the next packet (and, as noted before, actually $a \geq \ell_j$, as otherwise $q \in O$). Furthermore, assigning a packet p of size ℓ_j to q decreases a by ℓ_j if q is not added to O and by less than $2\ell_j$ if q is added to O. Altogether, after assigning the first $n-1$ packets, a decreases by less than $(2n-2)\ell_j$, thus we still have $a > 0$, and we can assign the last packet. The claim follows.

We now split the analysis into two cases, depending on whether there is a packet of size ℓ_j pending for $\mathsf{PG}(s)$ at all times in $[u, v)$, or not. In either case, we prove that the available space a is sufficiently large before assigning the packets of size ℓ_j.

In the first case, we suppose that a packet of size ℓ_j is pending for $\mathsf{PG}(s)$ at all times in $[u, v)$. Let z be the total size of packets of size at least ℓ_j charged by up charges in this block. The size of packets in P already assigned is at least $\ell(f^{-1}(Q)) \geq o/2$ and we have n yet unassigned packets of size ℓ_j in P. As ADV has to schedule all these packets and the packets with up charges in this block, its size satisfies $v-u \geq \ell(P)+z \geq n\ell_j+o/2+z$. Now consider the schedule of $\mathsf{PG}(s)$ in this block. By Lemma 3.2, there is no end of phase in (u, v) and jobs smaller than ℓ_j scheduled by $\mathsf{PG}(s)$ have total size less than $2\ell_j$. All the other completed packets contribute to one of a, o, or z. Using Lemma 3.1, the previous bound on $v - u$ and $s \geq 4$, the total size of completed packets is at least $s(v-u)/2 \geq 2n\ell_j + o + 2z$. Hence $a > (2n\ell_j + o + 2z) - 2\ell_j - o - z \geq (2n-2)\ell_j$, which completes the proof of the lemma in this case.

Otherwise, in the second case, there is a time in $[u, v)$ when no packet of size ℓ_j is pending for $\mathsf{PG}(s)$. Let τ be the supremum of times $\tau' \in [u, v]$ such that $\mathsf{PG}(s)$ has no

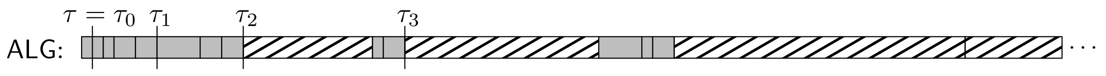


Figure 3.5: An illustration of bounding the total size of small packets completed after τ in the case when ℓ_j is not pending in the whole block. Gray packets are small, while hatched packets have size at least ℓ_j. The times τ_1, τ_2, and τ_3 are the ends of phases after τ (thus $\alpha = 3$), but τ need not be the end of a phase.

pending packet of size at least ℓ_j at time τ'; if no such τ' exists we set $\tau = u$. Let t be the time when the adversary starts the first packet p of size ℓ_j from P.

Since p is charged using a forward charge or p is not charged, we can apply Lemma 3.18 or 3.19, which implies that packets of size ℓ_j are pending for $\mathsf{PG}(s)$ from time $t - 3\ell_k$ till at least v. By the case condition, there is a time in $[u, v)$ when no packet of size ℓ_j is pending, and this time is thus before $t - 3\ell_k$, implying $u < t - 3\ell_k$. The definition of τ now implies that $\tau \le t - 3\ell_k$.

Towards bounding a, we show that (i) $\mathsf{PG}(s)$ runs a limited amount of small packets after τ and thus $a + o$ is large, and that (ii) $f^{-1}(Q)$ contains only packets run by ADV from τ on, and thus o is small.

We claim that the total size of packets smaller than ℓ_j completed in $\mathsf{PG}(s)$ in $(\tau, v]$ is less than $3\ell_k$. This claim is similar to Lemma 3.2 and we also argue similarly. Let $\tau_1 < \tau_2 < \ldots < \tau_\alpha$ be all the ends of phases in (τ, v) (possibly there is none, then $\alpha = 0$); also let $\tau_0 = \tau$. For $i = 1, \ldots, \alpha$, let r_i denote the packet started by $\mathsf{PG}(s)$ at τ_i; note that r_i exists since after τ there is a pending packet at any time in $[\tau, v]$ by the definition of τ. See Figure 3.5 for an illustration. First note that any packet started at or after time $\tau_\alpha + \ell_k/s$ has size at least ℓ_j, as such a packet is pending and satisfies the condition in Step (3) of the algorithm. Thus the total amount of the small packets completed in $(\tau_\alpha, v]$ is less than $\ell_k + \ell_{k-1} < 2\ell_k$. The claim now follows for $\alpha = 0$. Otherwise, as there is no fault in (u, v), at τ_i, $i = 1, \ldots, \alpha$, Step (4) of the algorithm is reached and thus no packet of size at most $s(\tau_i - \tau_{i-1})$ is pending. In particular, this implies that $\ell(r_i) > s(\tau_i - \tau_{i-1})$ for $i = 1, \ldots, \alpha$. This also implies that the amount of the small packets completed in $(\tau_0, \tau_1]$ is less than ℓ_k and the claim for $\alpha = 1$ follows. For $\alpha \ge 2$ first note that by Lemma 3.2(i), $s(\tau_i - \tau_{i-1}) \ge \ell_j$ for all $i = 2, \ldots, \alpha$ and thus r_i is not a small packet. Thus for $i = 3, \ldots, \alpha$, the amount of small packets in $(\tau_{i-1}, \tau_i]$ is at most $s(\tau_i - \tau_{i-1}) - \ell(r_{i-1}) < \ell(r_i) - \ell(r_{i-1})$. The amount of small packets completed in $(\tau_1, \tau_2]$ is at most $s(\tau_2 - \tau_1) < \ell(r_2)$ and the amount of small packets completed in $(\tau_\alpha, v]$ is at most $2\ell_k - \ell(r_\alpha)$. Summing this together, the amount of small packets completed in $(\tau_1, v]$ is at most $2\ell_k$ and the claim follows.

Let z be the total size of packets of size at least ℓ_j charged by up charges in this block and completed by $\mathsf{PG}(s)$ after τ. After τ, $\mathsf{PG}(s)$ processes packets of total size more than $s(v - \tau) - \ell_k$ and all of these packets contribute to one of a, o, z, or the volume of less than $3\ell_k$ of small packets from the claim above. Thus, using $s \ge 4$, we get

$$a > 4(v - \tau) - o - z - 4\ell_k \,. \tag{3.5}$$

Now we derive two lower bounds on $v - \tau$ using ADV schedule.

Observe that no packet contributing to z except for possibly one (the one possibly started by $\mathsf{PG}(s)$ before τ) is started by ADV before τ, as otherwise, it would be pending for $\mathsf{PG}(s)$ just before τ, contradicting the definition of τ.

Also, observe that in $(u, \tau]$, ADV runs no packet $p \in P$ with $\ell(p) > \ell_j$: For a contradiction, assume that such a p exists. As $\tau \le C_{j'}$ for any $j' \ge j$, such a p is charged. As $p \in P$, it is charged by a forward charge. However, then Lemma 3.18

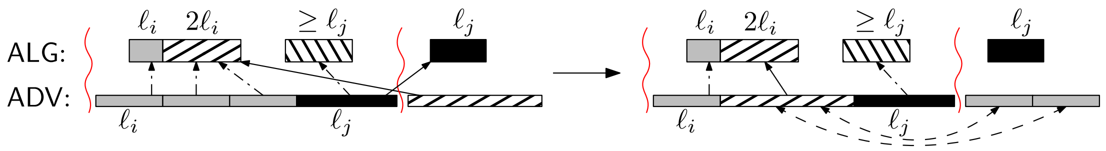


Figure 3.6: An illustration of the provisional assignment on the left; note that a packet of size ℓ_j with a forward charge is also assigned. Full arcs depict 1-to-1 charges and dashed arcs depict the provisional assignment. The result of modifying the adversary schedule on the right.

implies that at all times between the start of p in ADV and v a packet of size $\ell(p)$ is pending for $\mathsf{PG}(s)$; in particular, such a packet is pending in the interval before τ, contradicting the definition of τ.

These two observations imply that in $[\tau, v]$, ADV starts and completes all the assigned packets from P, the n packets of size ℓ_j from P, and all packets except possibly one contributing to z. This gives $v - \tau \geq \ell(f^{-1}(Q)) + n\ell_j + z - \ell_k \geq o/2 + n\ell_j + z - \ell_k$.

To obtain the second bound, we observe that the n packets of size ℓ_j from P are scheduled in $[t, v]$ and together with $t \geq \tau + 3\ell_k$ we obtain $v - \tau = v - t + t - \tau \geq n\ell_j + 3\ell_k$.

Summing the two bounds on $v - \tau$ and multiplying by two we get $4(v - \tau) \geq 4n\ell_j + 4\ell_k + o + 2z$. Summing with (3.5) we get $a > 4n\ell_j + z \geq 4n\ell_j$. This completes the proof of the second case. $\square$

As a remark, note that in the previous proof, the first case deals with blocks after C_j, it is the typical and tight case. The second case deals mainly with the block containing C_j, and also with some blocks before C_j, which brings some technical difficulties, but there is a lot of slack. This is similar to the situation in the local analysis using the Master Theorem.

Modifying the adversary schedule Now all the packets from P are provisionally assigned by f and for each $q \in Q$ we have that $\ell(f^{-1}(q)) \leq \ell(q)$.

We process each packet q completed by $\mathsf{PG}(s)$ in $(u, v]$ according to one of the following three cases; in each case we remove from ADV one or more packets with total size at most $\ell(q)$.

If $q \notin Q$, then the definition of P and Q implies that q is charged by an up charge from some packet $p \notin P$ of the same size. We remove p from ADV.

If $q \in Q$ does not receive a charge, we remove $f^{-1}(q)$ from ADV. Recall that $\ell(f^{-1}(q)) \leq \ell(q)$, so the size is as required. If any packet $p \in f^{-1}(q)$ is charged (necessarily by a forward charge), we remove this charge.

If $q \in Q$ receives a charge, it is a back charge from some packet p of the same size. We remove p from ADV and in the interval where p was scheduled, we schedule packets from $f^{-1}(q)$ in an arbitrary order. As $\ell(f^{-1}(q)) \leq \ell(q)$, this is feasible. If any packet $p \in f^{-1}(q)$ is charged, we keep its charge to the same packet in $\mathsf{PG}(s)$; the charge was necessarily a forward charge, so it leads to some later block. See Figure 3.6 for an illustration.

After we have processed all the packets q, we have modified ADV by removing an allowed total size of packets and rescheduling the remaining packets in $(u, v]$ so that any remaining charges go to later blocks. This completes processing of the block $(u, v]$ and thus also the proof of 1-competitiveness.

3.8 Lower Bounds

3.8.1 Lower Bound with Two Packet Sizes

In this section we study lower bounds on the speed necessary to achieve 1-competitiveness. We start with a lower bound of 2 which holds even for the divisible case. It follows that our algorithm PG-DIV and the algorithm in Jurdziński *et al.* [JKL15] are optimal. Note that this lower bound follows from results of Anta *et al.* [AGKZ15] by a similar construction, although the packets in their construction are not released together.

Theorem 3.21. *There is no 1-competitive deterministic online algorithm running with speed $s < 2$, even if packets have sizes only 1 and ℓ for $\ell > 2s/(2-s)$ and all of them are released at time 0.*

Proof. For a contradiction, consider an algorithm ALG running with speed $s < 2$ that is claimed to be 1-competitive with an additive constant A where A may depend on ℓ. At time 0 the adversary releases $N_1 = \lceil A/\ell \rceil + 1$ packets of size ℓ and $N_0 = \left\lceil \frac{2\ell}{s} \cdot (N_1 \cdot (s-1) \cdot \ell + A + 1) \right\rceil$ packets of size 1. These are all packets in the instance.

The adversary's strategy works by blocks where a block is a time interval between two faults and the first block begins at time 0. The adversary ensures that in each such block ALG completes no packet of size ℓ and moreover ADV either completes an ℓ-sized packet, or completes more 1's (packets of size 1) than ALG.

Let t be the time of the last fault; initially $t = 0$. Let $\tau \geq t$ be the time when ALG starts the first ℓ-sized packet after t (or at t) if now fault occurs after t; we set $\tau = \infty$ if it does not happen. Note that we use here that ALG is deterministic. In a block beginning at time t, the adversary proceeds according to the first case below that applies.

(D1) If ADV has less than $2\ell/s$ pending packets of size 1, then the end of the schedule is at t.

(D2) If ADV has all packets of size ℓ completed, then it stops the current process and issues faults at times $t+1, t+2, \ldots$ Between every two consecutive faults after t it completes one packet of size 1 and it continues issuing faults until it has no pending packet of size 1. Then there is the end of the schedule. Clearly, ALG may complete only packets of size 1 after t as $\ell > 2s/(2-s) > s$ for $s < 2$.

(D3) If $\tau \geq t + \ell/s - 2$, then the next fault is at time $t + \ell$. In the current block, the adversary completes a packet ℓ. ALG completes at most $s \cdot \ell$ packets of size 1 and then it possibly starts ℓ at τ (if $\tau < t + \ell$) which is jammed, since it would be completed at

$$\tau + \frac{\ell}{s} \geq t + \frac{2\ell}{s} - 2 = t + \ell + \left(\frac{2}{s} - 1\right)\ell - 2 > t + \ell$$

where the last inequality follows from $\left(\frac{2}{s} - 1\right)\ell > 2$ which is equivalent to $\ell > 2s/(2-s)$. Thus the ℓ-sized packet would be completed after the fault. See Figure 3.7 for an illustration.

(D4) Otherwise, if $\tau < t + \ell/s - 2$, then the next fault is at time $\tau + \ell/s - \varepsilon$ for a small enough $\varepsilon > 0$. In the current block, ADV completes as many packets of size 1 as it can, that is $\lfloor \tau + \ell/s - \varepsilon - t \rfloor$ packets of size 1; note that by Case (D1), ADV has enough 1's pending. Again, the algorithm does not complete the packet of size ℓ started at τ, because it would be finished at $\tau + \ell/s$. See Figure 3.8 for an illustration.

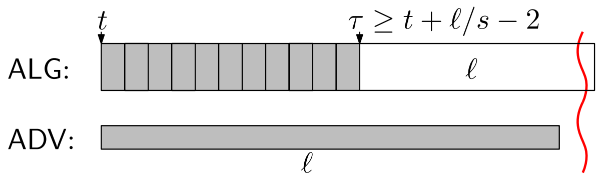


Figure 3.7: An illustration of Case (D3).

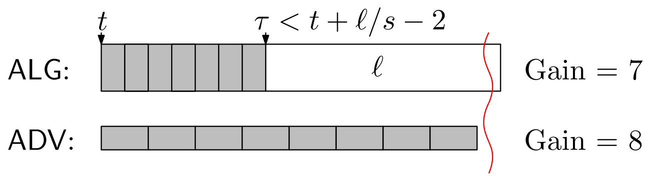


Figure 3.8: An illustration of Case (D4).

First notice that the process above ends, since in each block the adversary completes a packet. We now show $L_{\mathsf{ADV}} > L_{\mathsf{ALG}} + A$ which contradicts the claimed 1-competitiveness of ALG.

If the adversary's strategy ends in Case (D2), then ADV has all ℓ's completed and then it schedules all 1's, thus $L_{\mathsf{ADV}} = N_1 \cdot \ell + N_0 > A + N_0$. However, ALG does not complete any ℓ-sized packet and hence $L_{\mathsf{ALG}} \leq N_0$ which concludes this case.

Otherwise, the adversary's strategy ends in Case (D1). We first claim that in a block $(t, t']$ created in Case (D4), ADV finishes more 1's than ALG. Indeed, let o be the number of 1's completed by ALG in $(t, t']$. Then $\tau \geq t + o/s$ where τ is from the adversary's strategy in $(t, t']$, and we also have $o < \ell - 2s$ or equivalently $\ell > o + 2s$, because $\tau < t + \ell/s - 2$ in Case (D4). The number of 1's scheduled by ADV is

$$\left\lfloor \tau + \frac{\ell}{s} - \varepsilon - t \right\rfloor \geq \left\lfloor t + \frac{o}{s} + \frac{\ell}{s} - \varepsilon - t \right\rfloor \geq \left\lfloor \frac{o}{s} + \frac{o + 2s}{s} - \varepsilon \right\rfloor = \left\lfloor \frac{2}{s} o + 2 - \varepsilon \right\rfloor$$
$$\geq \left\lfloor \frac{2}{s} o + 1 \right\rfloor \geq o + 1$$

and we proved the claim.

Let α be the number of blocks created in Case (D3); note that $\alpha \leq N_1$, since in each such block ADV finishes one ℓ-sized packet. ALG completes at most $s\ell$ packets of size 1 in such a block, thus $L_{\mathsf{ADV}}((u, v]) - L_{\mathsf{ALG}}((u, v]) \geq (1 - s) \cdot \ell$ for a block $(u, v]$ created in Case (D3).

Let β be the number of blocks created in Case (D4). We have

$$\beta > \frac{s}{2\ell} \cdot \left(N_0 - \frac{2\ell}{s} \right) = \frac{s \cdot N_0}{2\ell} - 1 = N_1 \cdot (s - 1) \cdot \ell + A\,,$$

because in each such block ADV schedules less than $2\ell/s$ packets of size 1 and less than $2\ell/s$ of these packets are pending at the end. By the claim above, we have $L_{\mathsf{ADV}}((u, v]) - L_{\mathsf{ALG}}((u, v]) \geq 1$ for a block $(u, v]$ created in Case (D4).

Summing over all blocks and using the value of N_0 we get

$$L_{\mathsf{ADV}} - L_{\mathsf{ALG}} \geq \alpha \cdot (1 - s) \cdot \ell + \beta > N_1 \cdot (1 - s) \cdot \ell + N_1 \cdot (s - 1) \cdot \ell + A = A$$

where we used $s \geq 1$ which we may suppose w.l.o.g. This concludes the proof. □

3.8.2 Lower Bound for General Packet Sizes

Our main lower bound of $\phi + 1 = \phi^2 \approx 2.618$ (where $\phi = (\sqrt{5}+1)/2$ is the golden ratio) generalizes the construction of Theorem 3.21 for more packet sizes, which are no longer divisible. Still, we make no use of release times.

Theorem 3.22. *There is no 1-competitive deterministic online algorithm running with speed $s < \phi + 1$, even if all packets are released at time 0.*

Outline of the proof We start by describing the adversary's strategy which works against an algorithm running at speed $s < \phi + 1$, i.e., it shows that it is not 1-competitive. It can be seen as a generalization of the strategy with two packet sizes above, but at the end the adversary sometimes needs a new strategy how to complete all short packets (of size less than ℓ_i for some i), preventing the algorithm to complete a long packet (of size at least ℓ_i).

Then we show a few lemmas about the behavior of the algorithm. Finally, we prove that the gain of the adversary, i.e., the total size of its completed packets, is substantially larger than the gain of the algorithm.

Adversary's strategy The adversary chooses $\varepsilon > 0$ small enough and $k \in \mathbb{N}$ large enough so that $s < \phi + 1 - 1/\phi^{k-1}$. For convenience, the smallest size in the instance is ε instead of 1. There will be $k+1$ packet sizes in the instance, namely $\ell_0 = \varepsilon$, and $\ell_i = \phi^{i-1}$ for $i = 1, \dots, k$.

Suppose for a contradiction that there is an algorithm ALG running with speed $s < \phi + 1$ that is claimed to be 1-competitive with an additive constant A where A may depend on ℓ_i's, in particular on ε and k. The adversary issues N_i packets of size ℓ_i at time 0, for $i = 0, \dots, k$; N_i's are chosen so that $N_0 \gg N_1 \gg \cdots \gg N_k$. These are all packets in the instance.

More precisely, N_i's are defined inductively so that it holds that $N_k > A/\ell_k$, $N_i > \phi s \ell_k \sum_{j=i+1}^{k} N_j + A/\ell_i$ for $i = k-1, \dots, 1$, and finally

$$N_0 > \frac{A + 1 + \phi \ell_k}{\varepsilon^2} \cdot \left(\phi s \ell_k \sum_{i=1}^{k} N_i\right).$$

The adversary's strategy works by blocks where a block is again a time interval between two faults and the first block begins at time 0. Let t be the time of the last fault; initially $t = 0$. Let $\tau_i \geq t$ be the time when ALG starts the first packet of size ℓ_i after t (or at t) if no fault occurs after t; we set $\tau_i = \infty$ if it does not happen. Again, we use here that ALG is deterministic. Let $\tau_{\geq i} = \min_{j \geq i} \tau_j$ be the time when ALG starts the first packet of size at least ℓ_i after t. Let $P_{\mathsf{ADV}}(i)$ be the total size of ℓ_i's (packets of size ℓ_i) pending for the adversary at time t.

In a block beginning at time t, the adversary proceeds according to the first case below that applies. Each case has an intuitive explanation which we make precise later.

(B1) If there are less than $\phi \ell_k / \varepsilon$ packets of size ε pending for ADV, then the end of the schedule is at time t.

Lemma 3.23 below shows that in blocks in which ADV schedules ε's it completes more than ALG in terms of total size. It follows that the schedule of ADV has much larger total completed size for N_0 large enough, since the adversary scheduled nearly all packets of size ε; see Lemma 3.28.

(B2) If there is $i \geq 1$ such that $P_{\mathsf{ADV}}(i) = 0$, then ADV stops the current process and continues by Strategy Finish described below.

(B3) If $\tau_1 < t + \ell_1/(\phi \cdot s)$, then the next fault occurs at time $\tau_1 + \ell_1/s - \varepsilon$, so that ALG does not finish the first ℓ_1-sized packet. ADV schedules as many ε's as it can.

In this case, ALG schedules ℓ_1 too early and in Lemma 3.23 we show that the total size of packets completed by ADV is larger than the total size of packets completed by ALG.

(B4) If $\tau_{\geq 2} < t + \ell_2/(\phi \cdot s)$, then the next fault is at time $\tau_{\geq 2} + \ell_2/s - \varepsilon$, so that ALG does not finish the first packet of size at least ℓ_2. ADV again schedules as many ε's as it can. Similarly as in the previous case, ALG starts ℓ_2 or a larger packet too early and we show that ADV completes more in terms of size than ALG, again using Lemma 3.23.

(B5) If there is $1 \leq i < k$ such that $\tau_{\geq i+1} < \tau_i$, then we choose the smallest such i and the next fault is at time $t + \ell_i$. ADV schedules a packet of size ℓ_i. See Figure 3.9 for an illustration.

Intuitively, this case means that ALG skips ℓ_i and schedules ℓ_{i+1} (or a larger packet) earlier. Lemma 3.25 shows that the algorithm cannot finish its first packet of size at least ℓ_{i+1} (thus it also does not schedule ℓ_i) provided that this case is not triggered for a smaller i, or previous cases are not triggered.

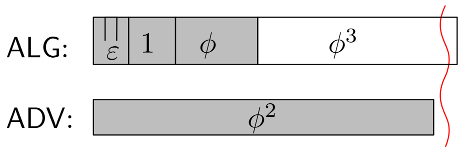


Figure 3.9: An illustration of Case (B5).

(B6) Otherwise, the next fault occurs at $t + \ell_k$ and ADV schedules a packet of size ℓ_k in this block. Lemma 3.26 shows that ALG cannot complete an ℓ_k-sized packet in this block. See Figure 3.10 for an illustration.

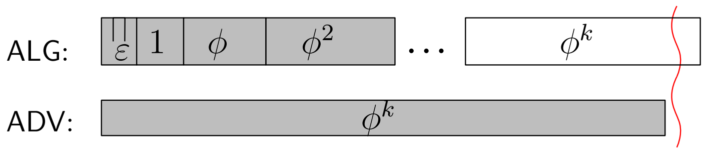


Figure 3.10: An illustration of Case (B6).

We remark that the process above eventually ends either in Case (B1), or in Case (B2), since in each block ADV schedules a packet. Also note that the length of each block is at most $\phi\ell_k$.

We describe Strategy FINISH, started in Case (B2). Let i be the smallest index $i' \geq 1$ such that $P_{\mathsf{ADV}}(i') = 0$. For brevity, we call a packet of size at least ℓ_i *long*, and a packet of size ℓ_j with $1 \leq j < i$ *short*; note that ε's are not short packets. In a nutshell, ADV tries to schedule all short packets, while preventing the algorithm from completing any long packet. Similarly to Cases (B3) and (B4), if ALG is starting a long packet too early, ADV schedules ε's and gains in terms of total size.

Adversary's Strategy FINISH works again by blocks. Let t be the time of the last fault. Let $\tau \geq t$ be the time when ALG starts the first long packet after t; we set $\tau = \infty$

if it does not happen. The adversary proceeds according to the first case below that applies:

(F1) If $P_{\mathsf{ADV}}(0) < \phi\ell_k$, then the end of the schedule is at time t.

(F2) If ADV has no pending short packet, then the strategy Finish ends and ADV issues faults at times $t+\varepsilon, t+2\varepsilon, \ldots$ Between every two consecutive faults after t it completes one packet of size ε and it continues issuing faults until it has no pending ε. Then there is the end of the schedule. Clearly, ALG may complete only ε's after t if ε is small enough. Note that for $i = 1$ this case is immediately triggered, as ℓ_0-sized packets are not short, hence there are no short packets whatsoever.

(F3) If $\tau < t + \ell_i/(\phi \cdot s)$, then the next fault is at time $\tau + \ell_i/s - \varepsilon$, so that ALG does not finish the first long packet. ADV schedules as many ε's as it can. Note that the length of this block is less than $\ell_i/(\phi \cdot s) + \ell_i/s \leq \phi\ell_k$. Again, we show that ADV completes more in terms of size using Lemma 3.23.

(F4) Otherwise, $\tau \geq t + \ell_i/(\phi \cdot s)$. ADV issues the next fault at time $t + \ell_{i-1}$. Let j be the largest $j' < i$ such that $P_{\mathsf{ADV}}(j') > 0$. ADV schedules a packet of size ℓ_j which is completed as $j \leq i - 1$. Lemma 3.27 shows that ALG does not complete the long packet started at τ.

Again, in each block ADV completes a packet, thus Strategy Finish eventually ends. Note that the length of each block is less than $\phi\ell_k$.

Properties of the adversary's strategy We now prove the lemmas mentioned above. In the following, t is the beginning of the considered block and t' is the end of the block, i.e., the time of the next fault after t. Recall that $L_{\mathsf{ALG}}((t,t'])$ is the total size of packets completed by ALG in $(t,t']$. We start with a general lemma that covers all cases in which ADV schedules many ε's.

Lemma 3.23. *In Cases* (B3), (B4), *and* (F3), $L_{\mathsf{ADV}}((t,t']) \geq L_{\mathsf{ALG}}((t,t']) + \varepsilon$ *holds.*

Proof. Let i and τ be as in Case (F3); we set $i = 1$ and $\tau = \tau_1$ in Case (B3), and $i = 2$ and $\tau = \tau_{\geq 2}$ in Case (B4). Note that the first packet of size (at least) ℓ_i is started at τ with $\tau < t + \ell_i/(\phi \cdot s)$ and that the next fault occurs at time $\tau + \ell_i/s - \varepsilon$. Furthermore, $P_{\mathsf{ADV}}(0,t) \geq \phi\ell_k$ by Cases (B1) and (F1). As $t' - t \leq \phi\ell_k$ it follows that ADV has enough ε's to fill nearly the whole block with them, so in particular $L_{\mathsf{ADV}}((t,t']) > t' - t - \varepsilon$.

Let $a = L_{\mathsf{ALG}}((t,t'])$. Since ALG does not complete the ℓ_i-sized packet we have $\tau \geq t + a/s$ and thus also $a < \ell_i/\phi$ as $\tau < t + \ell_i/(\phi \cdot s)$.

If $a < \ell_i/\phi - 3s\varepsilon/\phi$ which is equivalent to $\ell_i > \phi \cdot a + 3s\varepsilon$, then we show the required inequality by the following calculation:

$$L_{\mathsf{ADV}}((t,t']) + \varepsilon > t' - t = \tau + \frac{\ell_i}{s} - \varepsilon - t \geq \frac{a}{s} + \frac{\ell_i}{s} - \varepsilon > \frac{a + \phi \cdot a + 3s\varepsilon}{s} - \varepsilon > a + 2\varepsilon\,,$$

where the last inequality follows from $s < \phi + 1$.

Otherwise, a is nearly ℓ_i/ϕ and thus large enough. Then we get

$$L_{\mathsf{ADV}}((t,t']) + \varepsilon > t' - t = \tau + \frac{\ell_i}{s} - \varepsilon - t \geq \frac{a}{s} + \frac{\ell_i}{s} - \varepsilon > \frac{a}{s} + \frac{\phi a}{s} - \varepsilon > a + 2\varepsilon$$

where the penultimate inequality follows by $\ell_i > \phi a$, and the last inequality holds as $(1+\phi)a/s > a + 3\varepsilon$ for ε small enough and $a \geq \ell_i/\phi - 3s\varepsilon/\phi$. □

For brevity, we inductively define $S_0 = \phi - 1$ and $S_i = S_{i-1} + \ell_i$ for $i = 1, \dots, k$. Thus $S_i = \sum_{j=1}^{i} \ell_i + \phi - 1$ and a calculation shows $S_i = \phi^{i+1} - 1$. We prove a useful observation.

Lemma 3.24. *Fix $j \geq 2$. If Case* (B3) *and Case* (B5) *for $i < j$ are not triggered in the block, then $\tau_{i'+1} \geq t + S_{i'}/s$ for each $i' < j$.*

Proof. We have $\tau_1 \geq t + \ell_1/(\phi \cdot s) = t + (\phi - 1)/s$ by Case (B3) and $\tau_{i+1} \geq \tau_i + \ell_i/s$ for any $i < j$, since Case (B5) was not triggered for $i < j$ and the first ℓ_i-sized packet needs to be finished before starting the next packet. Summing the bounds gives the inequalities in the lemma. □

Lemma 3.25. *In Case* (B5)*, the algorithm does not complete any packet of size ℓ_i or larger.*

Proof. Recall that we have $\tau_{\geq i+1} < \tau_i$, thus the first started packet p of size at least ℓ_i has size at least ℓ_{i+1}. It suffices to prove

$$\tau_{\geq i+1} + \frac{\ell_{i+1}}{s} - t > \ell_i \,, \tag{3.6}$$

which means that p would be completed after the next fault at time $t + \ell_i$.

We start with the case $i = 1$ in which $\tau_{\geq 2} < \tau_1$. Since Case (B4) was not triggered, we have $\tau_{\geq 2} \geq t + \ell_2/(\phi \cdot s) = t + 1/s$. We show (3.6) by the following calculation:

$$\tau_{\geq 2} + \frac{\ell_2}{s} - t \geq \frac{1}{s} + \frac{\ell_2}{s} = \frac{1+\phi}{s} > 1 = \ell_1 \;\;,$$

where the strict inequality holds by $s < \phi + 1$.

Now consider the case $i \geq 2$. By the minimality of i satisfying the condition of Case (B5), we use Lemma 3.24 for $j = i$ and $i' = i - 2$ to get $\tau_{i-1} \geq t + S_{i-2}/s$. Since a packet ℓ_{i-1} is started at τ_{i-1} and must be finished by $\tau_{\geq i+1}$, it holds $\tau_{\geq i+1} \geq t + (S_{i-2} + \ell_{i-1})/s = t + S_{i-1}/s$. Thus

$$\begin{aligned}\tau_{\geq i+1} + \frac{\ell_{i+1}}{s} - t \geq \frac{S_{i-1} + \ell_{i+1}}{s} &= \frac{\phi^i - 1 + \phi^i}{s} \\ &= \frac{\phi^{i+1} + \phi^{i-2} - 1}{s} \geq \frac{\phi^{i+1}}{s} > \phi^{i-1} = \ell_i \,,\end{aligned}$$

where the penultimate inequality holds by $i \geq 2$ and the last inequality by $s < \phi + 1$. (We remark that the penultimate inequality has a significant slack for $i > 2$.) □

Lemma 3.26. *In Case* (B6)*, ALG does not complete a packet of size ℓ_k.*

Proof. It suffices to prove

$$\tau_k > t + \left(1 - \frac{1}{s}\right) \ell_k \,, \tag{3.7}$$

since then ALG completes the first ℓ_k-sized packet at

$$\tau_k + \frac{\ell_k}{s} > t + \left(1 - \frac{1}{s}\right) \ell_k + \frac{\ell_k}{s} = t + \ell_k \,,$$

i.e., after the next fault at time $t + \ell_k$.

Recall that we choose k large enough so that $s < \phi + 1 - 1/\phi^{k-1}$ or equivalently $\phi - 1/\phi^{k-1} > s - 1$. We multiply the inequality by ϕ^{k-1}, divide it by s and add t to both sides and we get

$$t + \frac{\phi^k - 1}{s} > t + \left(1 - \frac{1}{s}\right) \phi^{k-1} = t + \left(1 - \frac{1}{s}\right) \ell_k \,. \tag{3.8}$$

Since Cases (B3) and (B5) are not triggered, we use Lemma 3.24 for $j = k$ to show $\tau_k \geq t + S_{k-1}/s = t + (\phi^k - 1)/s$. We combine this with (3.8) and we have

$$\tau_k \geq t + \frac{\phi^k - 1}{s} > t + \left(1 - \frac{1}{s}\right) \ell_k \,, \tag{3.9}$$

which shows (3.7) and concludes the proof of the lemma. □

Lemma 3.27. *In Case* (F4)*, ALG does not complete any long packet.*

Proof. Recall that the first long packet p is started at τ and it has size of at least ℓ_i, thus it would be completed at $\tau + \ell_i/s$ or later. We show $\tau + \ell_i/s - t > \ell_{i-1}$ by the following calculation:

$$\tau + \frac{\ell_i}{s} - t \geq \frac{\ell_i}{\phi \cdot s} + \frac{\ell_i}{s} = \frac{\phi \ell_i}{s} > \frac{\ell_i}{\phi} = \ell_{i-1} \,,$$

where the strict inequality holds by $s < \phi + 1$. This implies that the long packet p would be completed after the next fault at time $t + \ell_{i-1}$. □

Analysis of the gains We are ready to prove that at the end of the schedule $L_{\mathsf{ADV}} > L_{\mathsf{ALG}} + A$ holds, which contradicts the claimed 1-competitiveness of ALG and proves Theorem 3.22. We inspect all the cases in which the instances may end, starting with Cases (B1) and (F1). We remark that we use only crude bounds to keep the analysis simple.

Lemma 3.28. *If the schedule ends in Case* (B1) *or* (F1)*, we have* $L_{\mathsf{ADV}} > L_{\mathsf{ALG}} + A$.

Proof. Recall that each block $(t, t']$ has length of at most $\phi \ell_k$, thus $L_{\mathsf{ALG}}((t, t']) \leq s\phi\ell_k$ and $L_{\mathsf{ADV}}((t, t']) \leq \phi\ell_k$.

We call a block in which ADV schedules many ε's *small*, other blocks are *big*. Recall that ADV schedules no ε in a big block. Note that Cases (B3), (B4), and (F3) concern small blocks, whereas Cases (B5), (B6), and (F4) concern big blocks.

By Lemma 3.23, in each small block $(t, t']$ it holds that $L_{\mathsf{ADV}}((t, t']) \geq L_{\mathsf{ALG}}((t, t']) + \varepsilon$. Let β be the number of small blocks. We observe that

$$\beta \geq \frac{\left(N_0 - \frac{\phi \ell_k}{\varepsilon}\right) \varepsilon}{\phi \ell_k} \,,$$

because in each such block ADV schedules at most $\phi\ell_k/\varepsilon$ packets of size ε and $P_{\mathsf{ADV}}(0) < \phi\ell_k$ at the end in Cases (B1) and (F1).

The number of big blocks is at most $\sum_{i=1}^{k} N_i$, since in each such block ADV schedules a packet of size at least ℓ_1. For each such block we have $L_{\mathsf{ADV}}((t, t']) - L_{\mathsf{ALG}}((t, t']) \geq -s\phi\ell_k$ which is only a crude bound, but it suffices for N_0 large enough.

Summing over all blocks we obtain

$$\begin{aligned} L_{\mathsf{ADV}} - L_{\mathsf{ALG}} &\geq \beta\varepsilon - \phi s \ell_k \sum_{i=1}^{k} N_i \\ &\geq \frac{\left(N_0 - \frac{\phi \ell_k}{\varepsilon}\right) \varepsilon^2}{\phi \ell_k} - \phi s \ell_k \sum_{i=1}^{k} N_i \\ &> A + \phi s \ell_k \sum_{i=1}^{k} N_i - \phi s \ell_k \sum_{i=1}^{k} N_i = A \;\;, \end{aligned} \tag{3.10}$$

where (3.10) follows from $N_0 > \phi\ell_k(A + 1 + \phi s \ell_k \sum_{i=1}^{k} N_i)/\varepsilon^2$. □

It remains to prove the same for termination by Case (F2), since there is no other case in which the strategy may end.

Lemma 3.29. *If Strategy* Finish *ends in Case* (F2)*, then* $L_{\mathsf{ADV}} > L_{\mathsf{ALG}} + A$.

Proof. Note that ADV schedules all short packets and all ε's, i.e., those of size less than ℓ_i. In particular, we have $L_{\mathsf{ADV}}(< i) \geq L_{\mathsf{ALG}}(< i)$.

Call a block in which ALG completes a packet of size at least ℓ_i *bad.* As the length of any block is at most $\phi\ell_k$ we get that $L_{\mathsf{ALG}}(\geq i, (t,t']) \leq s\phi\ell_k$ for a bad block $(t,t']$. Bad blocks are created only in Cases (B5) and (B6), but in each bad block ADV finishes a packet strictly larger than ℓ_i; note that here we use Lemmas 3.25 and 3.26. Hence the number of bad blocks is bounded by $\sum_{j=i+1}^{k} N_j$. As ADV completes all ℓ_i's we obtain

$$\begin{aligned} L_{\mathsf{ADV}}(\geq i) - L_{\mathsf{ALG}}(\geq i) &\geq \ell_i N_i - \phi s\ell_k \sum_{j=i+1}^{k} N_j \\ &> \ell_i \phi s\ell_k \sum_{j=i+1}^{k} N_j + A - \phi s\ell_k \sum_{j=i+1}^{k} N_j \geq A\,, \end{aligned}$$

where the strict inequality follows from $N_i > \phi s\ell_k \sum_{j=i+1}^{k} N_j + A/\ell_i$ for $i < k$ and from $N_k > A/\ell_k$ for $i = k$. By summing it with $L_{\mathsf{ADV}}(< i) \geq L_{\mathsf{ALG}}(< i)$ we conclude that $L_{\mathsf{ADV}} > L_{\mathsf{ALG}} + A$. □

3.9 Conclusions and Open Problems

In this chapter, we focused on determining the minimum speedup of a 1-competitive deterministic algorithm for Packet Scheduling under Adversarial Jamming. We designed a natural algorithm, which needs speedup of only 4 on general instances, and we proved a lower bound of $\phi + 1 \approx 2.618$ that holds even if all packets are released together. While it might be possible to improve the lower bound, perhaps using release times, we conjecture that it is actually optimal. On the other hand, a promising approach for improving the upper bound, is to adjust our algorithm so that it sends larger packets earlier (in the tight cases of the lower bound of $\phi + 1$, larger packets are executed earlier than in our algorithm's schedule). However, the analysis needs to be modified for the adjusted algorithm as in some cases it seems necessary to charge a large packet to several smaller packets.

Open Problem 8. *Design a (deterministic) algorithm for which a speed below* 4 *is sufficient for* 1*-competitiveness.*

In contrast to studying the speedup sufficient for 1-competitiveness, the previous works [AGK$^+$16, JKL15] resolved the optimal competitive ratio of the problem. However, the optimal lower bound of 3 by Anta *et al.* [AGK$^+$16] is relatively simple as it requires just two sizes and moreover, the optimal algorithm without speedup by Jurdziński *et al.* [JKL15] is quite complicated and does not have some desirable properties. Therefore, we consider studying the speedup sufficient for 1-competitiveness to be a better measure how to compare online algorithms in this model.

Another interesting direction is to study the tradeoff between the speedup and the competitive ratio such as we do for our algorithm using the local analysis framework (cf. Theorem 3.8).

Randomized algorithms. As far as we know, there are no results regarding randomized algorithms for our model. The sole exception is the claim of Anta *et al.* [AGK$^+$16] that their lower bound on the competitive ratio applies to randomized algorithms as well, but their argument works in the adaptive adversary model only. We remark that our lower bounds on the speedup do not hold for randomized algorithms against the adaptive adversary.

Open Problem 9. *Design a randomized algorithm for Packet Scheduling under Adversarial Jamming and analyze its competitive ratio without speedup against the oblivious adversary or its speedup sufficient for* 1*-competitiveness (against either adversary). Furthermore, construct lower bounds on the competitive ratio of randomized algorithms without speedup and on the speedup of* 1*-competitive randomized algorithms.*

General weights. In Packet Scheduling under Adversarial Jamming, packets have weights equal to their sizes. A natural generalization is thus that each packet arrives with a weight in addition to the size and the objective is to maximize the weighted throughput. As in our model, one has to consider the asymptotic competitive ratio, where the additive constant may depend on weights rather than sizes.

Open Problem 10. *Show upper and lower bounds on the optimal competitive ratio of the generalized Packet Scheduling under Adversarial Jamming problem, in which packets have weights and the goal is to maximize the total weight of transmitted packets. Similarly, study the effect of the speedup on the competitive ratio for the generalized model.*

More channels. Some of the previous work in [AGKZ15, JKL15] studied a more general setting in which there are f channels (or machines), where f is a constant. Nevertheless, there is no algorithm for general instances which needs a moderate speedup to be 1-competitive, perhaps with speedup depending on f.

Open Problem 11. *Design a (deterministic) algorithm for a constant number of channels which needs only a constant speed to be* 1*-competitive. On the other hand, is there a lower bound better than* $\phi + 1$ *for* $f > 1$ *channels? Does the optimal speedup depend on* f*?*

Stochastic models. Intuitively, the adversary is quite strong in our model as it controls both packet arrivals and jamming errors. A natural variant of Packet Scheduling under Adversarial Jamming is thus to have stochastic arrivals or stochastic faults (or both). Stochastic arrivals were already considered by Anta *et al.* [AGK$^+$16], who obtained tight bounds on the competitive ratio without speedup for a large range of distributions, but only when there are two packet sizes.

An interesting research direction is thus to study the required speedup for 1-competitiveness in the case of stochastic arrivals or stochastic jamming errors.

Bibliography

[ABEW18] Kamal Al-Bawani, Matthias Englert, and Matthias Westermann. Comparison-based buffer management in QoS switches. *Algorithmica*, 80(3):1073–1092, 2018.

[AGGP17] Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proc. of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC'17)*, pages 551–563. ACM, 2017.

[AGK$^+$16] Antonio Fernández Anta, Chryssis Georgiou, Dariusz R. Kowalski, Joerg Widmer, and Elli Zavou. Measuring the impact of adversarial errors on packet scheduling strategies. *Journal of Scheduling*, 19(2):135–152, 2016. Also appeared in Proc. of SIROCCO 2013: 261–273.

[AGKZ15] Antonio Fernández Anta, Chryssis Georgiou, Dariusz R. Kowalski, and Elli Zavou. Online parallel scheduling of non-uniform tasks: Trading failures for energy. *Theoretical Computer Science*, 590:129–146, 2015. Also appeared in Proc. of FCT 2013: 145-158.

[AGKZ18] Antonio Fernández Anta, Chryssis Georgiou, Dariusz R. Kowalski, and Elli Zavou. Competitive analysis of fundamental scheduling algorithms on a fault-prone machine and the impact of resource augmentation. *Future Generation Comp. Syst.*, 78:245–256, 2018. Also appeared in Proc. of ARMS-CC@PODC 2015, LNCS 9438: 1–16.

[Alb97] Susanne Albers. On the influence of lookahead in competitive paging algorithms. *Algorithmica*, 18(3):283–305, 1997.

[AM03] Nir Andelman and Yishay Mansour. Competitive management of non-preemptive queues with multiple values. In *Distributed Computing (DISC'03)*, volume 2848 of *LNCS*, pages 166–180. Springer Berlin Heidelberg, 2003.

[AMZ03] 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. SIAM, 2003.

[BBK$^+$94] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994. Also appeared in Proc. of STOC 1990: 379–386.

[BCD$^+$13a] 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. Also appeared in Proc. of SODA 2009: 1126–1135.

[BCD$^+$13b] 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.

[BCJ11] 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. Also appeared in Proc. of WAOA 2008: 92–104.

[BCJ$^+$16] Martin Böhm, Marek Chrobak, Łukasz Jeż, Fei Li, Jiří Sgall, and Pavel Veselý. Online packet scheduling with bounded delay and lookahead. In *Proc. of the 27th International Symposium on Algorithms and Computation (ISAAC '16)*, volume 64 of *LIPIcs*, pages 21:1–21:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

[BE98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.

[BJSV18] Martin Böhm, Łukasz Jeż, Jiří Sgall, and Pavel Veselý. On packet scheduling with adversarial jamming and speedup. In *Proc. of the 15th Workshop on Approximation and Online Algorithms (WAOA'17)*, volume 10787 of *LNCS*, pages 190–206, 2018.

[CCF$^+$06] 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. *Journal of Discrete Algorithms*, 4(2):255–276, 2006. Also appeared in Proc. of STACS 2004: 187–198.

[CF03] 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.

[CJST07] Marek Chrobak, Wojciech Jawor, Jiří Sgall, and Tomáš Tichý. Improved online algorithms for buffer management in QoS switches. *ACM Transactions on Algorithms*, 3(4), November 2007. Also appeared in Proc. of ESA 2004: 204–215.

[CY03] Ee-Chien Chang and Chee Yap. Competitive on-line scheduling with level of service. *Journal of Scheduling*, 6(3):251–267, May 2003. Also appeared in Proc. of COCOON 2001: 453–462.

[ES09] Tomáš Ebenlendr and Jiří Sgall. Optimal and online preemptive scheduling on uniformly related machines. *Journal of Scheduling*, 12(5):517–527, Oct 2009. Also appeared in Proc. of STACS 2004: 199–210.

[EW09] Matthias Englert and Matthias Westermann. Lower and upper bounds on FIFO buffer management in QoS switches. *Algorithmica*, 53(4):523–548, 2009. Also appeared in Proc. of ESA 2006: 352–363.

[EW12] 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. Also appeared in Proc. of SODA 2007: 209-218.

[FMN08] Amos Fiat, Yishay Mansour, and Uri Nadav. Competitive queue management for latency sensitive packets. In *Proc. of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 228–237. SIAM, 2008.

[FN17] Moran Feldman and Joseph (Seffi) Naor. Non-preemptive buffer management for latency sensitive packets. *Journal of Scheduling*, 20(4):337–353, Aug 2017. Also appeared in Proc. of IEEE INFOCOM 2010: 1–5.

[Fun10] Stanley P.Y. Fung. Bounded delay packet scheduling in a bounded buffer. *Operations Research Letters*, 38(5):396 – 398, 2010.

[GJL17] P. Garncarek, T. Jurdziński, and K. Loryś. Fault-tolerant online packet scheduling on parallel channels. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 347–356, May 2017.

[GK15] Chryssis Georgiou and Dariusz R. Kowalski. On the competitiveness of scheduling dynamically injected tasks on processes prone to crashes and restarts. *Journal of Parallel and Distributed Computing*, 84:94–107, 2015.

[Gol10] Michael H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, 2010.

[GR14] Michel X Goemans and Thomas Rothvoß. Polynomiality for bin packing with a constant number of item types. In *Proc. of the 25th annual ACM-SIAM symposium on Discrete algorithms (SODA'14)*, pages 830–839. SIAM, 2014.

[Gra66] Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell Labs Technical Journal*, 45(9):1563–1581, 1966.

[Gro95] Edward F. Grove. Online bin packing with lookahead. In *Proc. of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'95)*, pages 430–436. SIAM, 1995.

[Haj01] 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.

[Jeż09] Jan Jeżabek. Increasing machine speed in on-line scheduling of weighted unit-length jobs in slotted time. In *SOFSEM 2009: Theory and Practice of Computer Science*, volume 5404 of *LNCS*, pages 329–340. Springer Berlin Heidelberg, 2009.

[Jeż10] Łukasz Jeż. Randomized algorithm for agreeable deadlines packet scheduling. In *Proc. of the 27th International Symposium on Theoretical Aspects of Computer Science*, volume 5 of *LIPIcs*, pages 489–500. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.

[Jeż13] Łukasz Jeż. A universal randomized packet scheduling algorithm. *Algorithmica*, 67(4):498–515, Dec 2013. Also appeared in Proc. of ESA 2011: 239–250.

[JKL15] Tomasz Jurdziński, Dariusz R. Kowalski, and Krzysztof Loryś. Online packet scheduling under adversarial jamming. In *Proc. of the 12th Workshop on Approximation and Online Algorithms (WAOA'14)*, volume 8952 of *LNCS*, pages 193–206, 2015. See `http://arxiv.org/abs/1310.4935` for missing proofs.

[JLSS12] Łukasz Jeż, Fei Li, Jay Sethuraman, and Clifford Stein. Online scheduling of packets with agreeable deadlines. *ACM Transactions on Algorithms*, 9(1):5:1–5:11, December 2012.

[Kim05] Jae-Hoon Kim. Optimal buffer management via resource augmentation. In *Algorithms and Computation (ISAAC'04)*, volume 3341 of *LNCS*, pages 618–628. Springer Berlin Heidelberg, 2005.

[KLM$^+$04] 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. Also appeared in Proc. of STOC 2001: 520–529.

[KMRS88] Anna R Karlin, Mark S Manasse, Larry Rudolph, and Daniel D Sleator. Competitive snoopy caching. *Algorithmica*, 3(1-4):79–119, 1988. Also appeared in Proc. of FOCS 1986: 244–254.

[KMvS05] Alex Kesselman, Yishay Mansour, and Rob van Stee. Improved competitive guarantees for QoS buffering. *Algorithmica*, 43(1):63–80, Sep 2005. Also appeared in Proc. of ESA 2003: 361–372.

[Kob18] Koji M Kobayashi. An optimal algorithm for 2-bounded delay buffer management with lookahead. *arXiv preprint arXiv:1807.00121*, June 2018.

[KP00a] B. Kalyanasundaram and K. Pruhs. Fault-tolerant real-time scheduling. *Algorithmica*, 28(1):125–144, Sep 2000. Also appeared in Proc. of ESA 1997: 296–307.

[KP00b] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000. Also appeared in Proc. of FOCS 1995: 214–221.

[KWZ17] Dariusz R. Kowalski, Prudence W. H. Wong, and Elli Zavou. Fault tolerant scheduling of tasks of two sizes under resource augmentation. *Journal of Scheduling*, 20(6):695–711, Dec 2017.

[Li09] Fei Li. Improved online algorithms for multiplexing weighted packets in bounded buffers. In *Algorithmic Aspects in Information and Management (AAIM'09)*, volume 5564 of *LNCS*, pages 265–278. Springer Berlin Heidelberg, 2009.

[LSS05] 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. SIAM, 2005.

[LSS07] 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. SIAM, 2007.

[LT99] Tak Wah Lam and Kar-Keung To. Trade-offs between speed and processor in hard-deadline scheduling. In *Proc. of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, pages 623–632. SIAM, 1999.

[MST98] Rajeev Motwani, Vijay Saraswat, and Eric Torng. Online scheduling with lookahead: Multipass assembly lines. *INFORMS J. on Computing*, 10(3):331–340, 1998.

[PSTW02] Cynthia A. Phillips, Clifford Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32:163–200, 2002. Also appeared in Proc. of STOC 1997: 140–149.

[RSSZ13] Andrea Richa, Christian Scheideler, Stefan Schmid, and Jin Zhang. Competitive throughput in multi-hop wireless networks despite adaptive jamming. *Distributed Computing*, 26(3):159–171, Jun 2013. Also appeared in Proc. of DISC 2010.

[Sch16] Kevin Schewior. *Handling Critical Tasks Online: Deadline Scheduling and Convex-Body Chasing*. PhD dissertation, TU Berlin, 2016.

[ST85] Daniel Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985. Also appeared in Proc. of STOC 1984: 488–492.

[VCJS18] Pavel Veselý, Marek Chrobak, Łukasz Jeż, and Jiří Sgall. A ϕ-competitive algorithm for scheduling packets with deadlines. 2018. Submitted.

[ZA16] Elli Zavou and Antonio Fernández Anta. Online distributed scheduling on a fault-prone parallel system. *arXiv preprint arXiv:1603.05939*, March 2016.

[Zav16] Elli Zavou. *Online Scheduling in Fault-prone Systems: Performance Optimization and Energy Efficiency*. PhD thesis, Universidad Carlos III de Madrid, Spain, 2016.

[Zhu04] An Zhu. Analysis of queueing policies in QoS switches. *Journal of Algorithms*, 53(2):137 – 168, 2004. Also appeared in Proc. of SODA 2003: 761–770.

List of Figures