




On packet scheduling with adversarial jamming and speedup

Martin Böhm^{1,2} · Łukasz Jeż⁴ · Jiří Sgall² · Pavel Veselý^{2,3} 

Published online: 4 February 2019
© The Author(s) 2019

Abstract

In Packet Scheduling with Adversarial Jamming, packets of arbitrary sizes arrive over time to be transmitted over a channel in which instantaneous jamming errors occur at times chosen by the adversary and not known to the algorithm. The transmission taking place at the time of jamming is corrupt, and the algorithm learns this fact immediately. An online algorithm maximizes the total size of packets it successfully transmits and the goal is to develop an algorithm with the lowest possible asymptotic competitive ratio, where the additive constant may depend on packet sizes. Our main contribution is a universal algorithm that works for any speedup and packet sizes and, unlike previous algorithms for the problem, it does not need to know these parameters in advance. We show that this algorithm guarantees 1-competitiveness with speedup 4, making it the first known algorithm to maintain 1-competitiveness with a moderate speedup in the general setting of arbitrary packet sizes. We also prove a lower bound of $\phi + 1 \approx 2.618$ on the speedup of any 1-competitive deterministic algorithm, showing that our algorithm is close to the optimum. Additionally, we formulate a general framework for analyzing our algorithm locally and use it to show upper bounds on its competitive ratio for speedups in $[1, 4)$ and for several special cases, recovering some previously known results, each of which had a dedicated proof. In particular, our algorithm is 3-competitive without speedup, matching both the (worst-case) performance of the algorithm by Jurdzinski et al. (Proceedings of the 12th workshop on approximation and online algorithms (WAOA), LNCS 8952, pp 193–206, 2015. http://doi.org/10.1007/978-3-319-18263-6_17) and the lower bound by Anta et al. (J Sched 19(2):135–152, 2016. <http://doi.org/10.1007/s10951-015-0451-z>).

Keywords Packet scheduling · Adversarial jamming · Online algorithms · Throughput maximization · Resource augmentation

Work partially supported by GA ČR Project 17-09142S, GAUK Project 634217, and Polish National Science Center Grants 2016/21/D/ST6/02402 and 2016/22/E/ST6/00499. A preliminary version of this work is in Böhm et al. (2018).

✉ Pavel Veselý
vesely@iuuk.mff.cuni.cz

Extended author information available on the last page of the article

1 Introduction

We study an online packet scheduling model recently introduced by Anta et al. (2016) and extended by Jurdzinski et al. (2015). In our model, packets of arbitrary sizes arrive over time and they are to be transmitted over a single communication channel. The algorithm can schedule any packet of its choice at any time, but cannot interrupt its subsequent transmission. In the scheduling jargon, there is a single machine and no preemptions. There are, however, instantaneous *jamming errors* or *faults* at times chosen by the adversary, which are not known to the algorithm. A transmission taking place at the time of jamming is corrupt, and the algorithm learns this fact immediately. The packet whose transmission failed can be retransmitted immediately or at any later time, but the new transmission needs to send the whole packet, i.e., the algorithm cannot resume a transmission that failed.

The objective is to maximize the total size of packets successfully transmitted. In particular, the goal is to develop an online algorithm with the lowest possible competitive ratio, which is the asymptotic worst-case ratio between the total size of packets in an optimal offline schedule and the total size of packets completed by the algorithm on a large instance. (See the next subsection for a detailed explanation of competitive analysis.)

We focus on algorithms with resource augmentation, namely on online algorithms that transmit packets $s \geq 1$ times faster than the offline optimal solution they are compared against. Such algorithm is often said to be speed- s , running at speed s , or having a speedup of s . As our problem allows constant competitive ratio already at speed 1, we consider the competitive ratio as a function of the speed. This deviates from previous work, which focused on the case with no speedup or on the speedup sufficient for ratio 1, ignoring intermediate cases.

1.1 Competitive analysis and its extensions

Competitive analysis focuses on determining the *competitive ratio* of an online algorithm ALG, which is the supremum over all valid instances I of $\text{OPT}(I)/\text{ALG}(I)$, where $\text{OPT}(I)$ is the optimal profit and $\text{ALG}(I)$ is the profit of ALG on instance I .¹

Note that the optimal solution is to the whole instance. Thus, it can be thought of as being determined by an algorithm that knows the whole instance in advance and has unlimited computational power. For this reason, the optimal solution is sometimes called “offline optimum”. The name “competitive analysis” was coined by Karlin et al. (1988) but this kind analysis was applied even before (Graham 1966; Sleator and Tarjan 1985). Since then, competitive analysis was employed to the study of many online optimization problems, as evidenced by (now somewhat dated) textbook by Borodin and El-Yaniv (1998). A nice overview of competitive analysis and its many extensions in the scheduling context can be found in a survey by Pruhs (2007).

1.1.1 Asymptotic ratio and additive constant

In some discrete optimization problems, such as bin packing or various coloring problems, the standard notion of competitive analysis is too restrictive. The issue is that in order to attain a competitive ratio relatively close to 1 (or even any ratio), an online algorithm must behave

¹ We note that this ratio is always at least 1 for a maximization problem such as ours. However, some authors always consider the reciprocal, i.e., the “alg-to-opt” ratio, which is then at most 1 for maximization problems and at least 1 for minimization problems.

in a predictable way when the current optimal value is still small, which makes the algorithm more or less trivial and the ratio somewhat large. To remedy this, the “asymptotic competitive ratio” is often considered, which means essentially that only instances with a sufficiently large optimal value are considered. This is often captured by stating that an algorithm is R -competitive if (in our convention) there exists a constant c such that $R \cdot \text{ALG}(I) + c \geq \text{OPT}(I)$ holds for every instance I . The constant c is typically required not to depend on the class of instances considered, which makes sense for aforementioned problems where the optimal value corresponds to the number of bins or colors used, but is still sometimes too restrictive.

This is the case in our problem. Specifically, using an example we show that a deterministic algorithm running at speed 1 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., it is not possible to transmit anything later). 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 end of the schedule. In this case, the packet of size ℓ is completed in the optimal schedule, while the algorithm completes no packet again.

1.1.2 Resource augmentation

Moreover, some problems do not admit competitive algorithms at all or yield counterintuitive results. Again, our problem is an example of the former kind if no additive constant depending on packet sizes is allowed (cf. aforementioned example). The latter can be observed in the paging problem, where the optimal ratio equals the cache size, seemingly suggesting that the larger the cache size, the worse the performance, regardless of the caching policy. Perhaps for this reason, already Sleator and Tarjan (1985) considered *resource augmentation* for the paging problem, comparing an online algorithm with cache capacity k to the optimum with cache capacity $h \leq k$. The “resource(s)” depend on the problem at hand. In particular, in case of scheduling problems, the machine speed is a natural choice and was introduced in the seminal paper of Kalyanasundaram and Pruhs (2000). The name “resource augmentation” itself was coined in Phillips et al. (2002).

The article of Kalyanasundaram and Pruhs (2000) gives online algorithms that are constant-competitive when their machine runs at a constant speed $s > 1$ for two fundamental single machine scheduling problems that do not admit constant competitive algorithms in the standard setting with the machine running at speed 1. One of the two problems is a preemptive variant of real-time scheduling where 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. This was followed by numerous studies of similar problems, where one particularly interesting line of research (for multiple machine setting) aims at determining the minimum speedup which suffices for competitive ratio 1 (Phillips et al. 2002; Lam et al. 1999, 2004; Chrobak et al. 2003). An up-to-date overview of these still open problems can be found in the thesis of Schewior et al. (2016).

1.2 Previous and related results

Packet Scheduling with Adversarial Jamming was introduced by Anta et al. (2016), who resolve it for two packet sizes: If $\gamma > 1$ denotes the ratio of the two sizes, then the optimal competitive ratio for deterministic algorithms is $(\gamma + \lfloor \gamma \rfloor) / \lfloor \gamma \rfloor$, which is always in the range $[2, 3)$. Jurdzinski et al. (2015) extend this by proving that the optimal ratio for the case of multiple (though fixed) packet sizes is given by the same formula for the two packet sizes which maximize it.

Moreover, Jurdzinski et al. (2015) give further results for *divisible* packet sizes, i.e., instances in which every packet size divides every larger packet size. In particular, they prove that on such instances speed 2 is sufficient for 1-competitiveness in the resource augmentation setting. (Note that the above formula for the optimal competitive ratio without speedup gives 2 for divisible instances.)

In another work, Anta et al. (2018) consider popular scheduling algorithms and 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 that we aim to maximize, 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.

We note that Anta et al. (2016) demonstrate the necessity of instantaneous error feedback by proving that discovering errors upon completed transmission rules out a constant competitive ratio. They also provide improved results for a stochastic online setting.

1.2.1 Multiple channels or machines

The problem we study has been generalized to multiple communication channels, machines, or processors, 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.

For divisible instances, Jurdzinski et al. (2015) extend their (optimal) 2-competitive algorithm to an arbitrary number of channels. The same setting is studied by Anta et al. (2015), who consider both the completed load and the pending load objectives, and 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. (2015) obtain a tight bound of 2 on speedup for 1-competitiveness for two packet sizes. Moreover, they claim a 1-competitive algorithm with speedup $7/2$ for a constant number of sizes and pending (or completed) load, but the proof is incorrect. See Sect. 3.3 for a (single-channel) counterexample.

Georgiou and Kowalski (2015) 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.

Finally, Garncaek et al. (2017) 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 prove that for two packet sizes, as the number of channels tends to infinity, the optimal ratio tends to $4/3$ in the former setting and to $\phi = (\sqrt{5} + 1)/2 \approx 1.618$ in the latter.

1.2.2 Randomization

All aforementioned results, as well as our work, concern deterministic algorithms. In general, randomization often allows an improved competitive ratio. The idea is simply to replace the algorithm's cost or profit with its expectation in the competitive ratio, but a proper definition is subtle. One may consider the adversary's "strategies" for creating and solving an instance separately, possibly limiting their powers. Formal considerations lead to more than one *adversary model*, which may be confusing. As a case in point, Anta et al. (2016) note that their lower bound strategy for two sizes (in our model) applies to randomized algorithms as well, which would imply that randomization provides no advantage. However, their argument requires that the adversary acts based on the previous behavior of the algorithm, which depends on the algorithm's random bits. This is permitted in the adaptive adversary model but not in the far 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. To our best knowledge, randomized algorithms for our problem were never considered in the oblivious adversary model. For more details and formal definitions of these adversary models, we refer to the article that first distinguished them (Ben-David et al. 1994) or the textbook on online algorithms (Borodin and El-Yaniv 1998).

1.3 Our results

The major contribution of this paper is a uniform algorithm that we call *PrudentGreedy* (PG) and describe in Sect. 2.1. 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 Sect. 4.

However, we start by formulating a simpler (local) analysis framework and applying it to several settings in Sect. 3. 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 $[1, 4)$.

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 Sect. 3.2.3). 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 Sect. 3.4.

On the other hand, we prove that our algorithm PG 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 $\lim_{\alpha \rightarrow \infty} S_\alpha = 2$ (see Sect. 3.2.2 for the precise definition of S_α).

In Sect. 3.3, 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 $4/3$ -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 Sect. 3.4.1]. See Fig. 1 for a graph of our bounds.

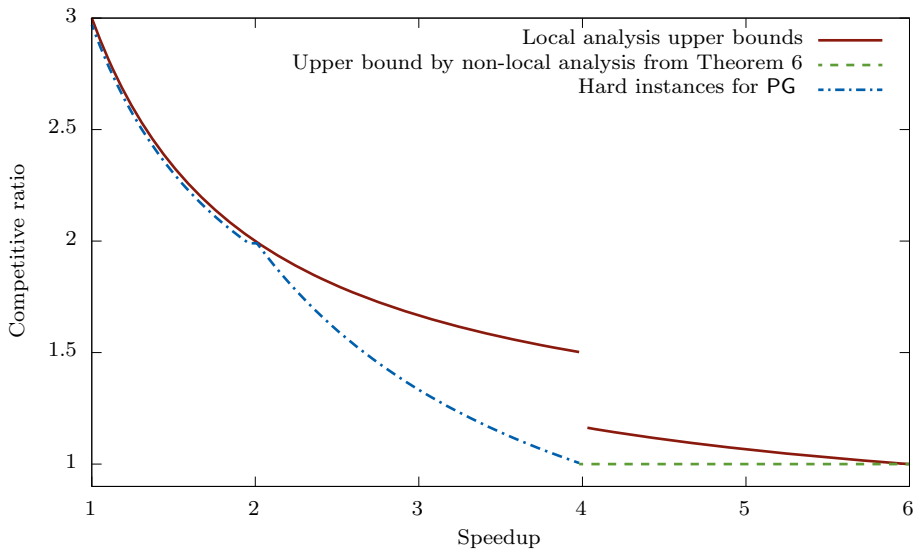


Fig. 1 A graph of our upper and lower bounds on the competitive ratio of algorithm $PG(s)$, depending on the speedup s . The upper bounds follow from Theorems 2 and 6, whereas the lower bounds follow from hard instances from Sect. 3.3

In Sect. 5, 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 Jurdzinski et al. (2015) 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 Sects. 3, 4, and 5 are independent of each other and can be read in any order. In particular, the reader may safely skip proofs for various special instances in Sect. 3 (e.g., the divisible instances), and proceed to Sect. 4 with the main result, which is 1-competitiveness with speedup 4.

1.3.1 Comparison to previous work

Summarizing, our algorithm PG works well in many settings, which we prove using a versatile local analysis framework (except for our main result in Sect. 4 which requires a more intricate analysis). This contrasts with the results of Jurdzinski et al. (2015), where each upper bound is 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 Jurdzinski et al. (2015) 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 Jurdzinski et al. (2015) can be unnecessarily idle if there is a small number of pending packets.

2 Algorithms, preliminaries, notations

We start by some notations. We assume there are k distinct non-zero packet sizes denoted by ℓ_i and ordered such that $\ell_1 < \dots < \ell_k$. For convenience, we define $\ell_0 = 0$. We say that the packet sizes are divisible if ℓ_i divides ℓ_{i+1} for all $i = 1, \dots, k - 1$. For a packet p , let $\ell(p)$ denote the size of p . For a set of packets P , let $\ell(P)$ denote the total size of all the packets in P .

During the execution of an algorithm, at time t , a packet is pending if it is released before or at t and not completed before or at t . At time t , if no packet is running, the algorithm may start any pending packet. As a convention of our model, if a fault (jamming error) happens at time t and this is the completion time of a previously scheduled packet, this packet is considered completed. Also, at the fault time, the algorithm may start any packet, including the one whose transmission has been jammed.

Let $L_{\text{ALG}}(i, Y)$ denote the total size of packets of size ℓ_i completed by an algorithm ALG during a time interval Y . Similarly, $L_{\text{ALG}}(\geq i, Y)$ (resp. $L_{\text{ALG}}(< i, Y)$) denotes the total size of packets of size at least ℓ_i (resp. less than ℓ_i) completed by an algorithm ALG during a time interval Y . Formally, we define $L_{\text{ALG}}(\geq i, Y) = \sum_{j=i}^k L_{\text{ALG}}(j, Y)$ and $L_{\text{ALG}}(< i, Y) = \sum_{j=1}^{i-1} L_{\text{ALG}}(j, Y)$. We use notation $L_{\text{ALG}}(Y)$ with a single parameter to denote the size $L_{\text{ALG}}(\geq 1, Y)$ of packets of all sizes completed by ALG during Y and notation L_{ALG} without parameters to denote the size of all packets of all sizes completed by ALG at any time.

By convention, the schedule starts at time 0 and ends at time T , which is a part of the instance unknown to an online algorithm until it is reached. (This is similar to the times of jamming errors as one can also alternatively say that after T the errors are very frequent and no packet is completed.) Algorithm ALG is called R -competitive if there exists a constant A , possibly dependent on k and ℓ_1, \dots, ℓ_k , such that for any instance and its optimal schedule OPT, we have $L_{\text{OPT}} \leq R \cdot L_{\text{ALG}} + A$. We remark that in our analyses we show only a crude bound on A .

We denote the algorithm ALG with speedup $s \geq 1$ by $\text{ALG}(s)$. The meaning is that in $\text{ALG}(s)$, packets of size L need time L/s to process. In the resource-augmentation variant, we are mainly interested in finding the smallest s such that $\text{ALG}(s)$ is 1-competitive, compared to $\text{OPT} = \text{OPT}(1)$ that runs at speed 1.

2.1 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 execution 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 $\text{rel}(t) = s \cdot (t - t_B)$. Since the algorithm does not insert unnecessary idle time, $\text{rel}(t)$ denotes the total size of transmitted packets in the current phase. Note that we use $\text{rel}(t)$ only when there is no packet running at time t , thus there is no partially executed packet. Intuitively, $\text{rel}(t)$ 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 $\text{rel}(t)$ 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 $\ell_1, \dots, \ell_{i-1}$ 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 ℓ_i and $\ell(P^{<i}) < \ell_i$. Schedule a packet of size ℓ_i and set $t_B = t$.
- (3) Choose the maximum i such that
 - (i) there is a pending packet of size ℓ_i ,
 - (ii) $\ell_i \leq \text{rel}(t)$.
 Schedule a packet of size ℓ_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 p in Step (2) if it has any packets pending. Moreover, if it succeeds in sending p , the length of thus started phase can be related to the total size of the packets completed in it.

Lemma 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 $L_{PG(s)}((t_B, t_E]) > s \cdot (t_E - t_B)/2$, 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. \square

The following lemma shows a crucial property of the algorithm. Namely, 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 2 *Let u be a start of a phase in $PG(s)$ 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 $L_{PG(s)}(< i, (u, v]) < \ell_i + \ell_{i-1}$. (Recall that $\ell_0 = 0$.)*

Proof (i) Suppose for a contradiction that the phase started at u ends at time $t' < t$. We have $\text{rel}(t') < \text{rel}(t) = \ell_i$. 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 $\text{rel}(t') < \ell_j$, and thus Step (3) does not choose a packet to be scheduled. Also note that any packet started before t' is 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, thus j is well-defined and $j \leq i$. Second, all packets of sizes smaller than ℓ_j pending at u are completed before or at t' , implying that their total size is at most $\text{rel}(t') < \ell_j$. Third, the phase started by a packet smaller than ℓ_j at time u . However, this is a contradiction 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 $\ell_i \geq \ell_j$ was pending at u . (Note that it is possible that no packet of size ℓ_j was 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 $\text{rel}(t) + \ell_{i-1} = \ell_i + \ell_{i-1}$. \square

3 Local analysis and results

In this section, we formulate a general method for analyzing our algorithm 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 Jurdzinski et al. (2015), letting us obtain the same upper bounds of 2 and 3 on competitiveness for divisible and unrestricted packet sizes, respectively, at no speedup. Furthermore, we get several new results for the non-divisible cases.

For the analysis, let $s \geq 1$ be the speedup. We fix an instance and its schedules for $\text{PG}(s)$ and OPT .

3.1 Critical times and master theorem

The common scheme is the following. We introduce a sequence of critical times $C_k \leq C_{k-1} \leq \dots \leq C_1 \leq C_0$, where $C_0 = T$ is the end of the schedule, that satisfy the following two informally stated properties: (1) till time C_i the algorithm has completed almost all 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 $L_{\text{OPT}}(i, (0, C_i])$ and $L_{\text{OPT}}(\geq i, (C_i, C_{i-1}])$, respectively, to their “PG counterparts”. Note that each packet of size ℓ_i completed by OPT belongs to exactly one of these sets. Specifically, such packet of size ℓ_i belongs to exactly one of $L_{\text{OPT}}(i, (0, C_i])$, $L_{\text{OPT}}(\geq i, (C_i, C_{i-1}])$, $L_{\text{OPT}}(\geq i-1, (C_{i-1}, C_{i-2}])$, \dots , $L_{\text{OPT}}(\geq 1, (C_1, C_0])$. See Fig. 2 for an illustration. Hence, summing aforementioned bounds yields R -competitiveness of the algorithm for appropriate R and speed s .

We first define the notion of i -good times such that they satisfy property (1), and then choose the critical times among their suprema such that those satisfy property (2) as well.

Definition 1 For $i = 1, \dots, k$, time t is called i -good if one of the following conditions holds:

- (i) At time t , no packet of size ℓ_i is pending for $\text{PG}(s)$,
- (ii) at time t , algorithm $\text{PG}(s)$ starts a new phase by scheduling a packet of size larger than ℓ_i , or

(iii) $t = 0$.

We define critical times C_0, C_1, \dots, C_k iteratively as follows:

- $C_0 = T$, i.e., it is the end of the schedule.
- For $i = 1, \dots, k$, 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 . Also, if no packet of size ℓ_i is pending at time t , then no packet of size ℓ_i is pending during the whole interval $[t, t')$ for some $t' > t$. This, together with 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 $C_i = C_{i-1}$,
- C_i is i -good according to condition (ii) or (iii) in Definition 1, which implies that a phase starts at C_i , 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 it is the supremum of i -good times. This in turn implies that no pending packet of size ℓ_i was released before C_i .

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 *Let $s \geq 1$ be the speedup. Then, for any i , it holds that $L_{OPT}(i, (0, C_i]) \leq L_{PG(s)}(i, (0, C_i]) + \ell_k$.*

Proof If C_i is i -good and satisfies condition (ii) in Definition 1, 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 $PG(s)$ has completed all the packets of size ℓ_i released before C_i , thus the inequality holds trivially even without the additive term. \square

Our remaining goal is to bound $L_{OPT}(\geq i, (C_i, C_{i-1}])$. We divide $(C_i, C_{i-1}]$ into i -segments by the faults. We prove the bounds separately for each i -segment. For the first i -segment, only a loose bound suffices as we can use the additive constant. It is the bound for i -segments started by a fault that is critical, as it determines the competitive ratio. Hence, the latter bound depends on the particular setting. We summarize the general method by the following definition and master theorem.

Definition 2 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 1 (Master Theorem)

Suppose that for $R \geq 1$ both of the following hold:

1. For each $i = 1, \dots, k$ and each proper i -segment $(u, v]$ with $v - u \geq \ell_i$, it holds that

$$(R - 1)L_{PG(s)}((u, v]) + L_{PG(s)}(\geq i, (u, v]) \geq L_{OPT}(\geq i, (u, v]). \tag{3.1}$$

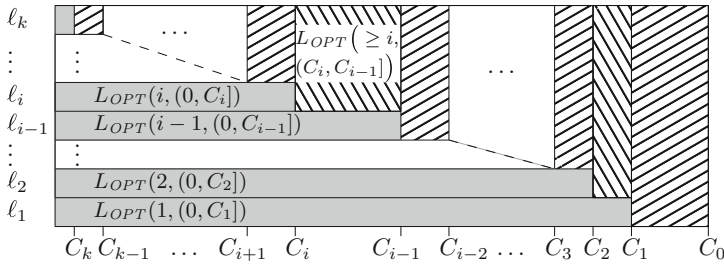


Fig. 2 An illustration of dividing the schedule of OPT in the local analysis, i.e., dividing the (total size of) packets completed by OPT into $L_{OPT}(i, (0, C_i])$ and $L_{OPT}(\geq i, (C_i, C_{i-1}])$ for $i = 1, \dots, k$. Rows correspond to packet sizes and the X-axis to time. Gray “horizontal” rectangles thus correspond to $L_{OPT}(i, (0, C_i])$, 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 “vertical” rectangles correspond to $L_{OPT}(\geq i, (C_i, C_{i-1}])$

2. For the initial i -segment $(u, v]$, it holds that

$$L_{PG(s)}(\geq i, (u, v]) > s(v - u) - 4\ell_k. \tag{3.2}$$

Then $PG(s)$ 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 $L_{OPT}(\geq i, (u, v]) = 0$ and (3.1) is trivial. It follows that (3.1) holds even without the assumption $v - u \geq \ell_i$.

Consider the initial i -segment $(u, v]$. We have $L_{OPT}(\geq i, (u, v]) \leq \ell_k + v - u$, as at most a single packet started before u can be completed. Combining this with (3.2) and using $s \geq 1$, we get $L_{PG(s)}(\geq i, (u, v]) > s(v - u) - 4\ell_k \geq v - u - 4\ell_k \geq L_{OPT}(\geq i, (u, v]) - 5\ell_k$.

Summing this with (3.1) for all proper i -segments and using $R \geq 1$, we get

$$\begin{aligned} (R - 1)L_{PG(s)}((C_i, C_{i-1}]) + L_{PG(s)}(\geq i, (C_i, C_{i-1}]) + 5\ell_k \\ \geq L_{OPT}(\geq i, (C_i, C_{i-1}]). \end{aligned} \tag{3.3}$$

Note that for $C_i = C_{i-1}$, eq.(3.3) holds trivially.

To complete the proof, note that each completed packet in the optimum contributes to exactly one of the $2k$ terms $L_{OPT}(\geq i, (C_i, C_{i-1}])$ and $L_{OPT}(i, (0, C_i])$, and similarly for $L_{PG(s)}$. Thus by summing both (3.3) and Lemma 3 for all $i = 1, \dots, k$, we obtain

$$\begin{aligned} L_{OPT} &= \sum_{i=1}^k L_{OPT}(\geq i, (C_i, C_{i-1}]) + \sum_{i=1}^k L_{OPT}(i, (0, C_i]) \\ &\leq \sum_{i=1}^k \left((R - 1)L_{PG(s)}((C_i, C_{i-1}]) + L_{PG(s)}(\geq i, (C_i, C_{i-1}]) + 5\ell_k \right) \\ &\quad + \sum_{i=1}^k (L_{PG(s)}(i, (0, C_i]) + \ell_k) \\ &\leq (R - 1)L_{PG(s)} + L_{PG(s)} + 6k\ell_k = R \cdot L_{PG(s)} + 6k\ell_k. \end{aligned}$$

□

3.2 Local analysis of PrudentGreedy (PG)

We prove a general lemma which is useful in establishing the preconditions of the Master Theorem. Namely, the first part of the lemma directly implies the precondition (3.2) for the initial i -segments. The second part of the lemma forms the foundation for proving the precondition (3.1) for proper i -segments, for appropriate R and s , which may depend on the instance class. To obtain tight bounds, this part has to be applied carefully, and sometimes be strengthened by leveraging additional structure of the instance class under consideration.

Lemma 4 (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 2(ii), we have $L_{PG(s)}(< i, (u', v]) < \ell_i + \ell_{i-1} < 2\ell_k$. Since at most one packet at the end of the segment is unfinished, we have $L_{PG(s)}(\geq i, (u, v]) \geq L_{PG(s)}(\geq i, (u', v]) > s(v - u') - 3\ell_k \geq s(v - u) - 4\ell_k$.

(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 2(ii). By the definition of C_i , u is not i -good, implying that 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 $L_{PG(s)}((u, v]) > s(v - u)/2$ by Lemma 1. The total size of completed packets smaller than ℓ_i is less than $\ell_i + \ell_{i-1}$ by Lemma 2(ii), and thus $L_{PG(s)}(\geq i, (u, v]) > s(v - u)/2 - \ell_i - \ell_{i-1}$. \square

3.2.1 General packet sizes

The next theorem gives a tradeoff of the competitive ratio of $PG(s)$ and the speedup s using our local analysis. While Theorem 6 shows that $PG(s)$ 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 Anta et al. (2016) shows that no algorithm is better than 3-competitive (for packet sizes 1 and $2 - \varepsilon$). See Fig. 1 for an illustration of our upper and lower bounds on the competitive ratio of $PG(s)$.

Theorem 2 *$PG(s)$ is R_s -competitive where:*

$$\begin{aligned} R_s &= 1 + 2/s \text{ for } s \in [1, 4), \\ R_s &= 2/3 + 2/s \text{ for } s \in [4, 6), \text{ and} \\ R_s &= 1 \text{ for } s \geq 6. \end{aligned}$$

Proof Lemma 4(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 $L_{OPT}(\geq i, (u, v]) \leq v - u$.

For $s \geq 6$, Lemma 4(ii) implies

$$\begin{aligned} L_{PG(s)}(\geq i, (u, v]) &> s(v - u)/2 - 2\ell_i \\ &\geq 3(v - u) - 2(v - u) = v - u \geq L_{OPT}(\geq i, (u, v]), \end{aligned}$$

which is (3.1) for $R = 1$.

For $s \in [4, 6)$, by Lemma 4(ii) we have $L_{PG(s)}((u, v]) > s(v - u)/2$. We multiply it by $(2/s - 1/3)$ and obtain

$$\left(\frac{2}{s} - \frac{1}{3}\right) \cdot L_{PG(s)}((u, v]) > \left(1 - \frac{s}{6}\right)(v - u).$$

Thus, to prove (3.1) for $R = 2/3 + 2/s$, it suffices to show that

$$L_{PG(s)}(\geq i, (u, v]) > \frac{s}{6}(v - u),$$

as clearly $v - u \geq L_{OPT}(\geq i, (u, v])$. The remaining inequality again follows from Lemma 4(ii), but we need to consider two cases:

If $(v - u) \geq \frac{6}{s}\ell_i$, then

$$L_{PG(s)}(\geq i, (u, v]) > \frac{s}{2}(v - u) - 2\ell_i \geq \frac{s}{2}(v - u) - \frac{s}{3}(v - u) = \frac{s}{6}(v - u).$$

On the other hand, if $(v - u) < \frac{6}{s}\ell_i$, then using $s \geq 4$ as well,

$$L_{PG(s)}(\geq i, (u, v]) > \frac{s}{2}(v - u) - 2\ell_i \geq 0.$$

Therefore, $PG(s)$ completes a packet of size at least ℓ_i which implies

$$L_{PG(s)}(\geq i, (u, v]) \geq \ell_i > \frac{s}{6}(v - u),$$

concluding the case of $s \in [4, 6)$.

For $s \in [1, 4)$, by Lemma 4(ii) we get $(2/s) \cdot L_{PG(s)}((u, v]) > v - u \geq L_{OPT}(\geq i, (u, v])$, which implies (3.1) for $R = 1 + 2/s$. □

3.2.2 Well-separated packet sizes

We can obtain better bounds on the speedup sufficient for 1-competitiveness if the packet sizes are substantially different. Namely, we call the packet sizes ℓ_1, \dots, ℓ_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, $PG(S_\alpha)$ is 1-competitive for the following S_α . We define

$$\begin{aligned} \alpha_0 &= \frac{1}{2} + \frac{1}{6}\sqrt{33} \approx 1.46, \text{ which is the positive root of } 3\alpha^2 - 3\alpha - 2. \\ \alpha_1 &= \frac{3 + \sqrt{17}}{4} \approx 1.78, \text{ which is the positive root of } 2\alpha^2 - 3\alpha - 1. \\ S_\alpha &= \begin{cases} \frac{4\alpha + 2}{\alpha^2} & \text{for } \alpha \in [1, \alpha_0], \\ 3 + \frac{1}{\alpha} & \text{for } \alpha \in [\alpha_0, \alpha_1), \text{ and} \\ 2 + \frac{2}{\alpha} & \text{for } \alpha \geq \alpha_1. \end{cases} \end{aligned}$$

This definition follows from the specifics of the proof of Theorem 3 (which follows). We prove that for α -separated packets, $PG(s)$ is 1-competitive if s is at least the maximum of $(4\alpha + 2)/\alpha^2$, $3 + 1/\alpha$, and $2 + 2/\alpha$. Hence, α_0 is chosen as the point where the first two functions cross. Moreover, for $\alpha \geq \alpha_1$, the speedup need not exceed $3 + 1/\alpha$ as the argument in case (viii) of the proof works, which means that for $\alpha \geq \alpha_1$ it is $2 + 2/\alpha$ that attains the maximum value, defining S_α . See Fig. 3 for an illustration.

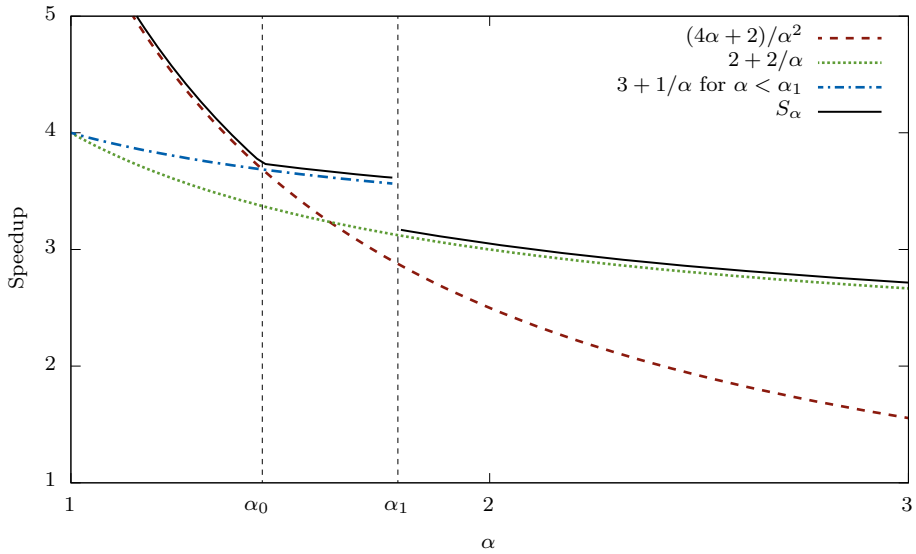


Fig. 3 A graph of S_α and the functions whose pointwise maximum defines it. Note that $3 + 1/\alpha$ need not be exceeded for $\alpha \geq \alpha_1$

Note that S_α is decreasing in α , with a single discontinuity at α_1 . We have $S_1 = 6$, matching the upper bound from Theorem 2. Moreover, $S_2 = 3$, i.e., $PG(3)$ is 1-competitive for 2-separated packet sizes, which includes the case of divisible packet sizes. (However, only $s \geq 2.5$ is needed in the divisible case, as we show later.) The limit of S_α for $\alpha \rightarrow +\infty$ is 2. For $\alpha < (1 + \sqrt{3})/2 \approx 1.366$, we get $S_\alpha > 4$, while Theorem 6 shows that $PG(s)$ is 1-competitive for $s \geq 4$. The weaker result of Theorem 3 below reflects the limits of the local analysis.

Theorem 3 *Let $\alpha > 1$. If the packet sizes are α -separated, then $PG(s)$ is 1-competitive for any $s \geq S_\alpha$.*

Proof Lemma 4(i) implies (3.2). We now prove for any proper i -segment $(u, v]$ with $v - u \geq \ell_i$ that

$$L_{PG(s)}(\geq i, (u, v]) \geq L_{OPT}(\geq i, (u, v]), \tag{3.4}$$

which is (3.1) for $R = 1$. The bound then follows by the Master Theorem.

Let $X = L_{OPT}(\geq i, (u, v])$. Note that $X \leq v - u$.

Lemma 4(ii) together with $\ell_{i-1} \leq \ell_i/\alpha$ gives $L_{PG(s)}(\geq i, (u, v]) > M$ for $M = sX/2 - (1 + 1/\alpha)\ell_i$.

We use the fact that both X and $L_{PG(s)}(\geq i, (u, v])$ are sums of some packet sizes $\ell_j, j \geq i$, and thus only some of the values are possible. However, the situation is quite complicated, as for example, $\ell_{i+1}, \ell_{i+2}, 2\ell_i, \ell_i + \ell_{i+1}$ 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, $L_{PG(s)}(\geq i, (u, v]) > M \geq 0$, which implies $L_{PG(s)}(\geq i, (u, v]) \geq \ell_i = X$. Hence, (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

$$M \geq \frac{s\ell_{i+1}}{2} - \left(1 + \frac{1}{\alpha}\right)\ell_i \geq \left(2 + \frac{1}{\alpha}\right)\ell_i - \left(1 + \frac{1}{\alpha}\right)\ell_i = \ell_i.$$

Hence, $L_{PG(s)}(\geq i, (u, v]) > \ell_i$ which together with $\ell_{i+1} \leq 2\ell_i$ implies $L_{PG(s)}(\geq i, (u, v]) \geq \ell_{i+1} = X$ 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

$$\begin{aligned} \alpha^3 + 1 - 2\alpha^2 &= \alpha^3 - \alpha^2 + 1 - \alpha^2 = \alpha^2(\alpha - 1) - (\alpha + 1)(\alpha - 1) \\ &= (\alpha - 1)(\alpha^2 - \alpha - 1) \leq 0, \end{aligned}$$

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 $\alpha \geq \phi$.

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 $n = 2, 3, \dots$. 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 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 $L_{PG(s)}(\geq i, (u, v])$, we have

$$L_{PG(s)}(\geq i, (u, v]) \geq \ell_{i+1} + \ell_i \geq (\alpha + 1)\ell_i > X$$

by the case condition and (3.4) holds. Otherwise, $L_{PG(s)}(\geq i, (u, v])$ 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 $L_{PG(s)}(\geq i, (u, v]) \geq n \cdot \ell_i = X$ 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

$$L_{PG(s)}((u, v]) > sX - \ell_{i+1} = (s - 1)\ell_{i+1} \geq \left(1 + \frac{2}{\alpha}\right) \ell_{i+1},$$

using $s \geq 2 + 2/\alpha$. Since the total size of packets smaller than ℓ_i is less than $(1 + 1/\alpha)\ell_i$ by Lemma 2(ii), we obtain

$$L_{PG(s)}(\geq i, (u, v]) - X > \frac{2\ell_{i+1}}{\alpha} - \left(1 + \frac{1}{\alpha}\right) \ell_i \geq 2\ell_i - \left(1 + \frac{1}{\alpha}\right) \ell_i > 0,$$

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 $L_{PG(s)}((u, v]) > \ell_{i+2}$. We have $\ell_{i+2} \geq \alpha\ell_{i+1}$ and by the case condition $\ell_{i+1} > 2\ell_i$, we obtain

$$L_{PG(s)}(\geq i, (u, v]) - X > (\alpha - 1)\ell_{i+1} - \left(1 + \frac{1}{\alpha}\right) \ell_i > 2(\alpha - 1)\ell_i - \left(1 + \frac{1}{\alpha}\right) \ell_i \geq 0,$$

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. □

3.2.3 Divisible packet sizes

Now, we turn briefly to even more restricted *divisible instances* considered by Jurdzinski et al. (2015), which are a special case of 2-separated instances. Namely, we improve upon

Theorem 3 in Theorem 4 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 by an example in Sect. 3.3, the bound of Theorem 4 is tight, i.e., $PG(s)$ is not 1-competitive for $s < 2.5$, even on divisible instances.

Theorem 4 *If the packet sizes are divisible, then $PG(s)$ is 1-competitive for $s \geq 2.5$.*

Proof Lemma 4(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 $L_{OPT}(\geq i, (u, v]) \leq v - u$.

By divisibility, we have $L_{OPT}(\geq i, (u, v]) = n\ell_i$ 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

$$L_{PG(s)}(\geq i, (u, v]) > 5(v - u)/2 - \ell_i - \ell_{i-1} - n\ell_i \geq 5n\ell_i/2 - 3\ell_i/2 - n\ell_i \geq (n - 1)\ell_i$$

by Lemma 1 and Lemma 2(ii). Divisibility now implies $L_{PG(s)}(\geq i, (u, v]) \geq n\ell_i = L_{OPT}(\geq i, (u, v])$.

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 $L_{PG(s)}(\geq i, (u, v]) > (n + 1)\ell_i - 3\ell_i/2 \geq (n - 1/2)\ell_i$. Divisibility again implies $L_{PG(s)}(\geq i, (u, v]) \geq n\ell_i = L_{OPT}(\geq i, (u, v])$, which shows (3.1). \square

3.3 Some examples for PG

3.3.1 General packet sizes

Speeds below 2 We show an instance on which the performance of $PG(s)$ matches the bound of Theorem 2.

Remark 1 $PG(s)$ 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 $2 < 4/s - \varepsilon$. 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, thus OPT completes all packets of size 1, while the algorithm has no pending packet of size 1. 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 $\varepsilon \rightarrow 0$. \square

Speeds between 2 and 4 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 6, which we prove later, cannot be improved.

Remark 2 PG(s) has competitive ratio at least $4/s > 1$ for $s \in [2, 4)$.

Proof Choose a large enough integer y . There are four packet sizes: 1, x , y , and z such that $1 < x < y < z$, $z = x + y - 1$, and $x = y \cdot (s - 2)/2 + 2$. Note that $s \in [2, 4)$ implies both $x \geq 2$ and $x \leq y - 1$, the latter for large enough y .

We 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)). 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 starts a packet of size z and fail due to a jam. We show that OPT completes 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, there are jams every 1 unit of time, thus OPT completes all the $N(y - 1)$ packets of size 1, while PG(s) is unable to complete any packet (of size y or larger). The ratio per phase is

$$\frac{\text{OPT}}{\text{PG}(s)} = \frac{y - 1 + y}{y - 1 + x} = \frac{2y - 1}{y - 1 + \frac{y \cdot (s - 2)}{2} + 2} = \frac{2y - 1}{\frac{y \cdot s}{2} + 1}$$

which tends to $4/s$ as $y \rightarrow \infty$. □

This example also disproves the claim of Anta et al. (2015) 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”, in particular, the same as chosen by PG(s).

3.3.2 Divisible case

We give an example that shows that PG is not very good for divisible instances, namely, it is not 1-competitive for any speed $s < 2.5$ and thus the bound in Theorem 4 is tight.

Remark 3 PG(s) has competitive ratio at least $4/3$ 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, whereas the packets of size ℓ arrive at specific times where PG schedules them immediately.

The faults occur at times divisible by 2ℓ , thus 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, PG(s) schedules $2\ell - 1$ packets of size 1, then a packet of size ℓ arrives and is scheduled, and finally, a packet of size 2ℓ is started. The algorithm needs speed $2.5 - 1/(2\ell)$ to complete it. Hence, 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, thus 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. Therefore, the optimum finishes all packets of size 2ℓ plus all small packets, a total of $4\ell - 1$ per phase. The ratio tends to $4/3$ as $\ell \rightarrow \infty$. □

3.4 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 Jurdzinski et al. (2015) 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 ℓ_i and $\ell(P^{<i}) < \ell_i$. Schedule a packet of size ℓ_i and set $t_B = t$.
- (3) Choose the maximum i such that
 - (i) there is a pending packet of size ℓ_i ,
 - (ii) $\ell_i \leq \text{rel}(t)$, and
 - (iii) ℓ_i divides $\text{rel}(t)$.
 Schedule a packet of size ℓ_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 Lemma 1 and 3, 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 1) remains the same. Thus, this section is devoted to leveraging divisibility to prove stronger analogues of Lemma 2 and Lemma 4 (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 $\text{rel}(t) = s \cdot (t - t_B)$ is the relative time after the start of the current phase t_B , scaled by the speed of the algorithm.

Lemma 5 (i) *If PG-DIV starts or completes a packet of size ℓ_i at time t , then ℓ_i divides $\text{rel}(t)$.*

- (ii) *Let t be a time with $\text{rel}(t)$ divisible by ℓ_i and $\text{rel}(t) > 0$. 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 the 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) Suppose that PG-DIV continues running a packet of size ℓ_j at t . By (i), the packet is started at time $t' < t$ with $\text{rel}(t')$ divisible by ℓ_j . Observe that $\ell_j > \ell_i$. Indeed, supposing otherwise, ℓ_j divides t by the assumption, which implies $t' \leq t - \ell_j$. However, this is a contradiction, since the packet of size ℓ_j would be completed by time t .

Next, suppose that PG-DIV starts a new packet. Then the packet of size ℓ_i , which is pending by the assumption, satisfies all the conditions from Step 3 of the algorithm, as $\text{rel}(t)$ is divisible by ℓ_i and $\text{rel}(t) \geq \ell_i$ (from $\text{rel}(t) > 0$ 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 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. The reason is that if a packet is completed at time $\tau \geq t'$, then $\text{rel}(\tau)$ is 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 and (iii) follows from $\ell(P^{<i}) \geq \ell_i$. □

Now we prove a stronger analogue of Lemma 4.

Lemma 6 (i) *If $(u, v]$ is the initial i -segment, then*

$$L_{PG-DIV(s)}(\geq i, (u, v]) > s(v - u) - 3\ell_k.$$

(ii) *If $(u, v]$ is a proper i -segment and $v - u \geq \ell_i$, then*

$$L_{PG-DIV(s)}(\geq i, (u, v]) > s(v - u)/2 - \ell_i.$$

Moreover, $L_{PG-DIV(s)}((u, v]) > s(v - u)/2$ and $L_{PG-DIV(s)}((u, v])$ is divisible by ℓ_i .

Proof We begin with an observation that we use to prove both (i) and (ii): Suppose that time $t \in [u, v)$ satisfies that $\text{rel}(t)$ is divisible by ℓ_i and $\text{rel}(t) > 0$. Then, Lemma 5(ii) together with the fact that a packet of size ℓ_i is always pending in $[u, v)$ (which follows from the definition of critical times and i -segments), implies that from time t on, only packets of size at least ℓ_i are scheduled. In particular, the current phase does not end before v .

For a proper i -segment $(u, v]$, we use the previous observation for $t = u + \ell_i/s$ to prove (ii). Observe that $t \leq v$ by the assumption of (ii). Now $L_{PG-DIV(s)}(< i, (u, v])$ is either equal to 0 (if the phase starts by a packet of size at least ℓ_i at time u), or equal to ℓ_i (if the phase starts by a smaller packet). In both cases, ℓ_i divides $L_{PG-DIV(s)}(< i, (u, v])$ and thus also $L_{PG-DIV(s)}((u, v])$. 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 note that the claim is trivial if $s(v - u) \leq 2\ell_i$. Thus, we assume that $u + 2\ell_i/s \leq v$. We distinguish two cases:

1. The phase of u ends at some time $u' \leq u + \ell_i/s$. Then, by Lemma 5(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 $L_{PG-DIV(s)}(< i, (u, v]) \leq 2\ell_i$.
2. The phase of u does not end by time $u + \ell_i/s$. In this case, there exists $t \in (u, u + \ell_i/s]$ such that ℓ_i divides $\text{rel}(t)$ and also $\text{rel}(t) > 0$ 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 $L_{PG-DIV(s)}(< i, (u, v]) \leq \ell_i$.

In either case, $L_{PG-DIV(s)}(< i, (u, v]) \leq 2\ell_i$ and at most one packet is possibly unfinished at time v . Thus, $L_{PG-DIV(s)}(\geq i, (u, v]) > s(v - u) - 2\ell_i - \ell_k$, and (i) follows. □

Theorem 5 *Let the packet sizes be divisible. Then $PG-DIV(1)$ is 2-competitive. Also, for any speed $s \geq 2$, $PG-DIV(s)$ is 1-competitive.*

Proof Lemma 6(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 $L_{\text{OPT}}(\geq i, (u, v]) \leq v - u$.

For $s \geq 2$, Lemma 6(ii) implies

$$L_{\text{PG-DIV}(s)}(\geq i, (u, v]) > s(v - u)/2 - \ell_i \geq v - u - \ell_i \geq L_{\text{OPT}}(\geq i, (u, v]) - \ell_i.$$

Since both $L_{\text{PG-DIV}(s)}(\geq i, (u, v])$ and $L_{\text{OPT}}(\geq i, (u, v])$ are divisible by ℓ_i , it holds that $L_{\text{PG-DIV}(s)}(\geq i, (u, v]) \geq L_{\text{OPT}}(\geq i, (u, v])$, i.e., (3.1) holds for $R = 1$.

For $s = 1$, Lemma 6(ii) implies

$$\begin{aligned} L_{\text{PG-DIV}}((u, v]) + L_{\text{PG-DIV}}(\geq i, (u, v]) &> (v - u)/2 + (v - u)/2 - \ell_i \\ &\geq v - u - \ell_i \geq L_{\text{OPT}}(\geq i, (u, v]) - \ell_i. \end{aligned}$$

Since $L_{\text{PG-DIV}}((u, v])$, $L_{\text{PG-DIV}}(\geq i, (u, v])$, and $L_{\text{OPT}}(\geq i, (u, v])$ are all divisible by ℓ_i , we have $L_{\text{PG-DIV}}((u, v]) + L_{\text{PG-DIV}}(\geq i, (u, v]) \geq L_{\text{OPT}}(\geq i, (u, v])$, i.e., (3.1) holds for $R = 2$. \square

3.4.1 Example with two divisible packet sizes

We show that neither of our algorithms is better than 2-competitive at speed less than 2, even if there are only two divisible packet sizes in the instance. This matches the upper bound given in Theorem 2 for PG(2) and our upper bounds for PG-DIV(s) on divisible instances, i.e., ratio 2 for $s < 2$ and ratio 1 for $s \geq 2$. We remark that by Theorem 7, 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 4 PG and PG-DIV have ratio no smaller than 2 when $s < 2$, even if packet sizes are only 1 and $\ell \geq \max\{s + \epsilon, \epsilon/(2 - s)\}$ for an arbitrarily small $\epsilon > 0$.

Proof We denote either algorithm by ALG. There are 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 $(2\ell - \epsilon)/s \geq \ell$, where the inequality holds by the bounds on ℓ . Then ALG completes all ℓ packets of size 1, but no packet of size ℓ . By the previous inequality, OPT completes 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 to complete 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. \square

4 PrudentGreedy with speed 4

In this section, we prove that speed 4 is sufficient for PG to be 1-competitive. An example in Sect. 3.3 shows that speed 4 is also necessary for our algorithm.

Theorem 6 $PG(s)$ 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 2(ii) and it has size smaller than $2\ell_i$ by the condition in Step (3). 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 $s < 4$, as exemplified by the instance in Remark 2. The property is important to our proof, as it shows that if the optimal schedule completes a packet of some size and such packet is pending for $\text{PG}(s)$, then $\text{PG}(s)$ completes a packet 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, $\text{PG}(s)$ 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 $1.5 - 2\varepsilon$ are 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 the adversary schedule in the additive constant of the competitive ratio. Note that this shifts the targets of the 1-to-1 charges backward in time.

After the critical time C'_i , packets of size ℓ_i are always pending for the algorithm, and thus (as we noted in the very beginning) the algorithm schedules a packet of size at least ℓ_i when the adversary completes a packet of size ℓ_i . It is more convenient not to work with phases, but rather partition the schedule into blocks between 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 completed by the adversary to algorithm's packets in the same block, such 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. This very issue can be seen in our introductory example. Instead, our analysis resolves the interactions of different blocks by carefully modifying the adversary schedule.

4.1 Blocks, critical times, 1-to-1 charges, and the additive constant

We now formally define the notions of blocks and (modified) critical times.

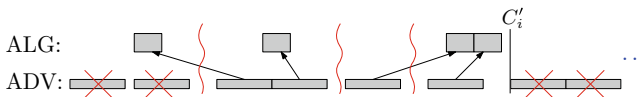


Fig. 4 An illustration of back, up, and forward 1-to-1 charges for ℓ_i -sized packets (other packets are not shown). The winding lines depict the times of jamming errors, which separate blocks. Note that the packets in the algorithm’s schedule are shorter, due to increased speed. They are also higher, thus the area corresponds to the packet size. Crossed packets are included in set A (and thus contribute to the additive constant)

Definition 3 Let f_1, f_2, \dots, f_N be the times of faults. Let $f_0 = 0$ and $f_{N+1} = T$ is the end of the schedule. Then the time interval $(f_i, f_{i+1}]$, $i = 0, \dots, N$, is called a block.

Definition 4 For $i = 1, \dots, k$, 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 1.

All C'_i ’s are defined, as $t = 0$ is i -good for all i . Similarly to Sect. 3.1, 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 the last 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 of size ℓ_i pending for $PG(s)$ and released before C'_i is less than ℓ_k .

Next, we define the set of packets that contribute to the additive constant.

Definition 5 Let set A contain for each $i = 1, \dots, k$:

- (i) the first $\lceil 3\ell_k/\ell_i \rceil$ packets of size ℓ_i completed by the adversary, and
- (ii) the first $\lceil 2\ell_k/\ell_i \rceil$ 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.

For each i , we put into A packets of size ℓ_i of total size at most $7\ell_k$. 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, \dots, 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 $PG(s)$ 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 $n + \lceil 3\ell_k/\ell_i \rceil$ packets of size ℓ_i are started by the adversary, and thus also released. At time C'_i , by its definition, all of those packets are completed by $PG(s)$, with the possible exception of fewer than ℓ_k/ℓ_i packets which may be pending. We charge p_m , for $m = 1, 2, \dots, n$, to the m th packet of size ℓ_i completed by $PG(s)$. Note that each packet started by the adversary is charged at most once and each packet completed by $PG(s)$ 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 Fig. 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 7 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 $PG(s)$ at time t' . Then at any time $\tau \in [t - 2\ell_k, t')$, more than ℓ_k/ℓ_i packets of size ℓ_i are pending for $PG(s)$.

Proof Let m be the number of packets of size ℓ_i that $\text{PG}(s)$ completes before q . Then, by the definition of A , the adversary completes $m + \lceil 3\ell_k/\ell_i \rceil$ packets of size ℓ_i before p (which it starts at time t). The adversary starts less than $2\ell_k/\ell_i$ of these packets in $(t - 2\ell_k, t]$. Thus, more than $m + \ell_k/\ell_i$ of the packets started by the adversary are released before or at time $t - 2\ell_k$. As $\text{PG}(s)$ completed only m packets of size ℓ_i by t' , it has more than ℓ_k/ℓ_i such packets pending at any time $\tau \in [t - 2\ell_k, t')$. \square

Lemma 8 *Let $p \notin A$ be an uncharged packet of size ℓ_i started by the adversary at time t . Then at any time $\tau \geq t - 2\ell_k$, a packet of size ℓ_i is pending for $\text{PG}(s)$.*

Proof Any packet of size ℓ_i started before $C'_i + 2\ell_k$ is either charged or put in A , thus $t - 2\ell_k \geq C'_i$. After C'_i , a packet of size ℓ_i is pending by the definition of C'_i . \square

4.2 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 as follows: We (i) remove from ADV some packets of total size of at most the total size of packets completed by $\text{PG}(s)$ in this block, including all packets in ADV charged to a packet in this block, and (ii) reschedule any remaining packets in ADV in this block to later blocks, while ensuring that ADV is still a feasible schedule. Summing over all blocks, (i) and (ii) guarantee that $\text{PG}(s)$ is 1-competitive with an additive constant $\ell(A)$. Moreover, they ensure that there are no charges to or from a processed block.

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. It also follows that Lemmas 7 and 8 apply to ADV even after rescheduling.

From now on, let $(u, v]$ be the current block that we are processing. All previous blocks ending at $v' \leq u$ are already 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 $\text{PG}(s)$ finishes any packet in the current block or not.

4.2.1 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 $\text{PG}(s)$ completes no packet in this block. It is also not charged by a back charge to a previous block, since there are no charges to already processed blocks. Hence, p is either charged by a forward charge or not charged. Lemma 7 or 8 implies that at time t some packet of size ℓ_i is pending for $\text{PG}(s)$.

Since PG does not idle unnecessarily, this means that some packet q of size ℓ_j for some j is started in $\text{PG}(s)$ at time $\tau \leq t$ and running at t . As $\text{PG}(s)$ 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 $\text{PG}(s)$ in this block, thus it starts a phase.

As this phase is started by packet q of size $\ell_j > \ell_i$, time τ is i -good and $C'_i \geq \tau$. All packets ADV started before time $C'_i + 2\ell_k$ 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 $v < \tau + \ell_j/s < C'_i + \ell_k/s < C'_i + 2\ell_k$, thus it is charged. It follows that p is charged by a forward charge. We now apply Lemma 7 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 contradicts the fact that PG(s) started a phase by q of size $\ell_j > \ell_i$ at τ .

4.2.2 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 , thus 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 such 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 $f : P \rightarrow Q$ 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 9 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 $\ell(p') \leq \ell(p)$. It follows that at the end, we obtain a valid provisional assignment.

Lemma 9 *The assignment process above assigns all packets in P .*

Proof We prove this independently for each packet size.

First, 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 ℓ_j or larger, but in the definition of o , we consider all packets in Q . However, only packets of size at least ℓ_j contribute to o , since packets in P are processed in the decreasing order of sizes.

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 $n - 1$



Fig. 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 $\tau_1, \tau_2,$ and τ_3 are the ends of phases after τ (thus $\alpha = 3$), but τ need not be the end of a phase

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 $PG(s)$ 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 $PG(s)$ 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. Recall that the size of packets in P already assigned is $\ell(f^{-1}(Q)) \geq o/2$, and that there are 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 $v - u \geq \ell(P) + z \geq n\ell_j + o/2 + z$. Consider the schedule of $PG(s)$ in this block. By Lemma 2, there is no end of phase in (u, v) and packets smaller than ℓ_j scheduled by $PG(s)$ have total size less than $2\ell_j$. All the other completed packets contribute to one of $a, o,$ or z . Using Lemma 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 $PG(s)$. Let τ be the supremum of times $\tau' \in [u, v]$ such that $PG(s)$ has no 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 7 or 8, which implies that packets of size ℓ_j are pending for $PG(s)$ from time $t - 2\ell_k$ 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 $t - 2\ell_k$, implying $u < t - 2\ell_k$. The definition of τ now implies that $\tau \leq t - 2\ell_k$.

Towards bounding a , we show that (i) $PG(s)$ completes small packets only of a limited total size 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 $PG(s)$ in $(\tau, v]$ is less than $3\ell_k$. (This claim and its proof are similar to Lemma 2.) Let $\tau_1 < \tau_2 < \dots < \tau_\alpha$ be all the ends of phases in (τ, v) (possibly there is none, then $\alpha = 0$). Also, let $\tau_0 = \tau$. For $i = 1, \dots, \alpha$, let r_i denote the packet started by $PG(s)$ at τ_i . Note that r_i exists since there is a pending packet at any time in $[\tau, v]$ by the definition of τ . See Fig. 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 size 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 $\tau_i, i = 1, \dots, \alpha$, 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 $i = 1, \dots, \alpha$. This also implies that the total size of the small packets completed in $(\tau_0, \tau_1]$ is less than ℓ_k and the claim for $\alpha = 1$ follows. For $\alpha \geq 2$, first note that by Lemma 2(i), $s(\tau_i - \tau_{i-1}) \geq \ell_j$ for all $i = 2, \dots, \alpha$ and thus r_i is not a small packet. Thus, for $i = 3, \dots, \alpha$, the total size 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 size of small packets completed in $(\tau_1, \tau_2]$ is at most $s(\tau_2 - \tau_1) < \ell(r_2)$ and the total size of small packets completed in $(\tau_\alpha, v]$ is at most $2\ell_k - \ell(r_\alpha)$. Thus, the total size 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 $PG(s)$ after τ . After τ , $PG(s)$ processes packets of total size more than $s(v - \tau) - \ell_k$, all of which contribute to one of a, o, z , with the exception of small packets of total size less than $3\ell_k$ by the claim above. Thus, using $s \geq 4$, we get

$$a > 4(v - \tau) - o - z - 4\ell_k. \tag{4.1}$$

Next, we derive two lower bounds on $v - \tau$ using schedule ADV.

Observe that no packet contributing to z is started by ADV before τ as otherwise, it would be pending for $PG(s)$ just before τ , contradicting the definition of τ .

Also, observe that in $(u, \tau]$, ADV runs no packet $p \in P$ with $\ell(p) \geq \ell_j$.

Indeed, for a contradiction, assume that such a p exists. As $\tau \leq C_{j'}$ for any $j' \geq j$, such a p is charged. As $p \in P$, it is charged by a forward charge. Hence, Lemma 7 implies that at all times between the start of p in ADV and v , a packet of size $\ell(p)$ is pending for $PG(s)$. 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 contributing to z . This gives $v - \tau \geq \ell(f^{-1}(Q)) + n\ell_j + z \geq o/2 + n\ell_j + z$.

For the second bound, we note that the n packets of size ℓ_j from P are scheduled in $[t, v]$. Combined with $t \geq \tau + 2\ell_k$, this yields $v - \tau = v - t + t - \tau \geq n\ell_j + 2\ell_k$.

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 (4.1), we obtain $a > 4n\ell_j + z \geq 4n\ell_j$. This completes the proof of the second case. \square

We remark that the first case, which deals with blocks after C_j , is the typical and tight one. The second case, which deals mainly with the block containing C_j and with some blocks before C_j , has a lot of slack, but it is more technically involved. 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 $PG(s)$ 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 definitions of P and Q imply 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. We have $\ell(f^{-1}(q)) \leq \ell(q)$, thus 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 $PG(s)$. The charge was necessarily a forward charge, thus it leads to some later block. See Fig. 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 $(u, v]$, while guaranteeing

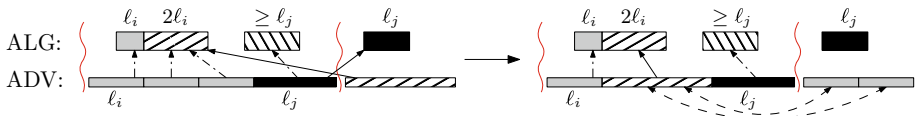


Fig. 6 An illustration of the provisional assignment on the left. Note that a packet of size ℓ_j with a forward charge is also assigned. Full arcs depict 1-to-1 charges and dashed arcs depict the provisional assignment. The result of modifying the adversary schedule on the right

that any remaining charges go to later blocks. This completes processing of the block $(u, v]$ and thus also the proof of 1-competitiveness.

5 Lower bounds

In this section, we study what speed is necessary to achieve 1-competitiveness. We start by revisiting a result of Anta et al. (2015) which applies to a very restricted setting. Namely, it gives a lower bound of 2 for instances with only two divisible packet sizes, proving that our algorithm PG-DIV and the algorithm of Jurdzinski et al. (2015) are optimal. We then extend the construction to a setting with multiple non-divisible packet sizes, for which we show a lower bound of $\phi + 1 \approx 2.618$.

In each proof, we describe a strategy that an adversary uses to create an instance for any algorithm ALG on which ALG is not 1-competitive. This requires comparing the profit of ALG to the optimal profit. As is common, we do not consider the optimal profit directly, but rather use a lower bound on it that follows from a particular offline scheduling algorithm. We think of this scheduling algorithm as a counterpart of the adversary’s strategy, and therefore denote it by ADV.

5.1 Lower bound with two packet sizes

Note that the following lower bound follows from results of Anta et al. (2015) by a similar construction, although the packets in their construction are not released together.

Theorem 7 *No deterministic online algorithm running at speed $s < 2$ is 1-competitive, 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 no fault occurs after t . We set $\tau = \infty$ if ALG never starts such packet. 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.

Fig. 7 An illustration of Case (D3)

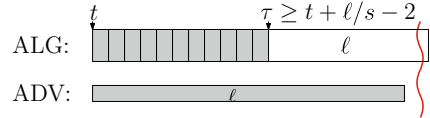
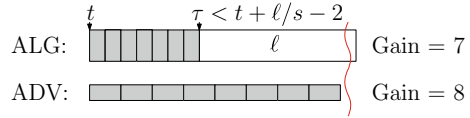


Fig. 8 An illustration of Case (D4)



- (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 the adversary stops the current process and issues faults at times $t + 1, t + 2, \dots$ until ADV, which completes a packet of size 1 between each pair of successive faults, has no 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 $\tau < t + \ell$. This packet 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 $(\frac{2}{s} - 1)\ell > 2$ which is equivalent to $\ell > 2s/(2 - s)$. Thus, the fault occurs before the ℓ -sized packet completes. See Fig. 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 Fig. 8 for an illustration.

First, notice that the process above ends, since in each block, the adversary completes a packet. We now show $L_{ADV} > L_{ALG} + A$ 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 $L_{ADV} = N_1 \cdot \ell + N_0 > A + N_0$. However, as ALG does not complete any ℓ -sized packet, $L_{ALG} \leq N_0$, 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

$$\begin{aligned} \left\lfloor \tau + \frac{\ell}{s} - \varepsilon - t \right\rfloor &\geq \left\lfloor t + \frac{o}{s} + \frac{\ell}{s} - \varepsilon - t \right\rfloor \geq \left\lfloor \frac{o}{s} + \frac{o + 2s}{s} - \varepsilon \right\rfloor = \left\lfloor \frac{2}{s}o + 2 - \varepsilon \right\rfloor \\ &\geq \left\lfloor \frac{2}{s}o + 1 \right\rfloor \geq o + 1, \end{aligned}$$

which proves 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 block, thus $L_{ADV}((u, v]) - L_{ALG}((u, v]) \geq (1 - s) \cdot \ell$ for a block $(u, v]$ created in Case (D3).

Let β be the number of blocks created in Case (D4). We have

$$\beta > \frac{s}{2\ell} \cdot \left(N_0 - \frac{2\ell}{s} \right) = \frac{s \cdot N_0}{2\ell} - 1 \geq N_1 \cdot (s - 1) \cdot \ell + A,$$

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 $L_{ADV}((u, v]) - L_{ALG}((u, v]) \geq 1$ for a block $(u, v]$ created in Case (D4).

Summing over all blocks gives

$$L_{ADV} - L_{ALG} \geq \alpha \cdot (1 - s) \cdot \ell + \beta > N_1 \cdot (1 - s) \cdot \ell + N_1 \cdot (s - 1) \cdot \ell + A = A,$$

where we used $s \geq 1$ which we may suppose w.l.o.g. □

5.2 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 7 for more packet sizes, which are no longer divisible. Still, we make no use of release times.

Theorem 8 *No deterministic online algorithm running at speed $s < \phi + 1$ is 1-competitive, 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 in order to complete all short packets (of size less than ℓ_i for some i), while preventing the algorithm from completing a long packet (of size at least ℓ_i).

Then we show a few lemma 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, such that $s < \phi + 1 - 1/\phi^{k-1}$. For convenience, the smallest size in the instance is ε instead of 1. There are $k + 1$ 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 at speed $s < \phi + 1$ that is 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 $i = 0, \dots, k$. The N_i ’s are chosen such that $N_0 \gg N_1 \gg \dots \gg N_k$. These are all the packets in the instance.

More precisely, N_i ’s are defined inductively, such that $N_k > A/\ell_k$ and $N_i > \phi s \ell_k \sum_{j=i+1}^k N_j + A/\ell_i$ holds for $i = k - 1, \dots, 1$, and finally,

$$N_0 > \frac{A + 1 + \phi \ell_k}{\varepsilon^2} \cdot (\phi s \ell_k \sum_{i=1}^k N_i).$$

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

Fig. 9 An illustration of Case (B5)

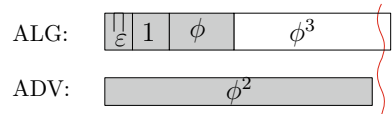
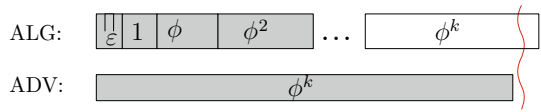


Fig. 10 An illustration of Case (B6)



no fault occurs after t . We set $\tau_i = \infty$ if ALG never starts such packet. 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 $P_{ADV}(i)$ 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 10 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 15).

- (B2) If there is $i \geq 1$ such that $P_{ADV}(i) = 0$, then the adversary 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 $\tau_1 + \ell_1 / s - \varepsilon$, thus 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 10, 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 $\tau_{\geq 2} + \ell_2 / s - \varepsilon$, thus 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 10.

- (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 Fig. 9 for an illustration.

Intuitively, this case means that ALG skips ℓ_i and schedules ℓ_{i+1} (or a larger packet) earlier. Lemma 12 shows that the algorithm cannot finish its first packet of size at least ℓ_{i+1} (nor of size ℓ_i , which it skipped).

- (B6) Otherwise, the next fault occurs at $t + \ell_k$ and ADV schedules a packet of size ℓ_k in this block. Lemma 13 shows that ALG cannot complete an ℓ_k -sized packet in this block. See Fig. 10 for an illustration.

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 $P_{ADV}(i') = 0$. 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 starts 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 $P_{ADV}(0) < \phi \ell_k$, then the end of the schedule is at time t .
- (F2) If ADV has no pending short packet, then the strategy FINISH ends and the adversary issues faults at times $t + \varepsilon, t + 2\varepsilon, \dots$. 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, and hence, there are no short packets whatsoever.
- (F3) If $\tau < t + \ell_i/(\phi \cdot s)$, then the next fault is at time $\tau + \ell_i/s - \varepsilon$, thus 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 10.
- (F4) Otherwise, $\tau \geq t + \ell_i/(\phi \cdot s)$. The adversary issues the next fault at time $t + \ell_{i-1}$. Let j be the largest $j' < i$ such that $P_{ADV}(j') > 0$. ADV schedules a packet of size ℓ_j which is completed as $j \leq i - 1$. Lemma 14 shows that ALG does not complete the long packet started at τ .

Again, ADV completes a packet in each block, 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 lemma 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 $L_{ALG}((t, t'])$ 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 10 *In Cases (B3), (B4), and (F3), $L_{ADV}((t, t']) \geq L_{ALG}((t, t']) + \varepsilon$ 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, $P_{ADV}(0, t) \geq \phi \ell_k$ by Cases (B1) and (F1). As $t' - t \leq \phi \ell_k$, it follows that ε -size packets fill nearly the whole block in ADV. In particular, $L_{ADV}((t, t']) > t' - t - \varepsilon$.

Let $a = L_{ALG}((t, t'])$. 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:

$$L_{ADV}((t, t']) + \varepsilon > t' - t = \tau + \frac{\ell_i}{s} - \varepsilon - t \geq \frac{a}{s} + \frac{\ell_i}{s} - \varepsilon > \frac{a + \phi \cdot a + 3s\varepsilon}{s} - \varepsilon > a + 2\varepsilon,$$

where the last inequality follows from $s < \phi + 1$.

Otherwise, a is nearly ℓ_i/ϕ and thus large enough. Then we get

$$L_{ADV}((t, t']) + \varepsilon > t' - t = \tau + \frac{\ell_i}{s} - \varepsilon - t \geq \frac{a}{s} + \frac{\ell_i}{s} - \varepsilon > \frac{a}{s} + \frac{\phi a}{s} - \varepsilon > a + 2\varepsilon$$

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 $S_i = \sum_{j=1}^i \ell_j + \phi - 1$ and a calculation shows $S_i = \phi^{i+1} - 1$. We prove a useful observation.

Lemma 11 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. \square

Lemma 12 If Case (B5) is triggered for (minimal) i , then 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

$$\tau_{\geq i+1} + \frac{\ell_{i+1}}{s} - t > \ell_i, \tag{5.1}$$

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 (5.1) 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 $i \geq 2$. As i is minimal satisfying the condition of Case (B5), Lemma 11 for $j = i$ and $i' = i - 2$ yields $\tau_{i-1} \geq t + S_{i-2}/s$. Since a packet of size ℓ_{i-1} is started at τ_{i-1} and must be finished by $\tau_{\geq i+1}$, we have $\tau_{\geq i+1} \geq t + (S_{i-2} + \ell_{i-1})/s = t + S_{i-1}/s$. 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 $i > 2$.) \square

Lemma 13 In Case (B6), ALG does not complete a packet of size ℓ_k .

Proof It suffices to prove

$$\tau_k > t + \left(1 - \frac{1}{s}\right) \ell_k, \tag{5.2}$$

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 $t + \ell_k$.

Since Cases (B3) and (B5) are not triggered, Lemma 11 for $j = k$ yields $\tau_k \geq t + S_{k-1}/s = t + (\phi^k - 1)/s$.

Recall that we choose k large enough such that $s < \phi + 1 - 1/\phi^{k-1}$ or equivalently $\phi - 1/\phi^{k-1} > s - 1$. We multiply this inequality by ϕ^{k-1} , divide it by s , and add t to both sides to get

$$t + \frac{\phi^k - 1}{s} > t + \left(1 - \frac{1}{s}\right)\phi^{k-1} = t + \left(1 - \frac{1}{s}\right)\ell_k, \tag{5.3}$$

which shows (5.2). □

Lemma 14 *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 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, $L_{ADV} > L_{ALG} + A$ holds, which contradicts the initial assumption about 1-competitiveness of ALG, proving Theorem 8. 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 15 *If the schedule ends in Case (B1) or (F1), we have $L_{ADV} > L_{ALG} + A$.*

Proof Recall that each block $(t, t']$ has length of at most $\phi \ell_k$, thus $L_{ALG}((t, t']) \leq s\phi \ell_k$ and $L_{ADV}((t, t']) \leq \phi \ell_k$.

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 10, in each small block $(t, t']$ it holds that $L_{ADV}((t, t']) \geq L_{ALG}((t, t']) + \varepsilon$. Let β be the number of small blocks. Then

$$\beta \geq \frac{\left(N_0 - \frac{\phi \ell_k}{\varepsilon}\right)\varepsilon}{\phi \ell_k},$$

because in each such block, ADV schedules at most $\phi \ell_k/\varepsilon$ packets of size ε , and $P_{ADV}(0) < \phi \ell_k$ holds when the strategy ends by Case (B1) or (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 $L_{ADV}((t, t']) - L_{ALG}((t, t']) \geq -s\phi \ell_k$. (This is a crude bound, but sufficient for large enough N_0 .)

Summing over all blocks, we obtain

$$\begin{aligned} L_{ADV} - L_{ALG} &\geq \beta\varepsilon - \phi s \ell_k \sum_{i=1}^k N_i \geq \frac{\left(N_0 - \frac{\phi \ell_k}{\varepsilon}\right)\varepsilon^2}{\phi \ell_k} - \phi s \ell_k \sum_{i=1}^k N_i \\ &> A + \phi s \ell_k \sum_{i=1}^k N_i - \phi s \ell_k \sum_{i=1}^k N_i = A, \end{aligned} \tag{5.4}$$

where (5.4) 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 16 *If Strategy FINISH ends in Case (F2), then $L_{ADV} > L_{ALG} + A$.*

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 $L_{ADV}(< i) \geq L_{ALG}(< i)$.

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 have $L_{ALG}(\geq i, (t, t']) \leq \phi\ell_k$ 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 12 and 13. Hence, the number of bad blocks is bounded by $\sum_{j=i+1}^k N_j$. As ADV completes all packets of size ℓ_i , we obtain

$$\begin{aligned} L_{ADV}(\geq i) - L_{ALG}(\geq i) &\geq \ell_i N_i - \phi s \ell_k \sum_{j=i+1}^k N_j \\ &> \ell_i \phi s \ell_k \sum_{j=i+1}^k N_j + A - \phi s \ell_k \sum_{j=i+1}^k N_j \geq A, \end{aligned}$$

where the strict inequality follows from $N_k > A/\ell_k$ for $i = k$ and from $N_i > \phi s \ell_k \sum_{j=i+1}^k N_j + A/\ell_i$ for $i < k$. By summing it with $L_{ADV}(< i) \geq L_{ALG}(< i)$, we conclude that $L_{ADV} > L_{ALG} + A$. \square

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.


References

- Anta, A. F., Georgiou, C., Kowalski, D. R., Widmer, J., & Zavou, E. (2016). Measuring the impact of adversarial errors on packet scheduling strategies. *Journal of Scheduling*, 19(2), 135–152. Also appeared in *Proceedings of SIROCCO 2013* (pp. 261–273). <http://doi.org/10.1007/s10951-015-0451-z>.
- Anta, A. F., Georgiou, C., Kowalski, D. R., & Zavou, E. (2015). Online parallel scheduling of non-uniform tasks: Trading failures for energy. *Theoretical Computer Science*, 590, 129–146. Also appeared in *Proceedings of FCT 2013* (pp. 145–158). <http://doi.org/10.1016/j.tcs.2015.01.027>.
- Anta, A. F., Georgiou, C., Kowalski, D. R., & Zavou, E. (2018). Competitive analysis of fundamental scheduling algorithms on a fault-prone machine and the impact of resource augmentation. *Future Generation Computer Systems*, 78, 245–256. Also appeared in *Proceedings of the 2nd international workshop on adaptive resource management and scheduling for cloud computing* (ARMS-CC@PODC 2015), LNCS 9438 (pp. 1–16). <http://doi.org/10.1016/j.future.2016.05.042>.
- Ben-David, S., Borodin, A., Karp, R. M., Tardos, G., & Wigderson, A. (1994). On the power of randomization in on-line algorithms. *Algorithmica*, 11(1), 2–14. <https://doi.org/10.1007/BF01294260>.
- Böhm, M., Jež, L., Sgall, J., & Veselý, P. (2018). On packet scheduling with adversarial jamming and speedup. In *Proceedings of the 15th international workshop on approximation and online algorithms (WAOA)* (pp. 190–206). http://doi.org/10.1007/978-3-319-89441-6_15.
- Borodin, A., & El-Yaniv, R. (1998). *Online computation and competitive analysis*. Cambridge: Cambridge University Press.
- Chrobak, M., Epstein, L., Noga, J., Sgall, J., van Stee, R., Tichý, T., et al. (2003). Preemptive scheduling in overloaded systems. *Journal of Computer and System Sciences*, 67, 183–197. [https://doi.org/10.1016/S0022-0000\(03\)00070-9](https://doi.org/10.1016/S0022-0000(03)00070-9).

- Garncarek, P., Jurdziński, T., & Loryś, K. (2017). Fault-tolerant online packet scheduling on parallel channels. In *2017 IEEE international parallel and distributed processing symposium (IPDPS)* (pp. 347–356). <http://doi.org/10.1109/IPDPS.2017.105>.
- Georgiou, C., & Kowalski, D. R. (2015). On the competitiveness of scheduling dynamically injected tasks on processes prone to crashes and restarts. *Journal of Parallel and Distributed Computing*, *84*, 94–107. <https://doi.org/10.1016/j.jpdc.2015.07.007>.
- Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *Bell Labs Technical Journal*, *45*(9), 1563–1581.
- Jurdzinski, T., Kowalski, D. R., & Loryś, K. (2015). Online packet scheduling under adversarial jamming. In *Proceedings of the 12th workshop on approximation and online algorithms (WAOA), LNCS 8952* (pp. 193–206). See <http://arxiv.org/abs/1310.4935> for missing proofs. http://doi.org/10.1007/978-3-319-18263-6_17.
- Kalyanasundaram, B., & Pruhs, K. (2000). Speed is as powerful as clairvoyance. *Journal of the ACM*, *47*(4), 617–643. Also appeared in *Proceedings of the 36nd IEEE symposium on foundations of computer science (FOCS)* (pp. 214–221) (1995). <http://doi.org/10.1145/347476.347479>.
- Karlin, A. R., Manasse, M. S., Rudolph, L., & Sleator, D. D. (1988). Competitive snoopy caching. *Algorithmica*, *3*(1–4), 79–119.
- Lam, T. W., Ngan, T.-W., & To, K.-K. (2004). Performance guarantee for EDF under overload. *Journal of Algorithms*, *52*, 193–206. <https://doi.org/10.1016/j.jalgor.2003.10.004>.
- Lam, T. W., & To, K.-K. (1999). Trade-offs between speed and processor in hard-deadline scheduling. In *Proceedings of the 10th annual ACM-SIAM symposium on discrete algorithms (SODA)* (pp. 623–632). ACM/SIAM. <http://dl.acm.org/citation.cfm?id=314500.314884>.
- Phillips, C. A., Stein, C., Torng, E., & Wein, J. (2002). Optimal time-critical scheduling via resource augmentation. *Algorithmica*, *32*, 163–200. <https://doi.org/10.1007/s00453-001-0068-9>.
- Pruhs, K. (2007). Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, *34*(4), 52–58. <https://doi.org/10.1145/1243401.1243411>.
- Schewior, K. (2016). *Deadline scheduling and convex-body chasing*. PhD dissertation, TU Berlin. <http://doi.org/10.14279/depositononce-5427>.
- Sleator, D. D., & Tarjan, R. E. (1985). Amortized efficiency of list update and paging rules. *Communications of the ACM*, *28*(2), 202–208. <https://doi.org/10.1145/2786.2793>.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Martin Böhm^{1,2} · Łukasz Jeż⁴ · Jiří Sgall^{2,3} 

Martin Böhm
martin.boehm@uni-bremen.de

Łukasz Jeż
lje@cs.uni.wroc.pl

Jiří Sgall
sgall@iuuk.mff.cuni.cz

¹ University of Bremen, Bremen, Germany

² Computer Science Institute of Charles University, Prague, Czech Republic

³ Department of Computer Science, University of Warwick, Coventry, UK

⁴ Institute of Computer Science, University of Wrocław, Wrocław, Poland