Scheduling Multipacket Frames With Frame Deadlines*

Łukasz Jeż\textsuperscript{1,2}, Yishay Mansour\textsuperscript{3,4}, and Boaz Patt-Shamir\textsuperscript{5}

\textsuperscript{1} Eindhoven University of Technology
\textsuperscript{2} Institute of Computer Science, University of Wrocław
\textsuperscript{3} Blavatnik School of Computer Science, Tel Aviv University
\textsuperscript{4} Microsoft Research
\textsuperscript{5} School of Electrical Engineering, Tel Aviv University

Abstract. We consider scheduling information units called frames, each with a delivery deadline. Frames consist of packets, which arrive on-line in a roughly-periodic fashion, and compete on allocation of transmission slots. A frame is deemed useful only if all its packets are delivered before its deadline. Using standard techniques, one can derive polylog-competitive algorithms for this model; in this paper we study special cases which allow for better results. Specifically, we present constant-competitive algorithms for two important cases: in one, the value of a frame is proportional to its size and all frames have (roughly) the same period, and in the other, each frame may have its own period but all frames have the same value and size. The former result also implies better polylog-competitive algorithm for the general case.

1 Introduction

In many networking settings the ingress flows to the network has a nice periodic, or almost periodic, structure. The network would like to guarantee the flows a pre-specified Quality of Service (QoS), where one of the most basic QoS guarantees is a deadline by which the transfer would be completed. The uncertainty regarding the arrival of future flows motivates the online setting. We study this setting from the competitive analysis viewpoint. Let us start by giving a few motivating examples.

Consider a switch with multiple incoming video streaming flows competing for the same output link. Each flow consists of frames, and each frame consists

\* The 1st author is partially supported by the NWO Vidi grant 639.022.211, the Israeli Centers of Research Excellence (I-CORE) program, Center No.4/11, and the Polish National Science Center (NCN) Grant DEC-2013/09/B/ST6/01538. The 2nd author is partially supported by the Israeli Centers of Research Excellence (I-CORE) program, Center No. 4/11, a grant from the Israel Science Foundation (ISF), and a grant from United States-Israel Binational Science Foundation (BSF). The 3rd author is partially supported by the Israel Science Foundation (grant No. 1444/14) and by a grant from Israel Ministry of Science and Technology. This work was carried out while the first author was visiting Tel Aviv University.
of a variable number of packets. The video source is completely periodic, but due to compression, different frames may consist of a different number of packets. On top of that, asynchronous network transfer typically adds some jitter, so the input at the switch is only approximately periodic. In order for a frame to be useful, all its packets must be delivered before the frame’s deadline. A frame is considered completed if all its packets are delivered before the frame’s deadline, and the goal of a scheduling algorithm is to maximize the number of completed frames. Partially completed frames are considered worthless.

As another example, consider a Voice over IP (VoIP) setting. Voice calls generate samples at a relatively fast rate. Samples are wrapped in packets which are aggregated in logical frames with lower-granularity deadlines. Frames deadlines are more lax due to the tolerance of the human ear. Completed frames are reconstructed and replayed at the receiver’s side; incomplete frames are discarded, resulting in an audible interruption (click) of the call. Our focus is on an oversubscribed link on the path of many such calls.

As a last example, consider a database (or data center) engaged in transferring truly huge files (e.g., petabytes of data) for replication purposes. It is common in such a scenario that the transfer must be completed by a certain given deadline. Typically, the transmission of such files is done piecemeal by breaking the file into smaller units, which are transmitted periodically so as to avoid overwhelming the network resources. We are interested in scenarios where multiple such transfers cross a common congested link.

Motivated by the above examples, we define the following abstract model. There are data units called frames, each with a deadline and a value. Each frame consists of several packets. Time is slotted. Packets arrive in an approximately periodic rate at a link, and can be transmitted (served) one packet at a step. A scheduling algorithm needs to decide which packet to transmit at each time slot. The goal of the algorithm is to maximize the total value of delivered frames, where a frame is considered delivered only if all its packets are transmitted before the frame’s deadline.

The scheduling algorithm may be preemptive or non-preemptive. An algorithm is called non-preemptive if any packet it transmits belongs to a frame which is eventually delivered, whereas a preemptive algorithm may transmit a packet from some frame but later decide not to complete that frame.

Our performance measure is the competitive ratio, i.e., the worst case ratio between the value delivered by the online algorithm and the best possible value that can be delivered by an optimal (offline) schedule for a given arrival sequence.

Our Approach and Results. Our model assumes that the arrival sequence is not arbitrary. Studying restricted instance classes and/or adversaries is common, and related work typically assumes specific order of frames and packets or restricted bursts. Instead, we assume that once the first packet of a frame arrives, the arrival times of the remaining packets are predictable within a given bounded jitter. Under this assumption, using the classify and select technique [1], it is relatively straightforward to guarantee a poly-logarithmic competitive ratio, cf. Section 2.2. The conceptual contribution of this work is to identify in-
teresting and important special cases where a constant competitive ratio can be achieved. Moreover, one of them results in improved polylog guarantees for the general case, cf. Section 3. Technically, the main results in this paper are constant-competitive, deterministic algorithms for the following cases.

- All frames have (roughly) the same period but arbitrary sizes, where the size of the frame is the number of its packets. The frame value is its size.
- All frames have the same size but possibly different periods, and they are perfectly periodic (no jitter, frame deadline determined by its period; cf. Section 2). The value of all frames is identical (say, 1).

In fact, the first result is more general: the periods can be arbitrary but the competitive ratio is proportional to the min-to-max period ratio. (And clearly, the same holds in general for “densities” of frames, i.e., their value-to-size ratios.)

We also consider similar case (common period, different size) assuming unit value per frame. By same token, there is a simple randomized algorithm whose competitive ratio (reciprocal) is logarithmic in the maximum number of packets in a frame. We show that in this case a few natural algorithms, such as Earliest Deadline First (EDF) or Shortest Remaining Processing Time (SRPT), cannot guarantee significantly better competitive ratio.

**Related work.** The first multipacket-frame on-line model was introduced in [7], and further studied in [10]. Emek at al. [3] consider the basic model where the main difficulty is not deadlines but rather limited buffer space. Their results express the competitive ratio as a function of the maximum burst size and the number of packets in a frame. Subsequent work considered extension to the basic model, including redundancy [8], and hierarchically structured frames [8, 10].

Possibly the work closest to ours is [9], which essentially uses the same model, except that in [9], each packet has its deadline, and the packet arrivals may be arbitrary (whereas we assume that packets arrive approximately periodically). It is shown in [9] that the competitive ratio of the problem (both a lower and an upper bound) is exponential in the number of packets in a frame. One can view our results as showing that adding the extra assumptions that (1) packet arrival is approximately periodic, and that (2) the deadlines are per frame rather than per packet, allows for significantly better competitive ratio, namely constant.

We note that the classic preemptive job scheduling problem of maximizing (weighted) throughput on a single machine [5, 6, 2] corresponds to a special case of the problem we study in which all frames have period 1 and have no jitter. Thus strong upper bounds (almost tight in the job scheduling problem) follow for the general setting of our problem if frame values are either unit or arbitrary [2]. However, none of the known results, neither upper nor lower bounds, apply or easily extend to special cases of our problem motivated by network applications.

**Paper Organization:** Section 2 introduces the model and a few basic properties. In Section 3, we study the model where the value of a frame is proportional to its size. In Section 4, we consider frames with common size and value but different periods. Section 5 discusses the case of different number of packets for each frame, assuming unit value and identical period. Some proofs are omitted due to lack of space.
2 Model and Preliminary Observations

We consider a standard scheduling model at the ingress of a link. Time is slotted, packets arrive on-line, and in each time slot at most one packet can be transmitted (meaning implicitly that we assume that all packets have the same length). The idiosyncrasies of our model are our assumptions about the arrival pattern and about the way the algorithm is rewarded for delivering packets.

Input: packets and frames. The basic entities in our model are frames and packets. Each frame $f$ consists of $k_f \in \mathbb{N}$ packets, and has a value $v_f \in \mathbb{N}$. We assume that packets of frame $f$ arrive with periodicity $d_f$ and jitter $\Delta_f$, namely if packet 1 of $f$ arrives at time $t$, then packet $i \in \{2, \ldots, k_f\}$ arrives in the time interval $t + (i - 1)d_f \pm \Delta_f$. Each frame $f$ has a slack $s_f \geq 1$, which determines the deadline of $f$ (see “output” paragraph below). A frame $f$ is called perfectly periodic if $\Delta_f = 0$ and $s_f = d_f$. The parameters of a frame $f$ (i.e., size $k_f$, value $v_f$, period $d_f$, jitter $\Delta_f$ and slack $s_f$) are made known to the algorithm when the first packet of $f$ arrives; it is also convenient to introduce a frame’s density, $\rho_f := v_f/k_f$. We denote the actual arrival time of the $i$-th packet of frame $f$, for $i \in \{1, \ldots, k_f\}$, by $t_i(f) \in \mathbb{N}$. The arrival time of the first packet of frame $f$, $t_1(f)$, is also called the arrival time of $f$.

We assume that the algorithm knows nothing about a frame $f$ before its arrival, and even then, it does not know the exact arrival times of the remaining packets: let $\tau_i(f) \overset{\text{def}}{=} t_i(f) + (i - 1)d_f$. Then the guarantee is that the actual arrival time satisfies that $t_i(f) \in [\tau_i(f) - \Delta_f, \tau_i(f) + \Delta_f]$ for $i > 1$.

For a given instance, and a parameter $\pi \in \{\Delta, s, k, d, v, \rho\}$, we let $\pi_{\text{max}} = \max_f(\pi_f)$ and $\pi_{\text{min}} = \min_f(\pi_f)$, both taken over all frames in the instance, and extend these to instance classes. We assume that there is a constant $c \geq 0$ such that $\Delta_f \leq c \cdot s_f$ holds for all frames $f$ (cf. Section 2.1 for its necessity).

Output: delivered frames. A schedule says which packet is transmitted in each time step. The deadline of frame $f$ is $D_f \overset{\text{def}}{=} \tau_{k_f}(f) + \Delta_f + s_f$, and a frame $f$ is said to be delivered in a given schedule if all its packets are transmitted before the frame deadline (we use $s_f$ instead of $s_f - 1$ to reduce clutter later.) Given a schedule, the value delivered by that schedule is the sum of values of frames delivered by that schedule. A schedule is called work conserving if it always transmits a packet if some packet is pending.

Algorithms. The duty of an algorithm is to produce a schedule for any given arrival sequence, and the goal is to maximize the sum of values of delivered frames. An algorithm is called on-line if its decision at any time $t$ depends only on the arrivals and transmissions before time $t$. We assume that the buffer space is unbounded, which means that the only contention is for the transmission slots.

The competitive ratio of an algorithm $A$ is the worst-case ratio, over all arrival sequences $\sigma$, between the value delivered on $\sigma$ by $A$ and by the optimal off-line schedule. Formally, the competitive ratio of $A$ is

$$\rho(A) \overset{\text{def}}{=} \inf_{\sigma \in \mathcal{M}(\pi_{\text{max}}, \Delta_{\text{max}})} \frac{A(\sigma)}{\text{OPT}(\sigma)}$$
where $A(\sigma)$ and $\text{OPT}(\sigma)$ denote the gain of $A$ on $\sigma$ and the optimum gain on $\sigma$ respectively, $M(s_{\text{max}}, \Delta_{\text{max}})$ is the set of arrival sequences with jitter at most $\Delta_{\text{max}}$ and slack of at most $s_{\text{max}}$. Note that $\rho(A) \in [0, 1]$ by definition.

2.1 On the Relation between $s$ and $\Delta$

Some settings of the parameters are uninteresting. In particular, we observe that if $s \ll \Delta$ (in words: the slack is much smaller than the input jitter), then one cannot expect good worst-case performance from any on-line algorithm, even if all frames have identical period, jitter, slack, value, and size. Specifically, we show that in such a case, denoting the common frame size by $k$, every on-line algorithm has competitive ratio at most $O(1/k)$, and that $\Omega(1/k)$-competitiveness is easily achievable if $k < s$ (see appendix for proofs).

**Theorem 1.** No randomized algorithm on instances with all frames of size $k$, jitter $\Delta$, slack $s$, and period $d \geq 2\Delta + s$ has competitive ratio larger than $s + 2\Delta/k$. 

Note that Theorem 1 is meaningless for instances with 0 jitter.

**Theorem 2.** If all frames have size $k$ and each frame $f$ has slack $s_f > k$, then there exists a $1/(2k)$-competitive deterministic on-line algorithm.

Theorems 1 and 2 motivate our assumption that $\Delta_f/s_f$ is bounded from above by a constant: otherwise there is no way to attain a non-trivial competitive ratio.

2.2 Uniform Instances and Polylog Competitiveness

For a tuple $\Pi_n = (\pi_1, \pi_2, \ldots, \pi_n)$ of frame parameters, such as size, value, period, or density, and a set $\Gamma_n = (\gamma_1, \gamma_2, \ldots, \gamma_n)$ of real numbers no smaller than 1, we call an instance $(\Pi, \Gamma)$-uniform if for every $1 \leq i \leq n$, the ratio of the max-to-min value of parameter $\pi_i$ over all frames in the instance is at most $\gamma_i$. In case of uniform instances, we generally assume that the extreme values of frame parameters in $\Pi$ are known to the algorithm. In such case, using the classify and randomly select paradigm [1] extends any algorithm for nearly uniform instances to general instances in the following sense.

**Lemma 1.** Let $\gamma > 1$ and let $A$ be a $\rho$-competitive algorithm for $((\pi), (\gamma))$-uniform instances. Then, given a $((\pi), (\gamma'))$-uniform class of instances $\mathcal{I}$ with $\pi_{\text{min}}$ and $\pi_{\text{max}}$ the minimum and maximum values of $\pi$ in $\mathcal{I}$, $B(A, \pi_{\text{min}}, \pi_{\text{max}})$ (defined below) is a $\left(\rho/(\lceil \log_\gamma \gamma' \rceil + 1)\right)$-competitive randomized algorithm for $\mathcal{I}$.

**Algorithm $B(A, \pi_{\text{min}}, \pi_{\text{max}})$:**
1. Classify each frame $f$ in class $\lceil \log_\gamma \pi_f - \log_\gamma \pi_{\text{min}} \rceil$.
2. Randomly select a class $i$ with uniform distribution, and run $A$ on frames of this class only, discarding packets of frames from all other classes.

**Proof.** The expected contribution of the chosen class of frames to the optimum throughput is clearly $1/\xi$, where $\xi = \lceil \log_\gamma \gamma' \rceil + 1$ is the number of classes. $\square$
Lemma 1 can be applied iteratively over successive parameters, yielding ratio 
\( \Omega(1 / \prod_{i=1}^n \log_\gamma \gamma'_i) \) for \((\Pi_n, \Gamma_n')\)-uniform instances if only we have a constant-
competitive algorithm for \((\Pi_n, \Gamma_n)\)-uniform instances. Fortunately, there is a 
simple \( \Omega(1) \)-competitive deterministic algorithm for instances that are nearly 
uniform in terms of frame size, value, and period. As a warm-up, to illustrate 
our approach, we state such algorithm instances with no jitter and slack \( s \geq d_{\text{min}} \).

The state of the algorithm consists of a set of up to \( d_{\text{min}} \) active frames, 
initially empty. (Recall that \( d_{\text{min}} \), the minimum period of frames in the instance, 
is known to the algorithm.) When a new frame arrives, it enters the set of active 
frames iff there are strictly less than \( d_{\text{min}} \) active frames at the time. A frame 
remains active until its deadline. The algorithm transmits available packets of 
active frames in FIFO order, and discards all packets of all inactive frames.

**Theorem 3.** The algorithm above is \( \left( \frac{2 \cdot k_{\text{max}} \cdot d_{\text{max}} \cdot v_{\text{max}}}{k_{\text{min}} \cdot d_{\text{min}} \cdot v_{\text{min}}} + 1 \right)^{-1} \) -competitive. 
Moreover, each packet of an active frame is transmitted within \( d_{\text{min}} \) steps of 
its arrival.

**Proof.** We begin with proving that each packet of an active frame is transmitted 
within \( d_{\text{min}} \) steps of its arrival. Suppose it does not hold, and let \( p \) be the first 
packet for which it fails. Then \( p \) is delayed by at least \( d_{\text{min}} \) active packets that 
were already in the buffer when it arrived. This implies that there are more than 
\( d_{\text{min}} \) active packets (counting \( p \) as well), so two of them must belong to the same 
frame \( f \). This is a contradiction to the choice of \( p \), since the earlier of those 
packets could not have been transmitted within \( d_f \geq d_{\text{min}} \) steps of its arrival.

We prove the competitive ratio by a charging scheme. For simplicity, we 
ignore frame values: as the worst case is that each frame of OPT has value \( v_{\text{max}} \) 
whereas each frame of the algorithm \( v_{\text{min}} \), this contributes the \( \frac{v_{\text{min}}}{v_{\text{max}}} \) factor to the 
competitive ratio. Firstly, each frame completed by both OPT and the algorithm 
is charged to itself. Moreover, each active frame \( f \), accepted at its arrival time 
\( t \), provides a credit of \( \frac{1}{k_{\text{min}} \cdot d_{\text{min}}} \) to each time slot in \([ t, D_f + k_{\text{max}} \cdot d_{\text{max}} ]\). 
Each \( f \) thus provides a credit of \( \frac{1}{k_{\text{min}} \cdot d_{\text{min}}} \cdot \frac{k_{\text{min}} \cdot d_{\text{min}} + k_{\text{max}} \cdot d_{\text{max}}}{k_{\text{min}} \cdot d_{\text{min}}} \) \( \leq 2 \cdot \frac{k_{\text{max}} \cdot d_{\text{max}}}{k_{\text{min}} \cdot d_{\text{min}}} \). Taking the self-
charges into account, this establishes the ratio. It remains to show how frames 
completed by OPT but rejected by the algorithm are charged to the credit. Let 
\( f' \) be such frame and \( t' \) be its arrival time. Then each packet of \( f' \) charges \( 1/k_{\text{f'}} \) to 
the credit of the time slot in which OPT sends it out. As \( f' \) is rejected, there 
were \( d_{\text{min}} \) active frames at time \( t' \), each of them contributing credit to each slot 
in \([ t', t'+k_{\text{max}} \cdot d_{\text{max}} ] \supseteq [ t', D_{f'} ] \). Hence, each slot that \( f' \) may charge to receives 
a credit of at least \( 1/k_{\text{min}} \geq 1/k_{\text{f'}} \). The theorem follows. \( \square \)

In the next section, we give an improved algorithm for instances that may 
have (larger) jitter and smaller slack, and that are nearly uniform in frame period 
and density. I.e., not only is the class of instances less restrictive in terms of jitter 
and slack, but also the extension to general instances via iterative application of 
Lemma 1 results in improved competitive ratio. Specifically, rather than losing 
a \( \log(v_{\text{max}}/v_{\text{min}}) \cdot \log(k_{\text{max}}/k_{\text{min}}) \) term for value and size parameters, we only
lose a \( \log(\rho_{\text{max}}/\rho_{\text{min}}) = \log(v_{\text{max}}/v_{\text{min}}) + \log(k_{\text{max}}/k_{\text{min}}) \) term in the competitive ratio, for density, which combines size and value.

3 Similar periods, uniform density

In this section we consider \(((d, \rho), (\delta, 1))\)-uniform instances, i.e., with periods between \(d_{\text{min}}\) and \(d_{\text{max}} = \delta d_{\text{min}}\) and uniform density, assumed to be 1. We give an algorithm with competitive ratio depending on \(c, \delta\), and \(\alpha := \max(0, \frac{\Delta_{\text{max}} + s_{\text{max}} - s_f}{d_{\text{max}}} - 1)\), i.e., \(\Omega(1)\)-competitive when all these are bounded by constants.

3.1 The Algorithm

Our approach is as follows. A packet is said to be of type 1 if it must be transmitted in less than \(d_{\text{min}}\) steps since its latest possible arrival time; other packets are type 2. Type 1 packets are exactly all last packets of frames whose slack is smaller than \(d_{\text{min}}\). Packets of the two types will be scheduled differently. We extend these types to frames and let them inherit the types of their last packets. At every point in time, the algorithm maintains up to \(d_{\text{min}}/2\) active frames. The algorithm guarantees that each type 2 packet of an active frame is delivered within the \(d_{\text{min}}\) steps following its latest possible arrival time. Limiting the number of active frames makes this invariant easy to maintain using greedy scheduling, but this cannot be applied to type 1 packets, because these must be transmitted in fewer than \(d_{\text{min}}\) steps after their latest possible arrival. To schedule type 1 packets, the algorithm maintains explicit slot reservations. To make sure that these do not interfere with type 2 packets, type 1 frames remain (quasi-)active for a short time after their completion and prevent accepting new type 1 size 1 frames, which could result in delaying type 2 packets too much.

Algorithm specification. The algorithm maintains a set \(\text{Act}\) of up to \(d_{\text{min}}/2\) active frames. Each active frame \(f\) with \(s_f < d_{\text{min}}\) has a reserved slot for its last packet in the interval \([\tau_k(f) + \Delta_f, \tau_k(f) + \Delta_f + s_f]\). The algorithm consists of two subroutines. Subroutine A decides, for each new frame \(f\), whether to add it to \(\text{Act}\) or not. In the former case we say that \(f\) is accepted, and in the latter that \(f\) is rejected. When a frame \(f\) is accepted, the algorithm may remove a previously active frame \(f'\) from \(\text{Act}\), in which case we say that \(f'\) is preempted. For conciseness, Subroutine A always preempt some \(f'\) when a new frame \(f\) is accepted, but \(f'\) may be “virtual”, in which case so is the preemption. We also maintain a set \(\text{Act}_1\) where active frames of type 1 remain for \(d_{\text{min}} - 1\) steps after they have been completed. This set, rather than \(\text{Act}\), determines whether an arriving type 1 frame of size 1 is accepted. All packets of non-active frames (those rejected or preempted) are dropped. Subroutine S schedules packets, deciding which one to transmit next. The following notions are used in the subroutines:
\[
S_f \overset{\text{def}}{=} \left[ \tau_k(f) + \Delta_f, \tau_{k'}(f) + \Delta_f + s_f \right] \quad \text{slack interval of type 1 frame } f
\]
\[
D_f(i) \overset{\text{def}}{=} \tau_i(f) + \Delta_f + d_{\min} \quad \text{deadline of packet } i \text{ of type 2 frame } f
\]
\[
I_f(i) \overset{\text{def}}{=} [\tau_i(f) + \Delta_f, D_f(i)) \quad \text{designated interval of packet } i \text{ of type 2 frame } f
\]

**Subroutine A** Upon arrival of a new frame \(f\):
- If \(f\) is type 1 and \(k_f = 1\) and \(|\text{Act}_1| \geq d_{\min}/2\): reject \(f\) and return.
- (Otherwise) If \(f\) is type 1 (and \(k_f > 1\)) and all slots in \(S_f\) are reserved:
  - let \(f'\) be the smallest frame with a reserved slot in \(S_f\)
- Else:
  - let \(f'\) be a virtual type 2 frame of size 0
  - If \(f'\) has size 0 and \(|\text{Act}| \geq d_{\min}/2\), let \(f'\) be the smallest frame in \(\text{Act}\).
  - If \(k_f < 2k_{f'}\): reject \(f\) and return
  - (Otherwise):
    - If \(f'\) is type 1, remove \(f'\) from \(\text{Act}_1\) and cancel its reservation for last packet
    - If \(f\) is type 1, add it to \(\text{Act}_1\) and make a reservation for its last packet in \(S_f\)
    - remove \(f'\) (if real) from \(\text{Act}\), add \(f\) to \(\text{Act}\), and return

**Subroutine S** In each step \(t\):
- If slot \(t\) is reserved for the last packet \(p\) of a frame \(f\):
  - remove \(f\) from \(\text{Act}\) now and mark it for deletion from \(\text{Act}_1\) at time \(t + d_{\min}\)
  - transmit \(p\) and return
- Else:
  - let \(p\) be the earliest deadline packet in \{packet \(i\) of \(f \in \text{Act} \land t \in I_f(i)\}\)
  - if \(p\) is the last packet of a frame \(f\), remove \(f\) from \(\text{Act}\)
  - transmit \(p\) and return

### 3.2 Analysis

Intuitively, the analysis is an extension of Theorem 3, whose two claims correspond to Theorem 4 and Lemma 2 respectively. Proving these is somewhat more involved: the latter due to the extra constraints and special treatment of type 1 packets, and the former due to varying sizes (and values) of frames.

**Lemma 2.** Every packet \(p\) of an active frame is sent out during its reserved slot if it is type 1 or during its designated interval \(I\) if it is type 2.

To analyze the competitive ratio, we define chains of frames inductively as follows. Each completed frame \(f\) is in a distinct chain \(C_f\), and if a frame \(f'\) was
preempted by a frame \( f \), and \( f \) is in a chain \( C \), then \( f' \) belongs to \( C \) as well, preceding \( f \) in it. All chains start with a frame that did not preempt any other frame, and end with a frame that was not preempted. We note that our chains are virtually the same as in the analyses of online interval scheduling [11, 4], and part of our analysis is reminiscent of those.

The high level overview of the charging scheme is as follows. There are three kinds of charges: a self-charge of \( f \) to itself if both OPT and the algorithm completed it and two further kinds of charges for the frames completed only by OPT. Here, we distinguish the cause of rejection. If \( f \) is a type 2 frame or a type 1 frames of size 1, it has been rejected due to too many active frames in \( \text{Act} \) and \( \text{Act}_1 \) respectively. Then each active frame from the respective set had at least half the size of \( f \), so \( f \) can be charged to any of such frames. If \( f \) is type 1 of size greater than 1, then \( f \) has been rejected due to lack of slots for its last packet in \( S_f \). Namely, each slot in \( S_f \) was reserved for a last packet of at least half \( f \)'s size, since otherwise \( f \) would preempt the smallest of those. Thus \( f \) can be charged to one of those frames. Note that in both cases the frame we charge to may not be completed by the algorithm in the end. But as it is a part of some chain, and frame sizes in a chain increase geometrically, all charges can be relayed to the last frames of chains, which the algorithm completes. For both kinds of charges, we show that globally there are sufficiently many active frames to be charged, rather than identify a particular active frame to be charged. To this end, both charges are towards a “credit” that the chain(s) provide, and in the end, this credit is charged to the last frame of a chain. We note that the jitter of last packets of frames effectively contribute to the frame sizes; as the jitter does not scale with frame size, the maximum effective sizes of frames preceding the last one in a chain do not form an exact geometric progression.

**Theorem 4.** The algorithm is \((2(5 + 2c + 4\delta + 2\alpha\delta))^{-1}\)-competitive.

**Proof.** We define *chains* of frames. Each completed frame \( f \) defines a chain \( C_f \) that ends with \( f \). Moreover, if a frame \( f \) that belongs to a chain \( C \) preempted a frame \( f' \), then \( f' \) belongs to \( C \) as well, preceding \( f \) in it; if \( f \) did not preempt any frame, then the chain \( C \) starts with \( f \).

Let us now define the credits associated with chains. For a given chain \( C \), let \( f'_C \) and \( f_C \) denote its first and last frame respectively, and let \( T(f_C) \) denote the time \( f_C \) was removed from both \( \text{Act} \) and \( \text{Act}_1 \). In other words, \( T(f_C) \) is the completion time of \( f_C \) if it is type 2, or its completion time plus \( d_{\text{min}} - 1 \) if it is type 1. We give a credit of \( 2/d_{\text{min}} \) to all time slots since the arrival of \( f'_C \) until \( 2(k_{f_C} - 1)d_{\text{max}} + \Delta_{\text{max}} + s_{\text{max}} \) time slots past \( T(f_C) \), i.e., to \( [t_1(f'_C), T(f_C) + 2(k_{f_C} - 1)d_{\text{max}} + (\Delta_{\text{max}} + s_{\text{max}})] \). We stress that the credits granted to a time slot from different chains add up.

We are now ready to describe the preliminary charging scheme, i.e., the charges that are later relayed to last frames of chains. Let \( f \) be a frame delivered by OPT. The charging is as follows:

1. If \( f \) was accepted by the algorithm, \( f \) is charged to itself.
2. If \( f \) was rejected by the algorithm due to lack of slot for its last packet, \( f \) is charged to the frames that prevented its acceptance; details are given later.

3. If \( f \) was rejected by the algorithm due to too many active frames, \( f \) is charged as follows: for each packet \( p \) of \( f \), charge \( p \) to the credit associated with the time slot in which OPT sends \( p \). Each such slot has a credit of at least 1.

When \( f \) arrived at time \( t_1(f) \), the algorithm had \( d_{\text{min}}/2 \) active frames, each of size at least \( k_f/2 \). (If \( f \) is type 1 of size 1, these are the frames from \( \text{Act}_1 \).) Thus our credit rule guarantees that each slot in \([t_1(f), t_1(f) + (k_f - 1)d_{\text{max}} + \Delta_f + s_f)\), i.e., from \( t_1(f) \) until the deadline of \( f \), receives a credit of \( 2/d_{\text{min}} \) from each of the \( d_{\text{min}}/2 \) chains corresponding to the active frames.

We now describe the charging for a frame \( f \) that was rejected due to lack of reservation space for the last packet. Then at \( f \)'s arrival time, \( t_1(f) \), all the slots that \( f \)'s last packet could have used were already reserved for other frames, all of size at least \( k_f/2 \). We charge \( f \) to those frames as follows.

Let \( A_i \) denote the set of frames of size at least \( i \) that OPT delivers and the algorithm rejects due to lack of slot for their last packets. Consider all maximal intervals \( L_1^i, L_2^i, \ldots, L_m^i \) of time such that \( \bigcup_j L_j^i \) is the (maximal) set of slots that the algorithm had ever reserved (i.e., these reservations may have been canceled later) for last packets of frames of size at least \( i \). For each interval \( L_j^i \), let \( L_j^i = [a_j^i, b_j^i) \) and \( |L_j^i| = b_j^i - a_j^i \).

Let \( t_0 \) be the time when OPT delivered \( f \)'s last packet. Then \( f \) is charged to the \( L_j^k \) where \( j \) is minimum such that \( t_0 < b_j^k \), i.e., to the \( L_j^k \) whose right end is the first one after \( t_0 \). (Note that we are not guaranteed that \( t_0 \in L_j^k \) since OPT might deliver the last packet before \( \tau_{k_j}(f) + \Delta_f \).) Next, for each \( L_j^i \), we distribute the charge it receives evenly between all the frames of size at least \( i \) that ever made reservation for their last packets within \( L_j^i \). Denote the set of frames charged to \( L_j^i \) by \( F_j^i \), and let \( f_0 = \arg \max_{g \in F_j^i} \Delta_g \). Then for any \( g \in F_j^i \), the following hold: \( D_g \leq b_j^i \), \( t_{k_g}(g) \geq a_j^i - 2\Delta_{f_0} \), and \( |L_j^i| \geq s_{f_0} \). Thus \( |F_j^i|/|L_j^i| \leq (s_{f_0} + 2\Delta_{f_0})/s_{f_0} \leq 1 + 2c \).

To summarize, for each \( A_i \), there is a corresponding set \( B_i \) of frames of size at least \( i/2 \) that made reservations for last packets in the union of intervals allowed for the last packets of frames in \( A_i \) such that \(|A_i| \leq (1 + 2c)|B_i|\). We charge \( \bigcup_i A_i \) to \( \bigcup_i B_i \). Despite different frame sizes, the charging ratio is at most \( 2(1 + 2c) \), as

\[
\sum_{i=1}^{k_{\text{max}}} i|A_i \setminus A_{i+1}| = \sum_{i=1}^{k_{\text{max}}} i(|A_i| - |A_{i+1}|) = \sum_{i=1}^{k_{\text{max}}} |A_i| \leq \sum_{i=1}^{k_{\text{max}}} (1 + 2c)|B_i| = (1 + 2c) \sum_{i=1}^{k_{\text{max}}} i(|B_i| - |B_{i+1}|)(1 + 2c) \sum_{i=1}^{k_{\text{max}}} i(|B_i \setminus B_{i+1}|) .
\]

We now bound the total charge that the last frame \( f_C \) of a chain \( C \) can receive. Each frame \( f \) belonging to the chain may receive a charge of the first type (a self-charge) of value \( k_f \) and a charge of the second type (from frames rejected due to lack of slots for their last packet) of value at most \( 2(1 + 2c)k_f \).
For each $f$ in $C$, these are relayed to $f_C$. As each frame in $C$ is at least twice as large as its predecessor (the one it preempted), the total charge of the first two types relayed to $f_C$ is at most $2(5 + 2c)k_f$.

It remains to do similar calculations for the charges of the last type, namely frames that are rejected due to too many active frames. These are slightly different, because now instead of summing the sizes of all frames in a chain, we need to determine to how many slots a chain might grant credit. I.e., we need to account for gaps between successive frames of the chain, which could be as large as $\Delta_{\text{max}} + s_{\text{max}}$, and the extra credit that is granted past the end of a chain.

Each frame $f$ that belongs to a chain $C$ may provide credit of $\frac{2}{d_{\text{min}}}$ per slot for up to $(k_f - 1)d_f + \Delta_f + s_f \leq (k_f - 1)d_{\text{max}} + \Delta_{\text{max}} + s_{\text{max}}$ time slots, plus additional $2(k_{f_C} - 1)d_{\text{max}} + \Delta_{\text{max}} + s_{\text{max}}$ slots in case of $f_C$, and $d_{\text{min}} - 1$ more slots if $f_C$ is type 1, due to $f_C$’s remaining longer in $\text{Act}_1$ — we call this last term spare type 1 credit and ignore it for the time being. As each frame in $C$ is at least twice as large as the one it preempted, the total credit provided by the chain $C$ of length $i_C$ is at most
\[
\frac{2}{d_{\text{min}}} \left( 4k_{f_C}d_{\text{max}} + (i_C + 1)(\Delta_{\text{max}} + s_{\text{max}} - d_{\text{max}}) \right)
= 8\delta k_{f_C} + \frac{2}{d_{\text{min}}}(i_C + 1)(\Delta_{\text{max}} + s_{\text{max}} - d_{\text{max}})
\leq 2\delta (4k_{f_C} + \alpha(i_C + 1)) ,
\]

since $\Delta_{\text{max}} + s_{\text{max}} - d_{\text{max}} \leq \alpha d_{\text{max}} = \alpha \delta d_{\text{min}}$. We can now justify why the spare type 1 credit can be ignored: the term $4k_{f_C}d_{\text{max}}$ in the above bound is an (over-)estimation of $k_{f_C}(2 + 1 + \frac{1}{2} \ldots)$, which corresponds to sum of sizes of frames in $C$. However, all frames have integer sizes, and thus their total size if at most $4k_{f_C} - 1$. Thus, we are overestimating the credit by at least $\frac{2}{d_{\text{min}}}d_{\text{max}}$, which is larger than the unaccounted for spare type 1 credit.

Overall, the total charge to $f_C$ is thus at most
\[
2(k_{f_C}(5 + 2c + 4\delta) + \alpha \delta(i_C + 1)) \leq 2(k_{f_C}(5 + 2c + 4\delta) + 2k_{f_C}\alpha\delta)
= 2k_{f_C}(5 + 2c + 4\delta + 2\alpha\delta) ,
\]
since $i_C \leq 1 + \lfloor \log_2 k_{f_C} \rfloor$ due to the sizes of successive frames in a chain, and finally since $\lfloor \log_2 k_{f_C} \rfloor + 2 \leq 2k_{f_C}$ for every positive integer $k_{f_C}$.

\[\square\]

4 Common size, different periods

In this section, we consider instances in which all frames have the same size $k$ and same value $v$ (w.l.o.g., $v = 1$), but each frame $f$ has a possibly different period $d_f$, focusing on the perfectly periodic instances. Surprisingly, we were unable to provide any impossibility result for this setting. Instead, we propose a $\Theta(1)$-competitive non-preemptive algorithm. We assume that each and every packet of a frame has a deadline that coincides with the deadline of the frame.
4.1 A Non-Preemptive Algorithm

As in Section 3, our algorithm consists of two subroutines. The first decides, for each newly arriving frame, whether to accept or reject it, and the second schedules for transmission packets of accepted frames. Unlike the algorithm in Section 3, however, accepted frames are never preempted. The algorithm classifies every frame as either completed, accepted, or rejected.

- **Frame Arrival:** When a new frame \( f \) arrives, the algorithm accepts it if and only if the set of all accepted frames together with \( f \) has a feasible schedule.
- **Packet Transmission:** The algorithm always transmits the packet with the earliest deadline from the set of all pending packets of accepted frames. Once all packets of a frame have been sent, the frame is marked “completed.”

Let us comment briefly on the feasibility test and the algorithm’s correctness (i.e., why the deadlines are met). The feasibility test considers packets rather than frames: the set of packets in question is that of all pending packets and those yet to arrive that belong either to an accepted frame or the frame \( f \) whose status is being decided. Note that by our assumption of perfectly periodic instances, the exact arrival time of all packets considered is known. Thus testing the feasibility of a set of packets (which are just unit-length jobs) can be done by running EDF on that set, since EDF produces a (single machine) feasible schedule if there is one. Similarly, our algorithm observes all deadlines because it produces an EDF schedule for a feasible set of packets.

Alternatively, the schedule for packets can be viewed as a bipartite matching of packets to time slots. Hence, one can test for feasibility with a new arriving frame \( f \) by using any dynamic matching algorithm that checks whether the current matching (schedule) can be augmented to match all packets of \( f \) as well. If so, the resulting schedule can then be reordered to become an EDF schedule.

The algorithm is non-preemptive. As only packets belonging to accepted frames are ever transmitted, the algorithm never “wastes” a slot. This, and the fact that all frames have the same size, allows for counting the number of transmitted packets instead of frames in the analysis.

We further note that the algorithm is “eager” in the sense that acceptance of an arriving frame is decided immediately. One can also consider a similar “lazy” algorithm that decides to either accept or reject a frame only when its first packet would be scheduled by EDF. At such point, if the set of accepted frames together with \( f \) is feasible, then \( f \) is accepted and the packet is transmitted. Otherwise, \( f \) is rejected and another EDF packet is chosen for inspection. Intuitively, the lazy algorithm should perform no worse than the eager one. However, we analyze the eager variant due to its immediate decisions. Moreover, in the next section we show that neither variant is 1-competitive.

4.2 Upper Bound for the Algorithm

We do not know of any impossibility result for perfectly periodic instances. However, we can show that neither variant of our algorithm is 1-competitive.
Theorem 5. On perfectly periodic instances with periods \(d\) and \(d/2\) such that \(k > 2(d+1)\), both variants of the algorithm have competitive ratios at most \(1 - 1/d\). Moreover, no non-preemptive work-conserving algorithm is 1-competitive.

4.3 Analysis of the Algorithm

For convenience, we extend the arrival time and deadline notation to packets: for a packet \(p\), these are denoted \(t(p)\) and \(D_p\) respectively; recall that a packet is assigned the deadline of its frame. To reason about intervals, we denote the left and the right endpoint of an interval \(I\) by \(l(I)\) and \(r(I)\) respectively. Moreover, for any family of intervals \(\mathcal{F}\), we let \(u(\mathcal{F}) = |\bigcup_{I \in \mathcal{F}} I|\) and \(s(\mathcal{F}) = \sum_{I \in \mathcal{F}} |I|\).

Analysis Outline To analyze the algorithm, we establish a charging scheme. As before, we charge a frame \(f\) delivered both by OPT and the algorithm to itself. Thus we can restrict our attention to frames delivered by OPT that the algorithm rejected. We observe in Lemma 3 that for every rejected frame \(f\), there is an interval \(I_f\) that covers \(f\), i.e., spans both its arrival time and deadline, such that the algorithm delivers a packet in roughly a constant fraction of \(I_f\)'s slots. We call such an \(I_f\) a busy interval.

Intuitively, this should yield a constant competitive ratio since we can count packets rather than frames as noted in Section 4.1. Specifically, every frame \(f\) delivered by OPT that is not covered by a busy interval is delivered by the algorithm as well. And in each busy interval \(I\), OPT can deliver at most \(|I|\) packets, which is proportional to the number of packets that the algorithm delivers in \(I\).

However, there are two issues. First, Lemma 3 states that the algorithm sends packets in \(|I|/2 - k\) slots of a busy interval \(I\), which means that we have a constant ratio on a packet basis only if \(I\) is sufficiently large. Fortunately, it follows from Lemma 3 that short busy intervals correspond to rejected frames of small periods, and we can deal with such frames separately.

Second, busy intervals may overlap, leading to overcounting the packets delivered by the algorithm (and OPT). Thus, we need a claim similar to Lemma 3 for the union of all busy intervals. We remedy this by showing that there is a subset of the busy intervals that covers every rejected frame, with an additional property that, when ordered by either endpoint, no three successive intervals in the subset intersect. Clearly, the number of packets that OPT sends in any busy interval is no larger than the total length of the intervals in the subset. Thus, if we charge these packets of OPT to those sent by the algorithm in either all odd-numbered or all even-numbered intervals from the subset, whichever maximizes the total length, we do not charge a single slot twice, as these intervals are disjoint, and we lose only a factor of 2 in the total length of the intervals.

We note that each rejected frame is covered a “busy” interval.

Lemma 3. If the algorithm rejects a frame \(f_0\) upon its arrival at time \(t_1(f_0)\), then there exists \(T \geq D_{f_0}\) such that in \([t_1(f_0), T]\), i.e., the interval of \(T - t_1(f_0)\) slots starting at \(t_1(f_0)\), the algorithm delivers strictly more than \((T-t_1(f_0))/2-k\) packets, each with a deadline no larger than \(T\), within \([t_1(f_0), T]\). Moreover, if
$d_{f_0} = 1$, then the algorithm delivers strictly more than $T - t_1(f_0) - k$ packets, each with a deadline no larger than $T$, within $[t_1(f_0), T)$.

It is an intriguing question whether the theorem can be strengthened: is it true that there exists a $T \geq D_{f_0}$ such that the algorithm delivers strictly more than $T - t_1(f_0) - k$ packets in the interval $[t_1(f_0), T)$?

Next, we construct a good family of busy intervals that underpins our analysis.

Again, one of the properties we guarantee is covering all rejected frames. Note that when we say that a family $F$ of intervals covers a frame, we mean that the frame is covered by $\bigcup I \in F$, rather than a particular $I \in F$.

**Lemma 4.** There exists a family $I_0$ of busy intervals of length at least $3k$ and a subset $I'_0 \subseteq I_0$ with the following properties.
1. Every rejected frame of period at least $3$ is covered by $\bigcup I_0$.
2. $u(I_0) \leq s(I_0) \leq 2 \cdot u(I_0)$.
3. $u(I'_0) = s(I'_0) \geq \frac{1}{2} \cdot s(I_0)$.

In particular, the last property implies that $I'_0$ is a family of disjoint intervals.

Together, Lemmas 3 and 4 imply the following.

**Theorem 6.** The algorithm is $\frac{1}{17}$-competitive on perfectly periodic instances.

## 5 Common period, unit value

In this section we consider instances in which all frames have the same period $d$ and unit value, but arbitrary sizes. Combining Lemma 1 with either of the algorithms from Sections 2.2 or 3 yields the following result.

**Corollary 1.** There is a $\Omega(1/\log k_{\max})$-competitive randomized algorithm for instances with common period and unit value.

We could not find a better algorithm. In fact, two natural algorithms, EDF and SRPT, cannot perform much better: we prove an $O(\log \log k_{\max}/\log k_{\max})$ upper bound on their competitive ratios. We do not provide any guarantees for either of them. One could expect SRPT to be $\Omega(1/\log k_{\max})$-competitive as it attains this ratio for single machine preemptive throughput maximization [6, 2], which corresponds exactly to our setting with $d = 1$ and arbitrary $s_f$ values.

However, we do not know if its analysis can be extended to our problem.

EDF and SRPT are defined as follows. At any given time $t$, we say that a frame $f$ with deadline $D_f$ is *feasible* if the number of remaining packets of $f$ (ones that were not yet transmitted, including those that did not arrive yet) is no more than $D_f - t$. Clearly, an infeasible frame cannot be delivered. At step $t$, both algorithms examine the set of all available packets of feasible frames, and transmits one chosen as follows. EDF chooses a packet of the frame with the earliest deadline. SRPT chooses a packet of the frame with the smallest number of remaining packets. Ties can be broken arbitrarily in both algorithms.

Since a frame’s deadline is roughly its arrival time plus $d$ times its size, these algorithms behave similarly. In particular, they share the following property: If
the algorithm starts transmitting packets of a frame whose deadline is $t_f$, then by time $t_f$ at least one frame is completed. However, ignoring long frames may not be the right choice, as the following theorem, which also stated a rather weak impossibility result for any algorithm, shows.

**Theorem 7.** The competitive ratio of any randomized algorithm on perfectly uniform instances is at most 0.75. Moreover, the competitive ratios of both EDF and SRPT on such instances are $O(\log \log k_{\max} / \log k_{\max})$.

### References

1. Baruch Awerbuch, Yair Bartal, Amos Fiat, and Adi Rosén. Competitive non-preemptive call control. In *Proc. of the 5th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 312–320, 1994.

2. Christoph Dürr, Lukasz Jeż, and Nguyen Kim Thang. Online scheduling of bounded length jobs to maximize throughput. *J. Scheduling*, 15(5):653–664, 2012. Also appeared in *Proc. of the 7th Workshop on Approx. and Online Algorithms (WAOA)*, pp. 116–127 (2009).

3. Yuval Emek, Magnús M. Halldórsson, Yishay Mansour, Boaz Patt-Shamir, Jaikumar Radhakrishnan, and Dror Rawitz. Online set packing. *SIAM J. Comput.*, 41(4):728–746, 2012. Also appeared in *Proc. of the 29th ACM Symp. on Principles of Distributed Comput.* (PODC), pp. 440–449 (2010).

4. Leah Epstein, Lukasz Jeż, Jiří Sgall, and Rob van Stee. Online Scheduling of Jobs with Fixed Start Times on Related Machines. In *Proc. of the 15th Int. Workshop on Approx. Algorithms for Comb. Optim. (APPROX)*, pages 134–145, 2012. Also to appear in Algorithmica: http://dx.doi.org/10.1007/s00453-014-9940-2.

5. Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000. Also appeared in *Proc. of the 36th Symp. on Foundations of Comp. Sci. (FOCS)*, pp. 214–221 (1995).

6. Bala Kalyanasundaram and Kirk Pruhs. Maximizing job completions online. *J. Algorithms*, 49(1):63–85, 2003. Also appeared in *Proc. of the 6th European Symp. on Algorithms (ESA)*, pp. 235–246 (1998).

7. Alexander Kesselman, Boaz Patt-Shamir, and Gabriel Scalosub. Competitive buffer management with packet dependencies. *Theor. Comput. Sci.*, 489-490:75–87, 2013. Also appeared in *23rd IEEE Int. Parallel and Distributed Processing Symp.* (IPDPS), pp. 1–12 (2009).

8. Yishay Mansour, Boaz Patt-Shamir, and Dror Rawitz. Overflow management with multipart packets. *Computer Networks*, 56(15):3456–3467, 2012. Also appeared in *Proc. of the 30th IEEE Int. Conf. on Computer Communications (INFOCOM)*, pp. 2606–2614 (2011).

9. Michael Markovitch and Gabriel Scalosub. Bounded delay scheduling with packet dependencies. In *Proc. of the IEEE INFOCOM Workshops*, pages 257–262, 2014.

10. Gabriel Scalosub, Peter Marbach, and Jörg Liebeherr. Buffer management for aggregated streaming data with packet dependencies. *IEEE Trans. Parallel Distrib. Syst.*, 24(3):439–449, 2013. Also appeared in *Proc. of the 29th IEEE Int. Conf. on Computer Communications (INFOCOM)*, pp. 241–245 (2010).

11. Gerhard J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theor. Comput. Sci.*, 130(1):5–16, 1994.