Better Online Buffer Management

Fei Li* Jay Sethuraman† Clifford Stein‡

Abstract

As the Internet becomes more mature, there is a realization that improving the performance of routers has the potential to substantially improve Internet performance in general. Currently, most routers forward packets in a First-In-First-Out (FIFO) order. However, the diversity of applications supported by modern IP-based networks has resulted in unpredictable packet flows, and heterogeneous network traffic. Thus, it is becoming more reasonable to consider differentiating between different types of packets, and perhaps to consider allowing packets to specify a deadline by which it must be processed. These issues have made buffer management at routers a critical issue in providing effective quality of service to the various applications that use the network.

In this paper, we study an online problem in which each packet is described by its discrete arrival time, nonnegative weight and discrete deadline; arriving packets are buffered for delivery and all packets have the same processing time. The packets arrive online, and our objective is to maximize the sum of weights of those packets that are sent by their deadlines. We describe an online deterministic algorithm with a competitive ratio of 1.854, improving the best previous known competitive ratio of 1.939 (Bartal et al. STACS 2004).

The algorithmic framework we use has several interesting features. First, we do not use a potential function. Instead, after each step we modify the adversary's buffer. Second, we introduce "dummy packets" to facilitate the decision making.

1 Introduction

As the Internet becomes more mature, there is a realization that improving the performance of routers has the potential to substantially improve Internet performance in general. Currently, most routers forward packets in a First-In-First-Out (FIFO) order. However, the diversity of applications supported by modern IP-based networks has resulted in unpredictable packet flows, and heterogeneous network traffic. Thus, it is becoming more reasonable to consider differentiating between different types of packets, and perhaps to consider allowing packets to specify a deadline by which it must be processed. These issues have made buffer management at routers a critical issue in providing effective quality of service to the various applications that use the network. Motivated by these considerations, Kesselman et al. [13] propose a model, called *buffer management with bounded delay*. In this model, packets arrive over time, and are buffered upon arrival. An arriving packet (w, d) has a *weight* w and a *deadline* d before which it must be transmitted. At most one packet can be sent in each (integer) time step. A packet with deadline d that is not sent before time d expires, and is dropped from the buffer. The objective is to maximize *weighted throughput*, defined as the total weight of the transmitted packets.

If the relevant characteristics — release date, weight, and deadline — of each packet are known ahead of time, an optimal schedule can be found efficiently, for instance, as a maximum weight matching problem on a convex bipartite graph. In most applications, however, we do not know this information ahead of time. Rather, packets arrive *online*, and we only learn about a packet and its associated characteristics when it actually arrives. An online algorithm is *k-competitive* if its weighted throughput on *any* instance is at least $1/k$ of the weighted throughput of an optimal offline algorithm on this instance. The smallest value of k for which an algorithm is k-competitive is called its *competitive ratio* [7]. If an algorithm decides which packet to process based only on the contents of its buffer, and independent of the packets that have already been processed, we call it *memoryless*.

1.1 Prior Work Since this online buffer management model was introduced in [13], many papers have considered this problem as well as several variants. Two natural restrictions on deadlines can be described in terms of a packet's *span*, defined as the difference between its deadline and release date. An input instance is called *s-bounded* if the span of any packet is at most s, and *s-uniform* if the span of any packet is exactly s. An input instance has *agreeable deadlines* (or is *simi-*

*Department of Computer Science, Columbia University. lifei@cs.columbia.edu

†Department of Industrial Engineering and Operations Research, Columbia University. Research partially supported by NSF Grant DMI-0093981 and by an IBM faculty award. jay@ieor.columbia.edu

‡Department of Industrial Engineering and Operations Research, Columbia University. Research partially supported by NSF Grant DMI-9970063. cliff@ieor.columbia.edu

larly ordered) if the deadlines of the packets (weakly) increase with their release dates. The agreeable deadline model generalizes both the s-uniform model and the 2-bounded model.

The best known lower bound on the competitive ratio of deterministic algorithms is $\phi \approx 1.618$ [12, 9, 2]; this lower-bound applies to 2-bounded instances, and hence also to instances with agreeable deadlines.

For an arbitrary deadline instance, a simple greedy algorithm that always schedules a maximum-weight packet in the buffer is 2-competitive [12, 13]. A generalization of the greedy algorithm, called EDF_α, always schedules the earliest packet with weight at least $1/\alpha$ ($\alpha \geq 1$) of the maximum-weight packet [6]. Although EDF_α improves the competitive ratio for s-bounded instances, the best competitive ratio of this family of algorithms is (asymptotically) 2 for the general case. To improve the competitive ratio, it is natural to consider alternating between the maximum-weight packet and an earliest-deadline packet with sufficiently large weight. As stated, this does not result in an improvement, but Chrobak et al. [8] discuss a clever modification that results in an algorithm with competitive ratio $64/33 \approx 1.939$. This algorithm is the first one with competitive ratio strictly below 2 for the general case.

For instances with agreeable deadlines (and hence also s-uniform instances), Li et al. [15] propose an (optimal) algorithm MG whose competitive ratio is ϕ. Unfortunately, this improved competitive ratio is achieved by exploiting the agreeable deadline assumption; on general instances, MG, like the greedy algorithm and EDF_α is also 2-competitive. (See the Appendix for such an instance.)

Finally, for 2-uniform instances, Chrobak et al. [8] find an algorithm that is 1.377-competitive and prove a matching lower bound. This algorithm uses information about the past, and so is not memoryless. In fact, a lower bound of $\sqrt{2}$ has been proved on the competitive ratio of memoryless algorithms for 2-uniform instances.

Randomized algorithms have also been given [6] with competitive ratios of $e/(e-1) \approx 1.582$, and 1.25 for 2-bounded instances. For 2-bounded instances, the lower bound is 1.25, while for the 2-uniform case it is 1.172.

There has also been work on models in which the FIFO discipline is enforced [16, 14, 5, 17, 10]. In the FIFO model, packets have weights, but no deadlines; and the buffer is finite. Some researchers also consider packet scheduling in multiple FIFO input queues connecting one output queue [3, 4, 1].

Englert and Westermann [11] independently gave a 1.828-competitive algorithm and a 1.893-competitive memoryless algorithm for the same buffer management problem that we consider in this paper. That work also appears in these proceedings. We compare our work and their work further in Section 5.

1.2 Our Contribution

We design an algorithm called DP (for *dummy packets*) whose competitive ratio on general instances is *at most* $3/\phi \approx 1.854$. In addition to improving the best known competitive ratio, the algorithm presented here and the analysis have several novel features. First, the algorithm generates "dummy" packets whose status encodes relevant information about the past behavior of the algorithm. Although these dummy packets cannot be transmitted by the algorithm, they influence the choice the algorithm makes. Second, the analysis does not rely on a potential function approach explicitly. Instead, it relies on modifying the algorithm's buffer judiciously and on assigning an appropriate credit to the adversary to account for these modifications. This is similar to, but more complicated than, the approach used in our earlier paper [15] for a special case of the problem.

2 Motivation for Our Algorithm DP

Before describing our algorithm in detail, we discuss some of the previous algorithms, and give insight into their limitations. We will the use that insight to explain the new features of our algorithm DP.

Recall that (w, d) denotes a packet with weight w and deadline d. We first note why the greedy algorithm, which always sends a heaviest packet regardless of the deadline, is at best 2-competitive. This is because when presented with packets $p = (1-\epsilon, 1)$ and $q = (1, 2)$, it picks q (and p expires at the end of this time-slot) whereas an optimal algorithm would send both. A natural strategy then is to pick the earliest packet whose weight is sufficiently large when compared with the heaviest packet; that is, for some $\alpha \geq 1$, send the earliest packet in the buffer whose weight is at least w_h/α. (This is the algorithm EDF_α; note that EDF_1 is a greedy algorithm.) The following example shows that EDF_α is also (at best) 2-competitive.

Example 1. At each $t = 0, 1, 2, \ldots, n-1$, packets $p_t = (1-\epsilon, t+1)$ and $q_t = (1, n+t+1)$ arrive. In addition, a packet $P = (\alpha, 2n+1)$ arrives at time zero. Thus at time zero the buffer contains p_0, q_0, and P. The algorithm EDF_α will, in each of the first n time slots, send the q packet, drop the p packet, and retain P. At time n, it sends the only packet, P, in the buffer. An optimal algorithm will deliver all the packets. It is easy to verify that the competitive ratio of EDF_α approaches 2 as n increases.

The main difficulty with the greedy algorithm is that when the algorithm sends a packet q, there is always the possibility that a packet p with $w_p = w_q - \epsilon$, and $d_p < d_q$ exists; if p cannot be sent later by the algorithm, but the adversary can send both p and q, the competitive ratio becomes essentially 2. The same difficulty persists with EDF_α, except that it cannot be done with 2 packets alone. In the above example, the packet P forces EDF_α to favor the q packet in each time step over the p packet, and repeating this a large number of times makes the larger weight of P negligible in comparison to the total weight of the p packets.

In designing an algorithm with competitive ratio better than 2, we have to address the following tension. On the one hand, the algorithm must send a packet whose weight is sufficiently large compared to the heaviest packet. On the other hand, when it sends a packet, it should consider the possibility that the adversary sends an *earlier* packet in the buffer with *slightly* smaller weight, especially when the algorithm does not get a chance to send this packet in the future. Understanding this latter possibility is critical in deriving an improved algorithm.

We show in [15] that for instances with agreeable deadlines, whenever the latter condition occurs, the "earlier" packet with slightly smaller weight must be an *earliest-deadline* packet. For such instances, it is natural to modify EDF_α as follows: instead of sending the earliest packet with weight at least w_h/α, we look for the earliest packet p in the buffer whose weight is at least $1/\alpha$ of the heaviest packet and whose weight is at least α times the earliest deadline packet. This algorithm is called MG_α and its competitive ratio is $\max\{\alpha,\ 1 + 1/\alpha\}$; choosing $\alpha = \phi$, we get an optimal deterministic competitive ratio of ϕ.

For instances that do not satisfy the agreeable deadline assumption, the latter case may occur even if the "earlier" packet is not an earliest deadline packet, so MG may not achieve this competitive ratio in general. Indeed, the appendix includes a family of examples on which MG's competitive ratio approaches 2.

Motivated by all of these observations, we describe a new algorithm that is able to achieve an improved competitive ratio. This algorithm is inspired by both MG and EDF_α. Like MG, we first find a maximum-weight subset, M_t, of packets that can be scheduled on-time at each time period t, and we identify two special packets from M_t: the "earliest-deadline" packet e and the "heaviest" packet h. One of the difficulties with EDF_α is that a *single* heavy packet P could influence the choice of the algorithm in each of the first n steps. To overcome this, we associate an additional *status* bit with each packet, and whenever we send a packet f other than e or h, we set h's status bit to 1, and reduce its weight to w_h/α; this reduced weight is used when identifying the "heaviest" packet in the future. If, in a future time step, this (reduced weight) packet is the heaviest packet, the algorithm simply sends it, preventing the scenario observed with EDF_α on Example 1. Moreover, a "dummy" packet h' is generated whose weight is w_h/α and whose deadline is d_f, the deadline of the packet just sent by the algorithm. The status of the dummy packet encodes useful information about the history of the packets sent by the algorithm as well as the input sequence. For instance, if it is optimal for the adversary *not* to send f, but to send a packet after f (for instance, the h packet itself), one may expect many packets better than f to arrive in the near future; if, on the other hand, it is optimal for the adversary to send a packet earlier than f now, one may expect future packets to be of lower value. When deciding which packet to send now, the algorithm has no way of knowing which of these cases will occur. However, the status of the dummy packet captures this information implicitly: in the former case, we expect the dummy packet to leave M_t soon, whereas in the latter case we expect the dummy packet to remain in M_t until its deadline. Thus it makes sense to let these packets influence the algorithm's choice. This argument is not rigorous, but this intuition drives the design of the algorithm, described next. An example further illustrating the use of dummy packets is described in the Appendix.

3 Algorithm DP

Associated with each packet p is a non-negative integer release date r_p, a positive integer deadline $d_p > r_p$, and a non-negative real weight w_p, which represents the value gained by sending p at some time in the interval $[r_p, d_p)$. We call a sequence *feasible* if all packets can be scheduled by their deadlines. We associate two additional pieces of data with each packet: a *virtual value* and a *status bit*. We also introduce *dummy packets*.

Status bits and virtual values. Each packet p in the buffer has a status bit $p.c$, which is set to zero upon p's arrival, but may be modified (and re-modified) during the course of the algorithm. The virtual value of a packet p in the buffer depends on its status bit, and is defined as

$$v_p := \begin{cases} w_p, & \text{if } p.c = 0, \\ w_p/\alpha, & \text{if } p.c = 1, \quad \alpha > 1. \end{cases}$$

Notice that v-value of p depends on both w_p and $p.c$, and that $p.c$ may be modified over time.

Dummy packets. In certain cases, the algorithm generates *dummy* packets. The dummy packets are not eligible to be sent by the algorithm, but play a role in deciding which packets the algorithm does send. The status bit of any dummy packet is always 0 until it is dropped from the buffer. Each dummy packet (with status bit of 0) in the buffer is always "paired" with a real packet with status bit of 1 in the buffer. Note also that real packets are never dropped from the buffer before they expire, but dummy packets may be.

3.1 The Algorithm We describe a family of algorithms, DP_α, parametrized by $\alpha \geq 1$. The analysis in the next section shows the competitive ratio β to be at most $\max\{\alpha, 1 + 1/\alpha, 3/\alpha, 3/(1 + 1/\alpha)\}$. Setting $\alpha = \phi$ results in an upper bound on the competitive ratio of $3/\phi$. The algorithm is described in two parts. At any time t, we first identify the set M_t, the set of matched packets at time t. From this set, two special packets — e and h — are identified, which influence the packet delivery part of the algorithm.

3.1.1 Identifying the matched packets for step t. Let M_t be a maximum-weight subset (using the w values) of (real and dummy) packets that can be sent successfully in steps $t, t + 1, \dots$, assuming no future arrivals. M_t can be determined as a maximum weight matching of the natural convex bipartite graph associated with the scheduling problem. The convexity is an immediate consequence of the fact that in the matching there is never a reason to have two edges that "cross."

Let p' be a dummy packet and p its associated real packet. Then the algorithm will always set $d_{p'}$ to a value less than d_p (the exact setting is described in the next subsection). The algorithm will always set the virtual value of the dummy packet equal to the virtual value of the real packet, i.e.

$$w_{p'} = v_{p'} = v_p = w_p/\alpha < w_p,$$

where the last inequality holds because $\alpha > 1$. Furthermore, we may assume without loss of generality that M_t contains p' only if it contains p. This is because any maximum-weight matching that contains p' but not p can be improved by replacing p' with p. Also, whenever $p \in M_t$ and $p' \notin M_t$, we drop p' from the buffer altogether and reset $p.c$ to zero (from one).

The packets in M_t are called *matched* packets for this step. They are placed in the buffer in *canonical order*: non-decreasing deadline order, with ties broken in non-increasing weight order. All *valid*, unmatched *real* packets are placed in an arbitrary order after all the matched packets; unmatched *dummy* packets are dropped (as mentioned earlier). In the rest of the paper, whenever we use the term "earlier in the buffer" we mean "earlier in M_t."

3.1.2 Packet delivery for step t.

2.0 **(Identifying h and e packets)** Let h denote the packet with the largest v-value among all *real* packets in M_t (ties are broken in favor of the earliest deadline packet); and and let e denote the earliest deadline (*real* or *dummy*) packet in M_t (ties are broken in favor of the heaviest packet).

2.1 If $h.c = 0$, let $\hat{f}$ be the earliest packet in M_t satisfying $v_{\hat{f}} \geq v_h/\alpha = w_h/\alpha$. (Clearly $\hat{f}$ exists as h itself is a candidate for $\hat{f}$.) If $\hat{f}$ is a real packet, $\hat{f}$ is sent, and its corresponding dummy packet (if any) is dropped from the buffer. If $\hat{f}$ is a dummy packet, its corresponding real packet is sent and $\hat{f}$ is dropped from the buffer. Let f denote the (real) packet sent.

If $f \neq e, h$ and $d_f < d_h$, set $h.c = 1$, and generate a dummy packet h' with $w_{h'} = w_h/\alpha$, $d_{h'} = d_f$, and $h'.c = 0$. (Note: $d_h > d_{h'} > t$.)

2.2 If $h.c = 1$, send h and drop the associated dummy packet h' from the buffer.

We use f to denote the packet sent by the algorithm, and observe the following.

- $e.c$ must be 0, otherwise, e', the dummy packet associated with e, is before e in the buffer.
- If $h.c = 0$, then
  (a) If f is the e packet or the h packet, $f.c = 0$.
  (b) If $f \neq e, h$ and $d_f \geq d_h$, then f must be the real packet associated with the dummy packet $\hat{f}$ that was "selected" by the algorithm. (Otherwise the algorithm would have sent the h packet or a packet that appears before the h packet.) Note that $\hat{f}$ is dropped from the buffer in this step.

4 Analysis of DP

This section is devoted to proving the following result.

THEOREM 4.1. *The competitive ratio of DP_α with $\alpha = \phi$ is at most $3/\phi \approx 1.854$.*

Let $\mathcal{O}$ be the sequence of packets sent by an optimal offline algorithm (= adversary). To analyze the algorithm, we maintain, at each time step t, a sequence of packets S_t such that it is possible to deliver all the packets in S_t by their deadlines. The sequence S_t will be constructed inductively, starting with $S_0 = \mathcal{O}$. Given S_t, the sequence S_{t+1} is constructed based on the the first packet in S_t, the packet sent by the algorithm in step t, as well as the following input sequence. Before enumerating the various cases, we state briefly how the proof proceeds. Let W_t be the weight of the packet sent by the algorithm in step t, and let

$$V_t \;=\; \sum_{p \in S_t} v_p - \sum_{p \in S_{t+1}} v_p.$$

Observe that $\sum_t V_t = \sum_{p \in S_0} v_p = \sum_{p \in \mathcal{O}} w_p$ is the total weight of the packets sent by the optimal offline algorithm, and $\sum_t W_t$ is the total weight of the packets sent by the algorithm DP. To prove the main result, it is sufficient to show that $V_t \le \beta \cdot W_t$, for each step t. Our proof does *almost* this, but not quite: we shall show that the time steps can be partitioned into groups $T_1, T_2, \ldots$, such that for any group T_k,

$$\sum_{i \in T_k} V_i \le \beta \cdot \sum_{i \in T_k} W_i.$$

Each of the groups we construct in the proof will involve either a single time-step or will involve two time-steps.

We consider cases defined 3 factors: the packet sent by the algorithm in step t, the first packet in S_t, and the subsequent input sequence. We make two important points about the sequence S_t that will be useful. At the beginning of step t, we are allowed to modify S_t using the following operations:

(A) We are free to reorder S_t as long as all the packets in the reordered sequence can be sent by their deadlines.

(B) We are free to replace a packet $p \in S_t$ by another packet $q \notin S_t$ as long as $v_q \ge v_p$ and the resulting sequence is still feasible; in particular, if p is the earliest deadline packet in S_t, $p.c = 1$, and if S_t contains p but not p', we can always replace p by p'.

Remark 4.1. *Note that operation (B) results in a larger V_t than what is necessary; as a result, $\sum_t V_t \ge \sum_{p \in \mathcal{O}} w_p$. We will use this operation frequently in what follows, so it will be convenient to let $S_t - p + q$ denote the sequence obtained by replacing $p \in S_t$ with $q \notin S_t$.*

Fix a time step t. A packet $p \in S_t$ is *available* if $r_p \le t$. We state and prove two useful lemmas, one about each of the operations discussed earlier. The first lemma is a standard result about the EDD algorithm.

Lemma 4.1. *Among all available packets from the sequence S_t, let i denote the one with the earliest deadline (ties broken arbitrarily). Then, there is a feasible reordering of S_t in which i appears as the first packet.*

Lemma 4.2. *Given a sequence S_t, let i denote the packet with the earliest deadline among all available packets (ties broken arbitrarily). Then, we may assume without loss of generality that $i \in M_t$ and $i.c = 0$.*

Proof. Suppose $i \notin M_t$. Then for some $k \ge d_i$, the buffer must contain k packets, all due by time $t + k$, and moreover, each of these matched packets p must satisfy $w_p \ge w_i$. (Otherwise, i must be matched.) As $i \in S_t$ and S_t is feasible, it follows that not all of these k packets belong to S_t. Let p be such a packet. Consider the sequence $\bar{S}_t := S_t - i + p$. By Lemma 4.1, we may assume that i appears first in S_t. Since $d_p \ge t + 1$, $\bar{S}$ is clearly feasible. To show the validity of the replacement, we need to show that $v_p \ge v_i$. This follows from $w_p \ge w_i$ if $p.c = 0$: in that case, $v_p = w_p$ whereas the largest v_i can be w_i. Suppose, however, $p.c = 1$. Then we know that $p' \in M_t$, and $d_{p'} \le d_p$, so p' is among the "bottleneck" set of k packets. In particular, this implies $v_{p'} \ge v_i$. Noting that $v_p = v_{p'}$, we conclude that $v_p \ge v_i$ in all cases. We now have a new sequence $\bar{S}_t$ that has one more packet in common with M_t. Let $\bar{i}$ denote the packet with the earliest deadline among all available packets in $\bar{S}_t$. If $\bar{i} \notin M_t$, we apply the same argument with $\bar{S}_t$ and $\bar{i}$, assuming the role of S_t and i respectively. As the number of packets in common with M_t increases at each step, eventually we find a sequence, S_t, whose earliest-deadline available packet $i \in M_t$. To prove the second statement, if $i.c = 1$ then i' (the dummy packet associated with i) must also be matched. Since $d_{i'} < d_i$, this implies $i' \notin S_t$. The sequence $S_t - i + i'$ satisfies the properties claimed in the lemma. ■

Recall that the algorithm *always* sends the h packet when $h.c = 1$, but makes a possibly different choice when $h.c = 0$. We examine these cases in turn. In what follows i denotes the first packet in the (possibly modified and reordered) sequence S_t. By Lemma 4.2, $i \in M_t$ and $i.c = 0$.

Case 1: $h.c = 1$. There must be an associated dummy packet h' that is also matched and appears earlier in the buffer. The algorithm sends h, and drops h' from the buffer. By the definition of the h packet, for any packet $p \in M_t$, $v_p \le v_h = w_h/\alpha$. Let i be the first packet in S_t. Note that i may be real or dummy, but i is matched. In either case, $v_i \le v_h$. Define S_{t+1} as $S_t - i - h' - h$. The feasibility of S_t implies the feasibility of S_{t+1}. Now,

$$\begin{aligned} V_t &= v_i + v_{h'} + v_h \le v_{h'} + 2 \cdot v_h \\ &= w_h/\alpha + 2 \cdot w_h/\alpha = (3/\alpha) \cdot w_h \\ &\le \beta \cdot w_h = \beta \cdot W_t. \end{aligned}$$

Case 2: $h.c = 0$. We make some general observations that will be useful later on. Note that $v_h = w_h$

as $h.c = 0$. Recall that i denotes the first packet in S_t. As $i \in M_t$, we may assume that i appears no later than h in the buffer. Otherwise, $i \neq h$ and $d_i > d_h$ (M_t is arranged in canonical order), and we can swap i and h in S_t; the new S_t remains feasible and its first packet is the h-packet, satisfying the property claimed. Thus, in the rest of this section we assume that i appears no later than h in the buffer.

Let f be the packet sent by the algorithm. In this case, f is the e packet, or the h packet, or neither. The last case ($f \neq h, e$) is further subdivided, depending on whether $d_f < d_h$ or not. We now examine each of these cases.

Case $2a$**:** $f = e$**.** If $e \in S_t$, we claim that we may consider e to be the first packet in S_t. Otherwise, assume $i \neq e$ is the first element; since $i, e \in M_t$, $d_i \geq d_e$ by definition, so we can reorder S_t by swapping i and e so that e is the first element of S_t. As $e.c = 0$, note that $v_e = w_e$.

Define S_{t+1} by omitting e from S_t. The feasibility of S_{t+1} follows from the feasibility of S_t, and

$$V_t \;=\; W_t \;=\; w_e \;<\; \beta \cdot W_t.$$

If $e \notin S_t$, let $S_{t+1} = S_t \setminus i$. Clearly, S_{t+1} is feasible; moreover

$$V_t = v_i \leq v_h \leq \alpha \cdot v_e = \alpha \cdot w_e < \beta \cdot w_e = \beta \cdot W_t.$$

Case $2b$**:** $f = h$**.** In this case, we know that f is a real packet with $f.c = 0$ (because $h.c$ is assumed to be zero). Moreover, any (real or dummy) packet $p \in M_t$ that appears in the buffer before h has $v_p < v_h/\alpha = w_h/\alpha$.

Let i be the first packet in S_t. Recall that i appears no later than h in M_t (and hence in the buffer). Define S_{t+1} by removing i and h from S_t. (If i and h are identical, or if $h \notin S_t$, only one packet is removed.) The feasibility of S_{t+1} is evident. Also,

$$V_t \;\leq\; v_i + v_h \;<\; w_h/\alpha + w_h \;\leq\; \beta \cdot w_h \;=\; \beta \cdot W_t.$$

Case $2c$**:** $f \neq e, h$**,** $d_f \geq d_h$**.** If $f \neq h$, and if $d_f \geq d_h$, it follows that a dummy packet $\hat{f}$ was "selected" by the algorithm (so $\hat{f} \in M_t$, see Case 2.2(b.) in the description of the algorithm), and f is the real packet associated with that dummy packet $\hat{f}$. In particular, $f.c = 1$ and so $f \neq i$ as $i.c = 0$.

As $\hat{f}$ was "selected" by the algorithm, $v_{\hat{f}} \geq v_h/\alpha$. Since $\hat{f}$ is associated with f, $v_{\hat{f}} = w_{\hat{f}} = w_f/\alpha$; also, $h.c = 0$ implies $v_h = w_h$. $w_f/\alpha \geq w_h/\alpha$, which implies $w_f \geq w_h$. Recall also that $i, \hat{f}, f$ are all in M_t, so all of these packets are matched packets in the buffer. Finally, if $\hat{f} \in S_t$ and $d_{\hat{f}} \leq d_i$ we can swap i and $\hat{f}$ in S_t, noting that $\hat{f}.c = 0$ as required; therefore we may assume that the first packet in S_t (denoted i) is a packet that appears no later than $\hat{f}$ in the buffer whenever $\hat{f} \in S_t$.

Suppose i appears no later than $\hat{f}$ in the buffer (this also covers the possibility $i = \hat{f}$). Then $v_i < v_h/\alpha = w_h/\alpha$. Define S_{t+1} by removing i, $\hat{f}$, and f from S_t; if $i = \hat{f}$ or if some of these packets are not in S_t, fewer packets will be removed. Clearly S_{t+1} is feasible because S_t is. For this step,

$$\begin{aligned} V_t &\leq v_i + v_{f'} + v_f \leq w_h/\alpha + w_f/\alpha + w_f/\alpha \\ &\leq (3/\alpha) \cdot w_f \leq \beta \cdot w_f = \beta \cdot W_t. \end{aligned}$$

Suppose i appears after $\hat{f}$ in the buffer. By the earlier discussion, $\hat{f} \notin S_t$. Define S_{t+1} by removing i and f from S_t (if $f \notin S_t$, only i is removed). As before, S_{t+1} is feasible because S_t is, and

$$\begin{aligned} V_t &\leq v_i + v_f = w_i + v_f \leq w_h + v_f \\ &\leq w_f + v_f = w_f + w_f/\alpha \leq (1 + 1/\alpha) \cdot w_f \\ &\leq \beta \cdot w_f = \beta \cdot W_t. \end{aligned}$$

Case $2d$**:** $f \neq e, h$**,** $d_f < d_h$**.** This case is the only one in which the algorithm generates a dummy packet. Note that any packet p that appears before f in the buffer satisfies $v_p < v_h/\alpha = w_h/\alpha$; and $v_f \geq v_h/\alpha = w_h/\alpha$. When $f.c = 0$, this implies $w_f \geq w_h/\alpha$, whereas when $f.c = 1$ this implies $w_f \geq w_h$. The algorithm generates a dummy packet h' with weight w_h/α and deadline d_f, and sets $h.c$ to 1. Since h' is a dummy packet, $h'.c = 0$ as long as h' is in the buffer. Recall that at the first time-step in which h' is unmatched, it is dropped from the buffer and $h.c$ is reset to zero. Therefore if h' is in the buffer, it implies h' is matched from the moment it was generated until the current time-step.

Case $2d(i)$**:** $f.c = 0$**.** Let i be the first packet in S_t. Then either $i = f$, i appears after f in the buffer, or i appears before f in the buffer. We analyze each of these possibilities after noting the following useful fact: If $f \in S_t$ and $d_f \leq d_i$ we can swap i and f in S_t, noting that $f.c = 0$ as required; therefore we may assume that the first packet in S_t (denoted i) is a packet that appears no later than f in the buffer whenever $f \in S_t$.

When $i = f$, define $S_{t+1} = S_t - f$. The feasibility of S_{t+1} is immediate. For this step, $V_t = w_f + (1 - 1/\alpha) \cdot w_h$, where the second term accounts for the change in the status bit of h in this step: $h.c$ is set to 1, resulting in v_h decreasing from w_h to w_h/α. Note that

$$\begin{aligned} V_t &= w_f + (1 - 1/\alpha) \cdot w_h \leq w_f + (1 - 1/\alpha) \cdot (\alpha \cdot w_f) \\ &= \alpha \cdot w_f \leq \beta \cdot w_f = \beta \cdot W_t. \end{aligned}$$

Now, suppose i appears after f in the buffer. By our earlier discussion, $f \notin S_t$. If $i = h$, let $S_{t+1} = S_t \setminus h$.

The feasibility of S_{t+1} is immediate. And

$$V_t = w_h = v_h \le \alpha \cdot v_f = \alpha \cdot w_f \le \beta \cdot w_f = \beta \cdot W_t.$$

Suppose $i \neq h$, $h \in S_t$ and assume $S_t = \{p_t = i,\ p_{t+1}, \ldots, p_{k-1},\ p_k = h,\ p_{k+1}, \ldots\}$. (If $h \notin S_t$, we can always use h to replace i in S_t since $h.c = i.c = 0$ and $v_h \geq v_i$. Then, we turn to the case in which $i = h$.) We now have 2 subcases, depending on the value of d_i.

Subcase 1: $d_i \geq k$: We reorder S_t by swapping i and h, that is, we let $S_t = \{p_t = h,\ p_{t+1}, \ldots, p_{k-1},\ p_k = i,\ p_{k+1}, \ldots\}$. Clearly, S_t is feasible if $d_i \geq k$. Thus, $S_{t+1} = S_t \setminus h$, and

$$V_t = v_h \le \alpha \cdot v_f = \alpha \cdot w_f \le \beta \cdot w_f = \beta \cdot W_t.$$

Subcase 2: $d_i < k$. Recall that in this time step, the algorithm changed the status bit of h from 0 to 1. Observe also that h will stay in the buffer until either the algorithm sends it or it expires. In the former case $h.c$ may be 0 or 1, but in the latter case $h.c$ will be 0. We further subdivide the analysis into two cases depending on whether $h.c$ is reset to 0 or not.

Subcase 2a: $h.c$ is reset to 0 by DP. Let $j > t$ be the earliest time step at which $h.c = 0$. It is clear that $j \leq d_i$ because $d_{h'} = d_f < d_i$ and $h.c$ is reset to 0 when h' expires. Observe also that $d_h \geq k$ and $w_h \geq w_i$. Therefore we let $S_{t+1} = S_t - i$, but we also modify the release date of h in S_{t+1} to j. That is, this packet h is "unavailable" until time step j. (Thus, it is not possible to "drop" this packet from $S_{t+1}, S_{t+2}, \ldots, S_{j-1}$ based on the operations of Lemma 4.2.) Therefore, $S_{t+1} = \{p_{t+1}, \ldots, p_{k-1},\ p_k = \hat{h},\ p_{k+1}, \ldots\}$, where $p_k = \hat{h}$ is the same as h, but with its release date modified to j. Thus,

$$V_t = v_i \le v_h \le \alpha \cdot v_f = \alpha \cdot w_f \le \beta \cdot w_f = \beta \cdot W_t.$$

Subcase 2b: $h.c$ is not reset to 0 by DP. In this case, it must be sent by the algorithm with $h.c = 1$. Suppose DP sends h in time step j. As $h.c = 1$, h' must be matched in steps $t+1, \ldots, j$. Let p_j be the packet with the earliest deadline among all available packets in S_j. (That is, p_j is the "i-packet" in step j.) Since $p_j.c = 0$ and $h.c = 1$, $p_j \neq h$. (Note that p_j may be h' as Lemma 4.2 may have been used earlier to replace h with h'.)

Let $\bar{h}$ be the heaviest packet (i.e. the "h-packet") in step j. As the algorithm sends h in step j, either (i.) $h = \bar{h}$, or (ii.) $h \neq \bar{h}$, but $\bar{h}.c = 0$. We will define S_{t+1} as $S_t - i$, and S_{j+1} as $S_j \setminus \{j, h\}$. We shall group these 2 time-steps together and relate $V_t + V_j$ to $W_t + W_j$. To that end, we first show $w_{p_j} \leq w_h$.

If $h = \bar{h}$, then h is the heaviest packet in the buffer at time j. So $v_{p_j} \leq v_h$. As $p_j.c = 0$, $w_{p_j} = v_{p_j} \leq v_h \leq w_h$, as required. If $h \neq \bar{h}$, then $\bar{h}.c = 0$. As DP sends h, we have $v_h \geq v_{\bar{h}}/\alpha$. But by the definition of $\bar{h}$, $v_{p_j} \leq v_{\bar{h}}$, so

$$w_{p_j} = v_{p_j} \le v_{\bar{h}} \le \alpha \cdot v_h = w_h.$$

It is clear that $W_t = w_f$ and $W_j = w_h$. Also, $V_t = w_i + (1 - 1/\alpha) \cdot w_h$, and $V_j = v_{p_j} + v_h = w_{p_j} + (1/\alpha) \cdot w_h$. Therefore, we have

$$\begin{aligned} V_t + V_j &= w_i + w_h + w_{p_j} \ \le\ 3 \cdot w_h \\ &\le \beta \cdot (1 + 1/\alpha) \cdot w_h = \beta \cdot (w_h/\alpha + w_h) \\ &\le \beta \cdot (w_f + w_h) = \beta \cdot (W_t + W_j). \end{aligned}$$

In the above chain of expressions, we used $w_i \leq w_h$, $(1 + 1/\alpha) \cdot \beta \geq 3$, and $w_f \geq w_h/\alpha$. The first follows because h was the "h-packet" in step t and i was in the buffer then; the second from the definition of β; and the third from the definition of f, the packet DP sent at time t. (Note that DP sends a packet with status bit 1 in time step j, so step j cannot fall into this case again. In other words, we are not led to a longer chain of steps.)

Finally, suppose i appears before f in the buffer. Note that $v_i < v_f$ because i was not chosen by the algorithm but f was; since $i.c = f.c = 0$, this implies $w_i < w_f$ as well. Therefore, we may assume that $f \in S_t$, otherwise we may replace i by f and appeal to the case "$i = f$" discussed earlier. To get S_{t+1} from S_t, we remove i, and replace $f \in S_t$ by the newly generated dummy packet h'. Since h' and f have the same deadline, the feasibility of S_t implies the feasibility of S_{t+1}. In this step,

$$\begin{aligned} V_t &= v_i + (v_f - v_{h'}) + (w_h - v_h) \\ &= w_i + (w_f - w_h/\alpha) + (1 - 1/\alpha) \cdot w_h \\ &\le w_i + w_f + (1 - 2/\alpha) \cdot w_h \\ &\le w_h/\alpha + w_f + (1 - 2/\alpha) \cdot w_h \\ &\le w_f + w_f + (1 - 2/\alpha) \cdot (\alpha \cdot w_f) \\ &= \alpha \cdot w_f \ \le \beta \cdot w_f \ =\ \beta \cdot W_t. \end{aligned}$$

Case 2d(ii): $f.c = 1$. Note that in this case, $i \neq f$ because $i.c = 0$ and $f.c = 1$. Let i be the first packet in S_t. Then either i appears after f in the buffer, or i appears before f in the buffer. We analyze each of these possibilities.

Now suppose i appears after f in the buffer. Since $f.c = 1$, we know that the dummy packet associated with f denoted f' satisfies $f'.c = 0$ and $f' \in M_t$. We may assume that $f' \notin S_t$, otherwise we can switch f' and i, and appeal to the case to be discussed later (the one in which i appears before f in the buffer).

Suppose $f \in S_t$, but $f' \notin S_t$. We obtain S_{t+1} from S_t by replacing f with i. Clearly, S_{t+1} is feasible.

Noting that the most f can contribute to V_t is w_f, we see that

$$\begin{aligned} V_t &\leq w_f + (1-1/\alpha)\cdot w_h = w_f + (1-1/\alpha)\cdot v_h \\ &\leq w_f + (1-1/\alpha)\cdot(\alpha\cdot v_f) = (2-1/\alpha)\cdot w_f \\ &\leq \alpha\cdot w_f \leq \beta\cdot w_f = \beta\cdot W_t. \end{aligned}$$

Here as $f.c = 1$, $v_f = w_f/\alpha < w_f$. But note that when f is sent per its slot in the current S_t, its status bit may be reset to zero; to account for this contingency, we let $V_t = w_f$ instead of the smaller value. Also the second term in V_t is due the h packet changing its status bit from zero to one.

If neither f nor f' belongs to S_t, S_{t+1} is obtained from S_t by deleting i; S_{t+1} is feasible because S_t is, and

$$\begin{aligned} V_t &= w_i + (1-1/\alpha)\cdot w_h < w_h\cdot(2-1/\alpha) \\ &= v_h\cdot(2-1/\alpha) \leq (2-1/\alpha)\cdot(\alpha\cdot v_f) \\ &= (2-1/\alpha)\cdot w_f \leq \alpha\cdot w_f \leq \beta\cdot w_f = \beta\cdot W_t. \end{aligned}$$

Suppose i appears before f in the buffer. Recall that at the end of this step, the algorithm will drop both f' and f and add a new dummy packet h'. We obtain S_{t+1} from S_t by dropping f' and i, and by replacing f with h'. (In other words, h' occupies f's slot.) Since h' has the same deadline as f, the new sequence is feasible. For this step,

$$\begin{aligned} V_t &= v_i + v_f + v_{f'} - v_{h'} + (w_h - v_h) \\ &= w_i + w_f/\alpha + w_f/\alpha - w_h/\alpha + (1-1/\alpha)\cdot w_h. \end{aligned}$$

Since i appears before f and was not chosen by the algorithm, we know that $v_i < v_h/\alpha$. This, along with $i.c = 0$ and $h.c = 0$, implies $w_i < w_h/\alpha$. Similarly, $v_f \geq v_h/\alpha$, which implies $w_f = \alpha\cdot v_f \geq v_h = w_h$. Using all of these in the expression for V_t, we get

$$\begin{aligned} V_t &\leq w_h/\alpha + 2\cdot w_f/\alpha + w_h - 2\cdot w_h/\alpha \\ &\leq (1+1/\alpha)\cdot w_f \ \leq\ \beta\cdot w_f = \beta\cdot W_t. \end{aligned}$$

5 Remarks

Independently of this work, Englert and Westermann [11] design an online algorithm for the same problem that we consider here. Their results include a 1.828-competitive algorithm for the buffer management problem, and a 1.893-competitive memoryless algorithm for the same problem. It is instructive to compare and contrast their algorithm with ours.

Their algorithm maintains what they call a "provisional schedule" (which is the same as the sequence of matched packets in our terminology), which is a schedule for all the pending packets in the buffer assuming no future arrivals. Unlike our algorithm, they do not use dummy packets. Instead they introduce the concept of a "suppressed" packet, defined as follows: if a packet p in the current provisional schedule is removed from the buffer, it may be possible to add a new packet q to the provisional schedule. In this case, p is said to suppress q (equivalently, q is suppressed by p). In evaluating whether or not to send a packet p in the current time step, most algorithms simply use the weight of packet p, perhaps in relation to other packets in the buffer. The key innovation in their algorithm is that in making this decision they take into account the weight of the packet suppressed by p. Thus, for each packet in the buffer, they compute a "rank", obtained by adding the packet's weight and a (fixed) fraction of the weight of the packet suppressed by it. Their algorithm makes a choice between the first packet in the provisional schedule and the maximum-rank packet in the buffer. This idea results in a memoryless algorithm that is 1.893-competitive. They improve the competitive ratio to 1.828 by letting the last step — deciding whether to send the first packet or the maximum rank packet — depend on the history.

In terms of the analysis, both analysis rest on an exhaustive enumeration of various cases. The key difference is that our analysis does not explicitly rely on a potential function argument, but their analysis does.

References

[1] S. Albers and M. Schmidt, *On the Performance of Greedy Algorithms in Packet Buffering*, In Proceedings of the 36th ACM Symposium on Theory of Computing (STOC), pages 35-44, 2004.

[2] N. Andelman, Y. Mansour, and A. Zhu, *Competitive Queuing Polices for QoS Switches*, In Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 761-770, 2003.

[3] Y. Azar and Y. Richter, *Management of Multi-queue Switches in QoS Networks*, In Proceedings of the 35th ACM Symposium on Theory of Computing (STOC), pages 82-89, 2003.

[4] Y. Azar and Y. Richter, *The Zero-one Principle for Switching Networks*, In Proceedings of the 36th ACM Symposium on Theory of Computing (STOC), pages 64-71, 2004.

[5] N. Bansal, L. K. Fleischer, T. Kimbrel, M. Mahdian, B. Schieber, and M. Sviridenko, *Further Improvements in Competitive Guarantees for QoS Buffering*, In Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP), pages 196-207, 2004.

[6] Y. Bartal, F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, R. Lavi, J. Sgall, and T. Tichy, *Online Competitive Algorithms for Maximizing Weighted Throughput of Unit Jobs*, In Proceedings of the 21st Symposium

on Theoretical Aspects of Computer Science (STACS), pages 190-210, 2004.

[7] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.

[8] M. Chrobak, W. Jawor, J. Sgall, and T. Tichy, *Improved Online Algorithms for Buffer Management in QoS Switches*, In Proceedings of the 12th European Symposium on Algorithms (ESA), pages 204-215, 2004.

[9] F. Y. L. Chin and S. P. Y. Fung, *Online Scheduling with Partial Job Values: Does Timesharing or Randomization Help?*, Algorithmica, Volume 37(3), pages 149-164, 2003.

[10] M. Englert and M. Westermann, *Lower and Upper Bounds on FIFO Buffer Management in QoS Switches*, In Proceedings of the 14th Annual European Symposium on Algorithms (ESA), pages 352-363, 2006.

[11] M. Englert and M. Westermann, *Considering Suppressed Packets Improves Buffer Management in QoS Switches*, to appear in Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007.

[12] B. Hajek, *On the Competitiveness of Online Scheduling of Unit-Length Packets with Hard Deadlines in Slotted Time*, In Proceedings of 2001 Conference on Information Sciences and Systems (CISS), pages 434-438, 2001.

[13] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko, *Buffer Overflow Management in QoS Switches*, SIAM Journal of Computing, Volume 33(3), pages 563-583, 2004.

[14] A. Kesselman, Y. Mansour, and R. van Stee, *Improved Competitive Guarantees for QoS Buffering*, Algorithmica, Volume 43, pages 63-80, 2005.

[15] F. Li, J. Sethuraman, and C. Stein, *An Optimal Online Algorithm for Packet Scheduling with Agreeable Deadlines*, In Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 801-802, 2005.

[16] Z. Lotker and B. Patt-Shamir, *Nearly Optimal FIFO Buffer Management for DiffServ*, In Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC), pages 134-142, 2002.

[17] M. Schmidt, *Packet Buffering: Randomization Beats Deterministic Algorithms*, In Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS), pages 293-304, 2005.

Appendix: Some Examples

In this appendix, we give some lower bound examples for DP and MG, and try to motivate further the use of dummy packets.

5.1 A lower bound on DP We are not able to construct an instance on which DP's competitive ratio is $3/\phi$. The best lower bound we have been able to achieve is via the following simple instance in which there are no arrivals.

Note that this example has agreeable deadlines. So DP clearly has a worse competitive ratio than algorithm MG on instances with agreeable deadlines.

Example 2. Consider the four packets $e_0 = (1-\epsilon, 1)$, $e_1 = (1-\epsilon, 2)$, $f_0 = (1, 3)$, and $h_0 = (\phi, 4)$. The optimal offline algorithm sends all the packets for a gain of $(3 + \phi - 2 \cdot \epsilon)$.

The algorithm sends f_0 first, and sets $h_0.c = 1$. A dummy packet $h'_0 = (1, 3)$ is generated, and e_0 is dropped; the buffer thus has the packets e_1, h'_0 and h_0 in that order at time 1. The algorithm then sends h_0, as h_0 has the maximum v-value 1. As $h_0.c = 1$, h'_0 is dropped when sending h_0. Finally, e_1 is dropped at time 2 because of its deadline. The buffer is empty, and the total gain of the algorithm is $1 + \phi$. The competitive ratio for this instance is thus $(3 + \phi - 2 \cdot \delta) \ / \ (1 + \phi) \approx 1.764$.

While we have not been able to find an example on which DP's competitive ratio is $3/\phi$, we can show that an improvement in DP's analysis must come from a more involved analysis. For the particular grouping we do (grouping steps of size 2), the following example shows that the bound of $3/\phi$ cannot be improved.

5.2 A lower bound on MG Algorithm MG [15] achieves a ϕ competitive ratio for agreeable deadline instances. It is natural to speculate that it might be better than 2 on general instances. However, the following example shows that MG's competitive ratio on general instances is (no better than) 2.

Example 3. Without loss of generality, we assume the first time step is 1 instead of 0 — this is for the ease of indexing groups of packets. We use ∞ in the deadline field of a packet to show that this packet's deadline is very large. Let $n = 2^k$. The packets are released in a stage-manner. There are $\log n = k$ stages. The superscript of a packet shows the stage in which it is released.

At the beginning of step 1, there are 3 packets in MG's buffer. The adversary has the same buffer. These 3 packets are $e_1^1 := (1, 2)$, $f_1^1 := (\phi - \epsilon, 2^{k+1} - k)$, and $h_1^1 := (\phi, \infty)$. MG sends h_1^1, and e_1^1 is dropped out of the buffer due to its deadline.

In each of the following $(2^k - k + 1)$ time steps, say step i, a group of 3 packets are released: $e_i^1 := (1, i+1)$, $f_i^1 := (\phi - \epsilon, 2^{k+1} - k)$, and $h_i^1 := (\phi, \infty)$. In step i, MG sends h_i^1 and drops e_i^1 due to its deadline. At the end of the $(2^k - k + 1)$-th step, MG's buffer is full of $(2^k - k + 1)$ f_i^1-packets ($\forall i = 1, 2, \ldots, 2^k - k + 1$). The first stage ends. The length of stage 1 guarantees that no f_i^1 packet, especially packet f_1^1, becomes the first packet in the buffer.

At the beginning of step $2^k - k + 1$, the second stage starts. The adversary releases a pair of packets $f_1^2 := (\phi \cdot (\phi - \epsilon) - \epsilon, 2^{k+1} - k + 1)$ and $h_1^2 := (\phi^2, \infty)$. The newly released packets have later deadlines and are sorted canonically after the packets already in MG's buffer. MG sends h_i^2. Stage 2 contains $2^{k-1} - k + 2$ steps. The length of stage 2 guarantees that no packet f_i^2 becomes the first packet in the buffer. In each of those $2^{k-1} - k + 2$ steps, say step i, 2 packets are released $f_i^2 := (\phi \cdot (\phi - \epsilon) - \epsilon, 2^{k+1} - k + 1)$ and $h_i^2 := (\phi^2, \infty)$. MG sends h_i^2 in step i. Stage 2 is half as long as stage 1.

We repeat this pattern in each stage, for k stages. Stage $i + 1$ is half as long as stage i. In each step j of stage i, 2 packets are released, $f_j^i := (\phi \cdot (w_{f_1^{i-1}} - \epsilon), 2^{k+1} - k + i)$ and $h_j^i := (\phi^i, \infty)$. MG sends h_j^i in step j. In the last stage, which is step 2^{k+1}, the adversary only releases 2 packets $f_1^k := (\phi^k, 2 \cdot n)$ and $h_1^k := (\phi^{k+1} + \epsilon, \infty)$. MG sends h_1^k and f_1^k is dropped out of the buffer due to its deadline.

For each step in stage i, MG only delivers the h^i packets, and eventually, all packet f^i are dropped out of the buffer due to their deadlines. On the contrary, the adversary sends all f^i packets and all h^i packets. A routine calculation shows that the optimal weighted throughput is nearly twice MG's weighted throughput.

5.3 The role of dummy packets Here, we show how dummy packets are used in our algorithm DP.

To show that dummy packets play an important role in scheduling packets, we first propose an algorithm called $Mark_\alpha$, which employs status bits but no dummy packets. $Mark_\alpha$ uses the same matched packets as DP. Each packet p has a status bit $p.c$ and $v_p = w_p/\alpha$ when $p.c = 1$. After identifying h, the maximum-v-value packet, if $h.c = 0$, $Mark_\alpha$ sends the earliest packet f such that $w_f \geq w_h/\alpha$. f can always be found since h itself is a candidate. If $f \neq e$ (e is the first packet), $h.c$ is marked 1. If $h.c = 1$, $Mark_\alpha$ sends h.

Example 4. Consider the following example for $Mark_\alpha$ where $\alpha = \phi$. Let n be a large number. At the beginning of a time step 0, there are 3 packets in the buffer: $\epsilon_0 := (\epsilon, 1), f_1 := (1, 2), h_1 := (\phi, n + 2)$. The algorithm sends f_1 and marks h_1 such that $h_1.c = 1$. At the beginning of step 1, 3 packets are released. $\epsilon_1 := (\epsilon, 2), f_2 := (1 + \epsilon, 3)$ and $h_2 := (\phi + \epsilon, n + 3)$. The algorithm sends f_2 and marks h_2, and so on.

In one of the following $n - 1$ steps, say step i, the adversary releases $\epsilon_i := (\epsilon, i + 1), f_i := (1 + i \cdot \epsilon, i + 2)$ and $h_i := (\phi + i \cdot \epsilon, n + i + 2)$. The algorithm sends f_i in step i and marks h_i. At the beginning of step n, the algorithm releases a single packet P with value ϕ and deadline $n + 1$. The algorithm sends this packet in step n.

At the beginning of step $n + 1$, $n - 1$ packets with value $(\phi - \epsilon, n + i + 1)$, where $i = 1, \ldots, n-1$, are released. All newly released packets are dropped out of the buffer due to the existence of those $n - 1$ h-packets. From the packets left in the buffer, the algorithm sends an h-packet in each time step since all these packets have their status bits set to 1. Note that h_{i-1} is sent first, then, h_{i-2}, and so on. Assume n is a large odd number, only $(n-1)/2$ packets are sent. Thus $Mark_\phi$'s weighted throughput is at most $(n-1) \cdot 1 + \phi + (\phi + n \cdot \epsilon) \cdot (n-1)/2$.

The adversary can send h_i ($i = 1, 2, \ldots, n - 1$) in the first $n - 1$ time steps, then, sends P in step n, and sends all newly released packets in the following $n - 1$ steps. The adversary's weighted throughput is at least $(n-1) \cdot \phi + \phi + (n-1) \cdot (\phi - \epsilon)$. Thus, on this instance, the adversary's weighted throughput is $(2 \cdot \phi)/(1 + \phi/2) \approx 1.788$ of $Mark_\alpha$'s weighted throughput for large n and small enough ϵ.

In the above example, DP works the same as $Mark_\alpha$ for the first n steps. At the beginning of step $n + 1$, all dummy packets are dropped due to their deadlines, and thus, all packets in the buffer have status bits reset 0. In step $n + 1$, the algorithm DP will send the first packet h_1 instead of the highest-v-value packet h_{n-1}. DP will send all packets in the buffer. The total value gained by DP is at least $(n-1) \cdot 1 + \phi + (n-1) \cdot \phi$. It is easy to verify that the optimal weighted throughput is only $(2 \cdot \phi)/(1 + \phi) \approx 1.236$ of the weighted throughput of DP, when n is large and ϵ is small enough. This is substantially better than $Mark_\alpha$ performance on the same instance.