On the Advice Complexity of Buffer Management

Reza Dorrigiv[⋆], Meng He[⋆⋆], and Norbert Zeh[⋆⋆⋆]

Faculty of Computer Science, Dalhousie University,
Halifax, NS, B3H 1W5, Canada,
{rdorrigiv,mhe,nzeh}@cs.dal.ca

Abstract. We study the advice complexity of online buffer management. Advice complexity measures the amount of information about the future that an online algorithm needs to achieve optimality or a good competitive ratio. We study the 2-valued buffer management problem in both preemptive and nonpreemptive models and prove lower and upper bounds on the number of bits required by an optimal online algorithm in either model. We also provide results that shed light on the ineffectiveness of advice to improve the competitiveness of the best online algorithm for nonpreemptive buffer management.

1 Introduction

Buffer management is an important online problem with applications in network communication [13]. It models admission policies for the buffers of packet switches in networks that support the QoS (Quality of Service) feature. In this problem, packets have different *values* corresponding to their importance or priority, and the packet switch has a FIFO buffer of size B. Packets arrive at arbitrary times and are placed at the end of the buffer. In each time unit, the switch retrieves a packet from the front of the buffer and transmits it, unless the buffer is empty. Multiple packets may arrive in the same time unit. If the buffer is full when a packet arrives, the switch's buffer management algorithm must reject the packet. Otherwise it may choose to accept or reject the packet. The goal of the algorithm is to maximize the total value of the transmitted packets, which, in the absence of preemption, are exactly the packets it accepts. We consider the *2-valued model* here, where there are two types of packets: low-priority packets (L-packets) of value 1 and high-priority packets (H-packets) of value $\alpha > 1$. We use $p_1, p_2, \ldots, p_n$ to denote the sequence of packets, and we assume for simplicity that transmission occurs at integral times, no two packets arrive at the same time, and no packets arrive at integral times. Time starts at 0. For an integer $j \geq 0$, we define *time unit* j to be the time interval $[j, j+1)$.

[⋆] Research supported by an NSERC postdoctoral fellowship.
[⋆⋆] Research supported by NSERC.
[⋆⋆⋆] Research supported by NSERC and the Canada Research Chairs programme.

There are two main models for buffer management. In the *nonpreemptive model* all packets accepted into the buffer are eventually transmitted, that is, accepted packets cannot be dropped at a later time. In the *preemptive model*, accepted packets can be dropped as long as they have not been transmitted yet. Online buffer management algorithms are usually analyzed using competitive analysis [19]: Let OPT be an optimal offline buffer management algorithm, and $\mathcal{A}$ an online algorithm. We use $\mathrm{OPT}(S)$ and $\mathcal{A}(S)$ to denote the solutions produced by these algorithms on an input sequence S, or the total values of the packets in these solutions. Which will be clear from the context. Algorithm $\mathcal{A}$ has *competitive ratio* c if $OPT(S) \leq c \cdot \mathcal{A}(S)$, for every request sequence S.

Competitive analysis of buffer management algorithms was initiated by Aiello et al. [1], Mansour et al. [17], and Kesselman et al. [16]. Since then, various algorithms have been proposed. We briefly review the results most relevant to our work and refer the reader to surveys by Azar [5], Epstein and van Stee [12], and Goldwasser [13] for a more comprehensive coverage. Aiello et al. [1] introduced the nonpreemptive 2-valued model and proved a lower bound of $(2 - \frac{1}{\alpha})$ on the competitiveness of any deterministic or randomized online algorithm in this model. The RATIO PARTITION algorithm by Andelman, Mansour, and Zhu [4] achieves this bound. This algorithm accepts each H-packet if possible and, for each accepted H-packet, marks the earliest $\frac{\alpha}{\alpha-1}$ unmarked L-packets in the buffer. It accepts an L-packet only if, after accepting it, the number of unmarked L-packets in the buffer is at most $\frac{\alpha}{\alpha-1}$ times the number of empty buffer slots. Mansour et al. introduced the preemptive model in the context of video streaming [17]. Kesselman et al. proved a lower bound of 1.282 on the competitiveness of any deterministic preemptive online algorithm [16]. The ACCOUNT STRATEGY algorithm by Englert and Westermann [4] achieves this bound.

These results completely characterize the competitiveness achievable for nonpreemptive or preemptive online buffer management without information about future requests. They do not, however, provide any insight into how much an algorithm may benefit from partial information about future requests that may be available in some applications. Various models have been proposed to facilitate the analysis of online algorithms that have access to such partial information, e.g., the finite lookahead model [15, 14, 2, 3, 9]. A more recent model, and the one we adopt in this paper, is *advice complexity* [10, 11, 7]. It measures the amount of information about the future that an online algorithm needs in order to achieve optimality or a certain competitive ratio. Two variants of this model have been proposed [11, 7]. In the model by Böckenhauer et al. [7], the online algorithm $\mathcal{A}$ has access to a tape of advice bits produced by an oracle. The oracle has unlimited computational power and has access to the whole input. No restrictions are placed on how $\mathcal{A}$ uses the advice bits. The performance of $\mathcal{A}$ is expressed as a combination of its competitive ratio and the number of advice bits it uses on an input of size n. In the model by Emek et al. [11], the oracle provides a fixed number of advice bits with each request, and the algorithm has access only to the advice bits associated with the requests that have arrived so far. Previous work on advice complexity of online algorithms has focused on paging [10, 11], ski rental [10], metrical task systems [11], the k-server problem [11, 6, 18], job shop scheduling and routing [7], and the knapsack problem [8]. To the best of our knowledge, the advice complexity of online buffer management has not been studied so far. This is the focus of this paper.

It is trivial even for a nonpreemptive buffer management algorithm to achieve optimality with one bit of advice per request: the oracle runs an offline optimal buffer management algorithm on the given request sequence and tells the online algorithm for each packet whether the optimal algorithm accepts or rejects this packet. Since one bit of advice per request is the minimum possible in the model of [11], online buffer management is not interesting in this model, and we adopt the model of [7]. Our main result is that $\Theta((n/B)\log B)$ bits of advice are necessary and sufficient for a preemptive or nonpreemptive online buffer management algorithm to produce an optimal solution. In this paper all logarithms are base 2. We also prove that a generalization of the RATIO PARTITION algorithm, which uses advice to choose the optimal ratio between unmarked L-packets and empty buffer slots, cannot outperform RATIO PARTITION without advice. We conjecture that an algorithm that chooses different ratios for different parts of the input *can* outperform RATIO PARTITION, but we were unable to prove this.

2 Optimal Preemptive Online Buffer Management

We focus on preemptive buffer management first, as our results for the nonpreemptive case are extensions of the ones for the preemptive case. We prove that, for a preemptive buffer management algorithm, $\Theta((n/B)\log B)$ bits of advice are sufficient and necessary to achieve optimality.

2.1 The Lower Bound

Theorem 1. *Any optimal preemptive online buffer management algorithm requires at least* $(n/(3B))\log(B+1)$ *bits of advice.*

Proof. Consider the following family of request sequences of length between $2B$ and $3B$: In time unit 0, B L-packets arrive. Between times 1 and $B+1$, one H-packet arrives per time unit. In time unit $B+1$, k H-packets arrive, where $0 \le k \le B$. In the next B time units, no packets arrive. The optimal solution accepts $B-k$ of the L-packets that arrive in time unit 0 and rejects the other L-packets. It then accepts all H-packets that arrive in subsequent time units.

The online algorithm also has to accept exactly $B-k$ L-packets in time unit 0. To see this, observe that, no matter how many of the L-packets are accepted, the algorithm will not preempt them. This is true because between times 1 and B, the buffer does not overflow, which implies that by time $B+1$, all L-packets have been transmitted and the buffer contains h H-packets, where h is the number of L-packets we accepted in time unit 0. Now, if $h > B-k$, we are forced to reject one of the k H-packets that arrive in time unit $B+1$, which leads to a suboptimal solution. If $h < B-k$, we could have accepted at least one more

L-packet in time unit 0 without forcing us to reject an H-packet in time unit $B+1$, which is again suboptimal.

Since at time 0 the algorithm needs to know the number, k, of H-packets that arrive in time unit $B+1$, and k can assume any value between 0 and B, we need $\log(B+1)$ bits of advice for this sequence of length between $2B$ and $3B$.

We can construct arbitrarily long inputs of this type by concatenating a number, q, of such request sequences $S_1, S_2, \ldots, S_q$. The last B time units of each sequence S_i during which no packets arrive guarantee that the algorithm needs to behave on each S_i as if S_i were the whole request sequence. Thus, at least $\log(B+1)$ bits of advice are needed for each S_i, for a total of $q\log(B+1)$ bits. The length of the entire sequence $S_1S_2\ldots S_q$ is $2qB \le n \le 3qB$. Thus, we need at least $(n/(3B))\log(B+1)$ bits of advice. $\square$

2.2 The Upper Bound

We now describe an optimal preemptive online buffer management algorithm that matches the lower bound of Theorem 1 up to a constant factor.

Theorem 2. *There exists an optimal preemptive online buffer management algorithm that uses $\lceil n/B\rceil\lceil\log(B+1)\rceil$ bits of advice.*

To prove Theorem 2, we consider a request sequence S and a "canonical" optimal solution $\mathrm{OPT_C}(S)$ for S, and we propose an online algorithm $\mathcal{A}$ that uses $\lceil n/B\rceil\lceil\log(B+1)\rceil$ bits of advice to produce this solution. Note that an optimal offline algorithm cannot benefit from preemption because it can immediately reject any packets it would preempt later. Thus, we define $\mathrm{OPT_C}(S)$ by describing a nonpreemptive optimal offline algorithm that produces $\mathrm{OPT_C}(S)$.

We divide S into contiguous subsequences $U_0, U_1, \ldots, U_T$, where U_t is the subsequence of requests that arrive in time unit t. Let H_t and L_t respectively be the numbers of H- and L-packets in U_t; let H'_t and L'_t respectively be the numbers of H- and L-packets in U_t that we accept; and let $Y_t = 1$ if we transmit a packet at time t, and $Y_t = 0$ otherwise. Since the buffer has capacity B, any feasible solution satisfies

$$\sum_{t=0}^{t'}(H'_t + L'_t) \le B + \sum_{t=0}^{t'} Y_t, \tag{1}$$

for all $0 \le t' \le T$. Since we can transmit a packet only if we have not already transmitted all the accepted packets, a feasible solution must also satisfy

$$\sum_{t=0}^{t'-1}(H'_t + L'_t) \ge \sum_{t=0}^{t'} Y_t, \tag{2}$$

for all $0 \le t' \le T$. Conversely, any set of values of H'_t, L'_t and Y'_t, $0 \le t \le T$, that satisfy (1) and (2) (as well as the trivial constraints that $0 \le H'_t \le H_t$, $0 \le L'_t \le L_t$, and $Y_t \in \{0,1\}$, for all $0 \le t \le T$) yields a feasible solution.

The construction of $\mathrm{OPT_C}(S)$ starts with $Y_t = H'_t = L'_t = 0$, for all $0 \le t \le T$. Next we greedily increase the number of H-packets accepted in each time unit and then greedily increase the number of L-packets accepted in each time unit, given the set of accepted H-packets. More precisely, we proceed in two rounds. In the first round, we iterate over $t' := 0, 1, \ldots, T$ and set $Y_{t'} := 1$ if $\sum_{t=0}^{t'-1} H'_t > \sum_{t=0}^{t'-1} Y_t$, and $Y_{t'} := 0$ otherwise; then we set $H'_{t'} := \min(H_{t'}, B + \sum_{t=0}^{t'} Y_t - \sum_{t=0}^{t'-1} H'_t)$. Before the second round, we set $Y_0 := 0$ and $Y_t := 1$, for all $1 \le t \le T$. Now we iterate over $t' := 0, 1, \ldots, T$ again. For time t', we update $Y_{t'}$ so that $Y_{t'} := 1$ if $\sum_{t=0}^{t'-1}(H'_t + L'_t) > \sum_{t=0}^{t'-1} Y_t$, and $Y_{t'} := 0$ otherwise. Next we choose $L'_{t'}$ maximally so that $L'_{t'} \le L_{t'}$ and $\sum_{t=0}^{t''}(H'_t + L'_t) \le B + \sum_{t=0}^{t''} Y_t$, for all $t' \le t'' \le T$. $\mathrm{OPT_C}(S)$ accepts the first H'_t H-packets and the first L'_t L-packets in each subsequence U_t, and rejects all other packets. It transmits a packet at time t if and only if $Y_t = 1$. The proof of the following lemma is omitted due to lack of space.

Lemma 1. $\mathrm{OPT_C}(S)$ *is an optimal solution for the request sequence* S*.*

Next we describe a preemptive online algorithm with advice, $\mathcal{A}$, that computes $\mathrm{OPT_C}(S)$. We divide the request sequence S into $q := \lceil n/B \rceil$ subsequences $S_1, S_2, \ldots, S_q$, which we call *phases*. For $1 \le i < q$, $|S_i| = B$. S_q contains the remaining $n-(q-1)B$ requests. For each phase S_i, the advice given to the algorithm is the number, a_i, of L-packets $\mathrm{OPT_C}(S)$ accepts from S_i. Since $|S_i| \le B$, this requires $\lceil \log(B+1) \rceil$ bits of advice for each S_i, and thus $\lceil n/B \rceil \lceil \log(B+1) \rceil$ bits in total. The online algorithm now processes the packets in S one by one. While processing the requests in S_i, it keeps a count, c_i, of the number of L-packets in S_i that it has accepted and not preempted. Immediately before processing the first request in S_i, we set $c_i := 0$. For each L-packet in S_i, if the buffer is not full and $c_i < a_i$, we accept the packet and increase c_i by one; otherwise we reject the packet. For each H-packet in S_i, if the buffer is not full, we accept the packet. If the buffer is full but contains an L-packet, we preempt the *most recently* queued L-packet, decrease c_i by one, and accept the H-packet. If the buffer is full and contains only H-packets, we reject the packet. Let $\mathcal{A}(S)$ be the solution this algorithm produces on input S. Together with Lemma 1, the next lemma implies that $\mathcal{A}(S)$ is an optimal solution.

Lemma 2. $\mathcal{A}(S) = \mathrm{OPT_C}(S)$.

Proof sketch. We use induction on i to prove: (i) The set of packets from S_i accepted and not preempted by $\mathcal{A}(S)$ by the end of phase S_i is the set of packets $\mathrm{OPT_C}(S)$ accepts (and does not preempt) from S_i. (ii) While processing the packets in S_i, $\mathcal{A}(S)$ does not preempt any packets from $S_1 S_2 \ldots S_{i-1}$. These two claims together imply that $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ transmit the same set of packets.

For $i = 0$, the two claims hold vacuously, so assume $i > 0$ and the two claims hold for phases $S_0, S_1, \ldots, S_{i-1}$. Then the buffer states of $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ at the beginning of phase S_i are identical. First we prove that $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ accept the same H-packets from S_i. Since $\mathrm{OPT_C}(S)$ and $\mathcal{A}(S)$

accept H-packets greedily, this follows if no L-packet accepted by $\mathcal{A}(S)$ forces it to reject an H-packet accepted by $\mathrm{OPT_C}(S)$. Any L-packet accepted but not yet transmitted by $\mathcal{A}(S)$ can be preempted if necessary to make room for an H-packet and thus does not force the rejection of an H-packet. After transmitting the first L-packet p_j from S_i and before processing the last packet from S_i, the buffer contains a subset of the packets $p_{j+1}, p_{j+2}, \ldots, p_{iB}$ and thus is not full. Thus, after transmitting p_j, no H-packet from S_i is rejected by $\mathcal{A}(S)$.

Next we show that $\mathcal{A}(S)$ does not preempt any packets from $S_0, S_1, \ldots, S_{i-1}$ while processing the packets in S_i. Let p_j once again be the first L-packet from S_i transmitted by $\mathcal{A}(S)$. We have just proved that $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ accept the same set of H-packets from S_i. We also argued that no H-packet p_k in S_i that succeeds p_j forces any preemption because $\mathcal{A}(S)$'s buffer cannot be full when p_k arrives. If p_k precedes p_j, it can force a preemption only if $\mathcal{A}(S)$'s buffer is full when p_k arrives. Since the buffer states of $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ are identical at the beginning of the ith phase and $\mathrm{OPT_C}(S)$ can accept p_k without preempting any packet, $\mathcal{A}(S)$'s buffer must contain an L-packet p_h from S_i that is not in $\mathrm{OPT_C}(S)$'s buffer. Thus, $\mathcal{A}(S)$ preempts $p_h \in S_i$ to make room for p_k.

It remains to prove that, by the end of phase S_i, the set of L-packets from S_i accepted but not preempted by $\mathcal{A}(S)$ is the same as the set of L-packets $\mathrm{OPT_C}(S)$ accepts from S_i. Since none of these packets are preempted in subsequent phases, these are exactly the L-packets from S_i *transmitted* by $\mathcal{A}(S)$. We prove here that $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ transmit the same *number* of L-packets from S_i. This is the first part of the proof. The second part of the proof uses this fact to prove that $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ transmit the same *set* of L-packets from S_i. The proof of this second part is omitted due to lack of space.

$\mathcal{A}(S)$ cannot transmit more L-packets than $\mathrm{OPT_C}(S)$ because it bounds the number of L-packets from S_i it has accepted and not preempted by a_i. Next we prove that, after processing each packet p_j in S_i, the number of L-packets from S_i that $\mathcal{A}(S)$ has accepted and not preempted so far is no less than the number of L-packets $\mathrm{OPT_C}(S)$ has accepted up to that point. We use induction on the position of p_j in S_i. Before processing the first packet in S_i, the claim holds. If $\mathcal{A}(S)$ accepts p_j without preempting any other packet or if it rejects p_j because $c_i = a_i$, the invariant is maintained. If p_j is an H-packet and $\mathcal{A}(S)$ preempts an L-packet in favour of p_j or p_j is an L-packet and $\mathcal{A}(S)$ rejects it, then $\mathcal{A}(S)$'s buffer is full when p_j arrives. Since $\mathcal{A}(S)$ had accepted and not preempted at least as many packets as $\mathrm{OPT_C}(S)$ after processing each of the packets $p_1, p_2, \ldots, p_{j-1}$, $\mathcal{A}(S)$ also transmitted at least as many packets as $\mathrm{OPT_C}(S)$ by the time each of these packets was processed. The number of packets any algorithm can accept up to a certain point is B plus the number of packets it has transmitted so far. Since $\mathcal{A}(S)$'s buffer is full after processing p_j, $\mathcal{A}(S)$ has accepted and not preempted exactly this number of packets so far, while $\mathrm{OPT_C}(S)$ cannot have accepted more packets by this time. Since we already proved that $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ accept the same set of H-packets, this shows that the number of L-packets accepted and not preempted by $\mathcal{A}(S)$ by the time p_j has been processed is at least the number of L-packets accepted by $\mathrm{OPT_C}(S)$ by this time. □

3 Optimal Nonpreemptive Online Buffer Management

In this section, we prove the somewhat surprising result that, up to constant factors, the same number of bits of advice required for an optimal preemptive online buffer management algorithm suffice for a nonpreemptive online buffer management algorithm to achieve optimality. Our lower bound for nonpreemptive online buffer management is slightly stronger than for the preemptive case.

Theorem 3. *There exists an optimal nonpreemptive online buffer management algorithm that achieves optimality using* $\lceil n/B \rceil (3\lfloor \log B \rfloor + 4)$ *bits of advice. Moreover, any optimal nonpreemptive online buffer management algorithm requires at least* $(n \log(B+1))/(2B)$ *bits of advice.*

The lower bound proof is similar to the lower bound proof for the preemptive case and is thus omitted. In the remainder of this subsection, we prove the upper bound. We describe an online algorithm $\mathcal{A}$ that accepts the same number of H-packets and L-packets as the canonical optimal solution $\mathrm{OPT_C}(S)$ from Section 2, even though the sets of accepted packets may differ.

$\mathcal{A}$ accepts H-packets greedily, that is, it accepts each H-packet when it arrives unless the buffer is full. This does not require any advice. To define the advice that determines the behaviour of $\mathcal{A}$ for L-packets, we divide the request sequence S into $q := \lceil n/B \rceil$ phases $S_1, S_2, \ldots, S_q$ as in Section 2. For $1 \le i \le q$, let s_i and e_i be the time units during which the first and last packets in S_i arrive, respectively, and let f_i be the number of packets in $\mathrm{OPT_C}(S)$'s buffer just before the arrival of the first packet in S_i. Observe that $f_0 = 0$. We distinguish between three different cases depending on the behaviour of $\mathrm{OPT_C}(S)$ in phase S_i.

- If $\mathrm{OPT_C}(S)$'s buffer is never full during phase S_i, we call S_i a type-I phase. In this case, $\mathcal{A}$'s advice consists of the number, a_i, of L-packets in S_i $\mathrm{OPT_C}(S)$ accepts. $\mathcal{A}(S)$ accepts the first a_i L-packets in S_i it can accept and rejects the remaining L-packets in S_i.
- If $\mathrm{OPT_C}(S)$'s buffer is full at least once during phase S_i and all packets in S_i arrive over at most f_i time units (i.e., $e_i - s_i + 1 \le f_i$), we call S_i a type-II phase. In this case, $\mathcal{A}$'s advice consists of the number, r_i, of L-packets in S_i $\mathrm{OPT_C}(S)$ rejects. $\mathcal{A}(S)$ rejects the first r_i packets in S_i and accepts the remaining L-packets in S_i. (We prove below that it can accept these packets.)
- If $\mathrm{OPT_C}(S)$'s buffer is full at least once during phase S_i and the packets in S_i arrive over more than f_i time units (i.e., $e_i - s_i + 1 > f_i$), we call S_i a type-III phase. Let t_i be the last time unit in S_i when $\mathrm{OPT_C}(S)$'s buffer is full. We divide S_i into two subphases: S_i^1 contains the packets in S_i that arrive between time units s_i and t_i, inclusive, and S_i^2 contains the remaining packets in S_i. $\mathcal{A}$'s advice consists of t_i, the number, r'_i, of L-packets in S_i^1 rejected by $\mathrm{OPT_C}(S)$, and the number, a'_i, of L-packets in S_i^2 accepted by $\mathrm{OPT_C}(S)$. $\mathcal{A}(S)$ then treats S_i^1 as a type-II phase and S_i^2 as a type-I phase.

To encode this advice, we use one bit per phase to indicate whether it is a type-III phase. If it is, then the next $3(\lfloor \log B \rfloor + 1)$ bits represent x_i, r'_i and

a'_i. If not, we use another bit to indicate whether this phase is of type I or II and accordingly encode a_i or r_i using $\lfloor \log B \rfloor + 1$ bits. In the worst case, we use $1 + 3(\lfloor \log B \rfloor + 1) = 3\lfloor \log B \rfloor + 4$ bits per phase and $\lceil n/B \rceil (3\lfloor \log B \rfloor + 4)$ bits for the whole sequence. It remains to show that $\mathcal{A}(S)$ is an optimal solution.

Lemma 3. *$\mathcal{A}(S)$ is an optimal solution.*

Proof. It suffices to prove the following three claims for every phase S_i: (i) The buffers of $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ contain the same number of packets at the end of phase S_i. (ii) $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ accept the same number of H-packets in S_i. (iii) $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ accept the same number of L-packets in S_i.

For $i = 0$, these claims hold vacuously, so assume $i > 0$ and (i)–(iii) hold for phases $S_0, S_1, \ldots, S_{i-1}$. By the induction hypothesis, $\mathcal{A}(S)$'s and $\mathrm{OPT_C}(S)$'s buffers both contain f_i packets at the beginning of phase S_i. We prove that (i)–(iii) hold for phase S_i by considering the three possible types of S_i.

If S_i is of type I, $\mathrm{OPT_C}(S)$ does not reject any H-packet in S_i because $\mathrm{OPT_C}(S)$'s buffer is never full during S_i. Moreover, the L-packets in S_i accepted by $\mathrm{OPT_C}(S)$ are exactly the first a_i L-packets in S_i. To see this, observe that $\mathrm{OPT_C}(S)$ rejects an L-packet p_j in S_i and accepts a subsequent L-packet p_k in S_i only if its buffer is full when p_j arrives or accepting p_j forces $\mathrm{OPT_C}(S)$ to reject an H-packet p_l in S_i. (If accepting p_j forces the rejection of an H-packet after phase S_i, so does accepting p_k. Since $\mathrm{OPT_C}(S)$ never accepts an L-packet that forces the rejection of an H-packet, accepting p_j can only force the rejection of an H-packet in S_i.) Then, however, $\mathrm{OPT_C}(S)$'s buffer must be full either when p_j arrives or when p_l arrives, a contradiction because S_i is of type I. Now, since $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ have the same number of packets in their buffers at the beginning of S_i, $\mathcal{A}(S)$ can also accept the first a_i L-packets and all the H-packets in this phase without filling its buffer. Therefore, $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ accept the same set of packets in S_i and claims (i)–(iii) hold.

Next consider the case when S_i is of type II. Since all packets of S_i arrive over at most f_i time units and the buffers of $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ contain f_i packets at the beginning of S_i, $\mathcal{A}(S)$'s and $\mathrm{OPT_C}(S)$'s buffers are never empty during this phase. Thus, $\mathcal{A}(S)$ and $\mathrm{OPT_C}(S)$ transmit the same number of packets during phase S_i, and claim (i) follows if we can prove claims (ii) and (iii).

First (ii). We observe that, at any time during S_i, the number of L-packets from S_i that $\mathcal{A}(S)$ has accepted so far cannot be larger than the number of L-packets from S_i that $\mathrm{OPT_C}(S)$ has accepted so far. This is true because $\mathcal{A}(S)$ rejects the first r_i L-packets it receives during S_i, while $\mathrm{OPT_C}(S)$ rejects a total of r_i L-packets from S_i. Since $\mathcal{A}(S)$'s buffer and $\mathrm{OPT_C}(S)$'s buffer contain the same number of packets at the beginning of S_i and $\mathcal{A}(S)$ accepts H-packets greedily, this implies that, at any time during S_i, the number of H-packets from S_i accepted by $\mathcal{A}(S)$ so far is no less than the number of such packets accepted by $\mathrm{OPT_C}(S)$ so far. Conversely, $\mathrm{OPT_C}(S)$ accepts H-packets greedily and accepts an L-packet only if this does not prevent it from accepting an H-packet it could otherwise have accepted. Thus, the number of H-packets from S_i accepted by $\mathcal{A}(S)$ up to some point during S_i is no greater than the number of such packets accepted by $\mathrm{OPT_C}(S)$ up to this point.

To show that $\mathcal{A}(S)$ and $\mathrm{OPT}_{\mathrm{C}}(S)$ accept the same number of L-packets, we prove that $\mathcal{A}(S)$ rejects only the first r_i L-packets in S_i. Assume the contrary. Among the L-packets after the first r_i L-packets in S_i, let p_j be the first L-packet rejected by $\mathcal{A}(S)$. This means that the buffer of $\mathcal{A}(S)$ is full when p_j arrives. As shown in the previous paragraph, we know that up to this point, $\mathcal{A}(S)$ and $\mathrm{OPT}_{\mathrm{C}}(S)$ have accepted the same number of H-packets from S_i, and $\mathcal{A}(S)$ has accepted no more L-packets from S_i than $\mathrm{OPT}_{\mathrm{C}}(S)$ has. This implies that $\mathrm{OPT}_{\mathrm{C}}(S)$'s buffer is also full when p_j arrives, and $\mathrm{OPT}_{\mathrm{C}}(S)$ has to reject p_j. Since both algorithms have f_i packets in their buffers at the beginning of S_i, they transmit the same number of packets up to the arrival of p_j, and both algorithms' buffers are full when p_j arrives, they must both have rejected the same number of L-packets from S_i before the arrival of p_j. Since $\mathcal{A}(S)$ rejects r_i L-packets from S_i before p_j, so does $\mathrm{OPT}_{\mathrm{C}}(S)$, and p_j is the $(r_i + 1)$st L-packet from S_i rejected by $\mathrm{OPT}_{\mathrm{C}}(S)$, which is a contradiction.

Finally, consider the case when S_i is a type-III phase. Since there are at most B packets in S_i and $\mathrm{OPT}_{\mathrm{C}}(S)$'s buffer is full at time t_i, the buffer can never run empty during subphase S_i^1. Thus, $\mathrm{OPT}_{\mathrm{C}}(S)$ transmits a packet in each time unit between s_i and t_i, and $\mathrm{OPT}_{\mathrm{C}}(S)$'s buffer contains at most $f_i + |S_i^1| - (t_i - s_i + 1) \le f_i + B - (t_i - s_i + 1)$ packets at the end of time unit t_i. Since $\mathrm{OPT}_{\mathrm{C}}(S)$'s buffer is full at the end of time unit t_i, this implies that $t_i - s_i + 1 \le f_i$. Thus, the argument for type-II phases shows that $\mathcal{A}(S)$ and $\mathrm{OPT}_{\mathrm{C}}(S)$ accept the same number of packets from S_i^1, and the argument for type-I phases shows that they accept the same number of packets from S_i^2. This completes the proof. □

4 Advice Does Not Help Ratio Partition

The final question we investigate is whether using advice to adjust the ratio in the Ratio Partition algorithm helps. More precisely, we consider a class of algorithms $\mathit{\Gamma}(\tau)$. For a fixed parameter τ, $\mathit{\Gamma}(\tau)$ accepts each H-packet whenever possible and marks the τ earliest unmarked L-packets in the buffer. It accepts an L-packet only if after accepting it, the number of unmarked L-packets in the buffer is at most τ times the number of empty slots in the buffer. Ratio Partition is the same as $\mathit{\Gamma}(\frac{\alpha}{\alpha-1})$. Let Best-Threshold be an algorithm that uses advice to choose the best possible threshold τ for the given input and then runs $\mathit{\Gamma}(\tau)$. The following result shows that this use of advice is ineffective, that is, that Best-Threshold is no better than Ratio Partition.

Theorem 4. *The competitive ratio of* Best-Threshold *is at least* $2 - 1/\alpha$.

To prove Theorem 4, we ask the reader to verify that $\mathit{\Gamma}(\tau)$, for *any* $\tau \in [0, \infty)$, achieves a competitive ratio of exactly $2-1/\alpha$ on the following input S. S consists of α subsequences, each spanning $B + 1$ time units. In the first subsequence, B L-packets arrive in time unit 0, followed immediately by B H-packets in the same time unit. No further packets arrive in the remaining B time units of this subsequence. For each of the remaining $\alpha - 1$ subsequences, B L-packets arrive in time unit 0, and no further packets arrive in the remaining B time units. The key to this proof is that any threshold that is good for the first subsequence of S is bad for the remaining α subsequences and vice versa. We conjecture that an *adaptive* threshold algorithm, which chooses different thresholds for different portions of the input, can achieve a better competitive ratio.

Acknowledgements. We would like to thank Marc Renault and two anonymous reviewers for noticing an error in the original lower bound proof for non-preemptive online buffer management and for suggesting a way to correct it.

References

1. Aiello, W., Mansour, Y., Rajagopolan, S., Rosen, A.: Competitive queue policies for differentiated services. In: INFOCOM. pp. 414–420 (2000)
2. Albers, S.: On the influence of lookahead in competitive paging algorithms. Algorithmica 18(3), 283–305 (1997)
3. Albers, S.: A competitive analysis of the list update problem with lookahead. Theoretical Computer Science 197(1-2), 95–109 (May 1998)
4. Andelman, N., Mansour, Y., Zhu, A.: Competitive queueing policies for QoS switches. In: SODA. pp. 761–770 (2003)
5. Azar, Y.: Online packet switching. In: WAOA. pp. 1–5 (2005)
6. Böckenhauer, H., Komm, D., Královic, R., Královic, R.: On the advice complexity of the k-server problem. In: ICALP (1). pp. 207–218 (2011)
7. Böckenhauer, H., Komm, D., Královic, R., Královic, R., Mömke, T.: On the advice complexity of online problems. In: ISAAC. pp. 331–340 (2009)
8. Böckenhauer, H., Komm, D., Královic, R., Rossmanith, P.: On the advice complexity of the knapsack problem. In: LATIN. pp. 61–72 (2012)
9. Breslauer, D.: On competitive on-line paging with lookahead. Theoretical Computer Science 209(1-2), 365–375 (1998)
10. Dobrev, S., Královic, R., Pardubská, D.: Measuring the problem-relevant information in input. ITA 43(3), 585–613 (2009)
11. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. In: ICALP (1). pp. 427–438 (2009)
12. Epstein, L., van Stee, R.: Buffer management problems. ACM SIGACT News 35(3), 58–66 (2004)
13. Goldwasser, M.H.: A survey of buffer management policies for packet switches. SIGACT News 41(1), 100–128 (Mar 2010)
14. Grove, E.F.: Online bin packing with lookahead. In: SODA. pp. 430–436 (1995)
15. Kao, M., Tate, S.R.: Online matching with blocked input. Information Processing Letters 38(3), 113–116 (1991)
16. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. In: STOC. pp. 520–529 (2001)
17. Mansour, Y., Patt-Shamir, B., Lapid, O.: Optimal smoothing schedules for real-time streams (extended abstract). In: PODC. pp. 21–29 (2000)
18. Renault, M.P., Rosén, A.: On online algorithms with advice for the k-server problem. In: WAOA. pp. 198–210 (2011)
19. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28(2), 202–208 (1985)