ONLINE SCHEDULING OF PACKETS WITH AGREEABLE DEADLINES*

ŁUKASZ JEŻ†, FEI LI‡, JAY SETHURAMAN§, AND CLIFFORD STEIN¶

Abstract. This paper concerns an online packet scheduling problem that arises as a natural model for buffer management at a network router. Packets arrive at a router at integer time steps, and are buffered upon arrival. Packets have non-negative weights and integer deadlines that are (weakly) increasing in their arrival times. In each integer time step, at most one packet can be sent. The objective is to maximize the sum of the weights of the packets that are sent by their deadlines. The main results include an optimal ($\phi := (1+\sqrt{5})/2 \approx 1.618$)-competitive deterministic online algorithm and a ($4/3 \approx 1.33$)-competitive randomized online algorithm against an oblivious adversary. The analysis does not use a potential function explicitly, but instead modifies the adversary's buffer and credits the adversary to account for these modifications.

Key words. online algorithms; competitive analysis; buffer management; packet scheduling

AMS subject classifications. 68W40, 68W01, 68W05

1. Introduction. Buffer management at routers is a critical issue in providing effective quality of service to various Internet applications. Motivated by this consideration, Kesselman et al. [20, 21] 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 non-negative *weight* w and an integer *deadline* d before which it must be transmitted. At each integer time step, exactly one packet can be sent. 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. This paper deals with an important special case of the problem in which the packet deadlines are (weakly) increasing in their release times — the *agreeable deadline* model.

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 weighted 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. The scheduling algorithm, therefore, is required to make its decisions at any step based only on all the packets that have arrived so far, without making any assumptions about future arrivals. Such an algorithm is called

*The extended abstracts of this paper appeared in the *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA'05), Vancouver, British Columbia, Canada, 2005 and in the *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science* (STACS'10), Nancy, France, 2010.

†Institute of Computer Science, University of Wrocław, 15 Joliot-Curie st. PL 50-383 Wrocław, Poland. Research supported by MNiSW Grants N N206 1723 33, 2007–2010 and N N206 490638, 2010–2011. lje@cs.uni.wroc.pl.

‡Department of Computer Science, George Mason University, Fairfax, VA 22030, USA. Research partially supported by NSF Grant CCF-0915681. lifei@cs.gmu.edu.

§Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027, USA. 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, New York, NY 10027, USA. Research partially supported by NSF Grant CCF-0728733 and CCF-0915681. cliff@ieor.columbia.edu.

an online algorithm, and is typically evaluated by its *competitive ratio*. 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* [8]. Our goal is to design an online algorithm for this scheduling problem with a good competitive ratio. If an online 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*. All algorithms we design in this paper are memoryless.

1.1. Prior Work. Since the introduction of this online buffer management model in [20, 21], many researchers have studied this problem as well as several variants. For the general problem, a simple greedy algorithm that schedules a maximum-weight packet in the buffer is 2-competitive [18, 20, 21]. The best-known lower bound on the competitive ratio of deterministic algorithms, however, is $\phi \approx 1.618$ [18, 10, 2]. Much research has been motivated by attempting to design algorithms with improved competitive ratios, and also by considering natural restrictions on problem instances that make an improved competitive ratio possible. Two such 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 *similarly 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.

A generalization of the greedy algorithm, called EDF_α, always schedules the earliest-deadline packet with weight at least $1/\alpha$ ($\alpha \geq 1$) of the maximum-weight packet [6, 9]. 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. Chrobak et al. [11, 12] introduced the idea of alternating between the maximum-weight packet and an earliest-deadline packet with sufficiently large weight. As stated, this idea does not result in an improvement, but they design a clever modification that improves the competitive ratio to $64/33 \approx 1.939$. Their algorithm is the first one with a competitive ratio strictly below 2 for the general case. The best currently known deterministic algorithm has competitive ratio $2\sqrt{2} - 1 \approx 1.828$ [15]. For 2-uniform instances, Chrobak et al. [11, 12] designed an algorithm that is 1.377-competitive and proved a matching lower bound. Their algorithm uses information about the past, and so is not memoryless. In fact, a tight lower bound of $\sqrt{2} \approx 1.414$ has been proved on the competitive ratio of memoryless algorithms for 2-uniform instances [2, 6, 9].

Randomized algorithms (against oblivious adversary) have also been given [6, 9] with competitive ratios of $e/(e-1) \approx 1.582$ for the general case, 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. For the general case, 1.25 is still the best known lower bound for randomized algorithms. The adaptive-online adversary model has been recently studied, for which an upper and lower bound of 4/3 has been given for 2-bounded instances [7]. It turns out that the randomized algorithm by Bartal et al. [6, 9] remains $e/(e-1)$-competitive against an adaptive adversary, even though subtle changes in its analysis are required [19].

Finally, we mention a large body of work on a closely related model where packets do not have deadlines, the buffer capacity is finite, and packets must be sent in a FIFO

manner [24, 25, 22, 5, 26, 14, 16]. Note that a c-competitive algorithm for the FIFO buffer model in which the buffer size is s implies a c-competitive algorithm for s-uniform bounded delay instances. Some researchers also consider packet scheduling in multiple FIFO input queues connecting one output queue [3, 4, 1]. We refer to a survey by Goldwasser [17] for an overview of results and techniques for both models and some of their extensions.

The known bounds are summarized in Table 1.1. Our results in this paper are marked with [$*$]. A blank entry indicates that the bound in this entry follows from another bound in the same column.

	Deterministic		Randomized	
	Upper bounds	Lower bounds	Upper bounds	Lower bounds
General	2 [18, 20, 21] 1.939 [11, 12] 1.854 [23] 1.828 [15]		1.582 [6, 9]	
s-bounded	$2 - 2/s + o(1/s)$ [6, 9] 1.854 [23] 1.828 [15]			
agreeable-deadline	1.838 [11, 12] 1.618 [$*$]		1.33 [$*$]	
2-bounded	1.618 [20, 21] [$*$]	1.618 [18, 10, 2]	1.25 [6, 9]	1.25 [10]
s-uniform	1.732 [14, 16] 1.618 [$*$]			1.25 [6, 9] ($s \to \infty$)
2-uniform	1.414 [2] 1.377 [11, 12]	1.377 [11, 12]		1.172 [6, 9]

1.2. Our Contribution. Our main contribution includes the design of two online algorithms for the agreeable-deadline packet scheduling problem described earlier. The first algorithm, called MG (for *modified greedy*), is deterministic and achieves a competitive ratio ϕ. This is the best possible competitive ratio for any deterministic online algorithm, as the instances in the lower bound proof [18, 10, 2] are 2-bounded (which is a special case of agreeable deadlines). The algorithm is strikingly simple, and its analysis is simpler than those of previous algorithms with competitive ratios below 2.

Besides the improved competitive ratio, we believe that our method of analysis makes an important contribution. With any online algorithm, one needs to compare the performance of the algorithm to the performance of an offline optimal algorithm. When analyzing such algorithms step-by-step, one immediately encounters a challenge, since at some point the online algorithm will make a different choice than the offline optimal, leaving them with different buffers. It then can become difficult to compare the two algorithms going forward, as they have different "states." The standard method of dealing with this difficulty is to introduce a potential function which implicitly compensates for the difference in states, measuring how far ahead or behind the online algorithm is. We take a different approach. After each step of the algorithm, for the purposes of analysis, we explicitly modify the buffer of the optimal offline algorithm. We must be careful to do so in a way that can only help the adversary, by either giving him a "better" set of packets, or by increasing his objective function to compensate for any loss of packets.

Our second algorithm, called RMG, is randomized and achieves a competitive ratio $4/3 \approx 1.33$ against an oblivious adversary. The analysis of RMG is similar to the analysis of MG, in that it also explicitly modifies that adversary's buffer.

2. The Agreeable Deadline Model.

2.1. Motivation. For motivation, we consider the greedy algorithm, which sends the maximum-weight packet among all the packets in the buffer that can still meet their deadlines [18, 20, 21]. Fix an $\epsilon > 0$ and consider the two packets $(1-\epsilon,\ 1)$ and $(1,\ 2)$, where packet $(w,\ d)$ represents a packet with weight w and deadline d. The greedy algorithm sends the $(1,\ 2)$ packet in the first slot, after which the $(1-\epsilon,\ 1)$ packet expires. The optimal algorithm, however, sends both packets. Thus we have a class of instances on which the greedy algorithm's competitive ratio can be arbitrarily close to 2. The key limitation of the greedy algorithm is that it considers only the weights of the packets but not the deadlines.

A natural modification to the greedy algorithm is to consider packets with a "sufficiently large" weight, and pick the earliest-deadline packet among those [6, 9]: For instance, pick an earliest-deadline packet with weight at least α times the maximum-weight, for some $\alpha \in (0,\ 1)$. The algorithm that selects the earliest-deadline packet to send is called EDF, and the algorithm selects the earliest-deadline packet with weight at least α times the maximum-weight is called EDF_α [6, 9]. Note that the greedy algorithm is EDF_1, whereas EDF is EDF_0.

Unfortunately, EDF_α for all $\alpha > 0$ runs into the same difficulty as can be seen in the following example. Fix some large integer t_0 and consider the following set of packets, all released at the very beginning. The set of packets consists of three groups. The first group consists of a single packet $(1,\ 2t_0+2)$, i.e., a packet with weight 1 that expires after step $2t_0+1$, while the two remaining groups have t_0 packets each. The packets of the second group all have weights $\alpha-\epsilon$ and deadlines $2,\ 3,\ \ldots,\ t_0+1$ respectively, while those of the third group all have weights α and deadlines $t_0+2,\ t_0+3,\ \ldots,\ 2t_0+1$ respectively. Note that all the packets can be sent in an EDF manner, but EDF_α only manages to send the packets from the first and third group. As ϵ can be arbitrarily small while t_0 arbitrarily large, EDF_α's competitive ratio is no better than 2.

On the other hand EDF is not competitive at all as can be seen in the next example. Fix some large integer t_0 and consider the following set of packets, all released at the very beginning. The first group is formed by t_0 packets of weight ϵ (arbitrarily small) and deadlines $2,\ 3,\ \ldots,\ t_0+1$, while the second group is formed by t_0 packets $(1,\ t_0+1)$, i.e., packets of weight 1 that expire only after step t_0. EDF sends the worthless packets from the first group in every step, with the possible exception of t_0, whereas it is possible to send a packet of weight 1 from the second group in every step up to t_0. As ϵ can be arbitrarily small while t_0 is arbitrarily large, EDF's competitive ratio is unbounded.

It can be noted that in the second example EDF has more than t_0 packets of weight 1 to choose from right in the beginning, while all the pending packets expire after step t_0 at the latest. Therefore it makes little sense to send a very light earliest-deadline packet while heavy packets abound. The solution then is to first identify the packets that it makes sense to send, and send either the earliest-deadline one or the maximum-weight one of them, depending on the ratio of their weights. This is the idea underlying the algorithm MG.

2.2. Algorithm MG. We start with some definitions.

DEFINITION 2.1. **Provisional schedule** *[13, 15]. Given a set of pending packets P, a* provisional schedule *S specifies which packet in P should be sent in which time step.*

DEFINITION 2.2. **Optimal provisional schedule** *[13, 15]. Given a set of pending packets P, an* optimal provisional schedule *S^* is one that achieves the maximum weighted throughput among all provisional schedules on pending packets P.*

Clearly, we can calculate the *optimal provisional schedule* S^* at time t by finding a maximum-weight bipartite matching over pending packets (or by simpler combinatorial algorithms.) There may be many optimal provisional schedules. To simplify the analysis, we pick one such schedule, defined in terms of the following *dominance relation* among the packets.

DEFINITION 2.3. **Dominance relation**. *For any two packets p and q pending at a certain time, p is said to dominate q if either $w_p > w_q$ and $d_p \leq d_q$, or $w_p \geq w_q$ and $d_p < d_q$.*

Given the set S^* of packets in an optimal provisional schedule, assign to every step the earliest-deadline non-dominated packet from S^* that has not yet been assigned. The schedule thus obtained is the *canonical schedule*. Note that in the canonical schedule, the transmitted packets are arranged in (weakly) increasing deadline order, with ties broken in favor of heavier packets.

Algorithm MG is now easy to describe; its pseudocode is given in Algorithm 1.

Algorithm 1 MG.

1: In the time step t, find the (canonical) optimal provisional schedule S^* from the pending packets in the buffer (including the new arrivals) at time t.
2: Let
- e denote the first (i.e. earliest-deadline non-dominated) packet in the buffer; (Note that e has the maximum weight among all earliest deadline packets, as none of them dominates it.)
- h denote the first maximum-weight (i.e. heaviest non-dominated) packet in the buffer. (Note that h has the earliest deadline among all maximum-weight packets in the buffer, as none of them dominates it.)

3: **if** $w_e \geq w_h/\phi$ **then**
4: send e;
5: **else**
6: send h.
7: **end if**

2.3. Analysis of MG. As described above, a key contribution of our work is to avoid the use of a potential function. The potential function approach is a commonly used method in analyzing online algorithms [8]. However, in our analysis, we do not use the potential function argument explicitly. Instead, our analysis relies on modifying the adversary's buffer judiciously in each step and on assigning an appropriate credit to the adversary to account for these modifications. Then, we bound the competitive ratio by comparing the modified adversary's gain to that of MG's in each step.

Suppose that at some time t, MG and the adversary have identical buffers. Both MG and the adversary will each process arriving packets and then send a packet, but at the end of the time step, their buffer contents may be different. We then modify the adversary's buffer and make it identical to the algorithm's buffer; but to do so,

we may have to let the adversary collect additional weight. The crux of the analysis is to show that the algorithm's gain in this time step is at least $1/\phi$ of the adversary's *modified gain*, which is the sum of the weight of the packet that the adversary sent and the additional weight given to the adversary in modifying its buffer. At the beginning of step $t+1$, both the algorithm and the (modified) adversary will start with identical buffers; the result follows by a simple inductive argument.

Let $\mathcal{O}_t$ denote the set of packets that the (oblivious) adversary is supposed to send in steps $t,\ t+1,\ \ldots$; note that some of them may be released after step t. Then a standard exchange argument yields,

REMARK 1. *The heaviest non-dominated packet h, without loss of generality, belongs to $\mathcal{O}_t$.*

This and the next observation prove useful in the analysis.

REMARK 2. *Increasing the weight or deadline of any pending packet, or sending an additional packet in a single step are advantageous to the adversary, i.e., can only increase the total weight of $\mathcal{O}_t$.*

Without loss of generality, we make the following assumptions on $\mathcal{O}_t$.

1. $\mathcal{O}_t$ is optimal, i.e., it has maximum total weight.
2. For each $j \in \mathcal{O}_t$, either $r_j \geq t$ or j is in the optimal provisional schedule S^*.
3. The adversary sends its packets in the canonical order, always sending the earliest non-dominated packet from S^*.

Together with Remark 1, this yields the following lemma.

LEMMA 2.4. *For every step t there exists an optimum adversary schedule that schedules either e or h first, where these denote, respectively, the earliest-deadline and the heaviest non-dominated packet.*

Proof. Pick any optimum adversary schedule in canonical order. If it contains e, then e comes first by the canonical ordering and we are done.

Suppose then that the schedule does not contain e. But it does, without loss of generality, contain h by Remark 1. We argue that the schedule can be reordered so that h is sent now, regardless of the future arrivals.

For convenience, let $p_1,\ p_2,\ \ldots$ be the packets in the buffer; note that $e = p_1$ and $h = p_l$ for some $l > 1$. Since the packets are all schedulable in the absence of future arrivals, $d_{p_i} \geq t + i$. A packet p_i is said to be *critical* if $d_{p_i} = t + i$. The adversary does not schedule $e = p_1$, but it does schedule $h = p_l$, and possibly some packets that appear in between. Since $d_{p_i} \geq t + i$, and as the deadline of every future arrival is at least d_h, none of the packets $p_{i_1},\ p_{i_2},\ \ldots,\ p_{i_j},\ p_l$ in the adversary's current schedule is critical. So the sequence $p_l,\ p_{i_1},\ p_{i_2},\ \ldots,\ p_{i_j}$ (with no reordering after this prefix) is a valid schedule for the adversary. □

THEOREM 2.5. *MG is ϕ-competitive for packet scheduling with agreeable deadlines.*

Proof. Fix a time t, and suppose MG and ADV have identical buffers. After processing the arrivals at time t, the buffer contents of MG and ADV remain the same.

The proof proceeds by considering various cases, depending on which packet MG and ADV send. Recall that due to Lemma 2.4 and algorithm's definition, each of MG and ADV sends either e or h. Let v_{ADV} and v_{MG} denote the values collected by the (modified) adversary and the algorithm at time t. We prove that for each case in one time step, $v_{\mathrm{ADV}} \leq \phi \cdot v_{\mathrm{MG}}$.

1. Suppose ADV and MG send the same packet j; (Note: this includes the case

$e = h$.)

$$v_{\mathrm{ADV}} = v_{\mathrm{MG}} = w_j.$$

ADV and MG's buffers are identical at the end of step t.

2. Suppose MG sends e, and ADV sends packet h.
   As MG sends e, $w_e \geq w_h/\phi$, and so we have

$$v_{\mathrm{ADV}} = w_h \leq \phi \cdot w_e = \phi \cdot v_{\mathrm{MG}}.$$

   Modify the adversary's buffer by replacing e with the packet h; this only helps the adversary (see Remark 2) as $d_e \leq d_h$ and $w_e \leq w_h$ hold by the definitions of e and h, and the fact that they do not dominate one another. After this modification, both ADV and MG have identical buffers, and $v_{\mathrm{ADV}}/v_{\mathrm{MG}} \leq \phi$, as required.
3. Suppose MG sends h and ADV sends e.
   Note that $w_h > \phi \cdot w_e$, and $d_h > d_e$. We let the modified ADV send both e and h in this time step and keep e in its buffer. This modification can only help ADV, and its buffer after the modification is identical to MG's buffer.
   Now, $v_{\mathrm{ADV}} = w_e + w_h$, and $v_{\mathrm{MG}} = w_h$. As $w_h > \phi \cdot w_e$,

$$\frac{v_{\mathrm{ADV}}}{v_{\mathrm{MG}}} = 1 + \frac{w_e}{w_h} < 1 + \frac{1}{\phi} = \phi.$$

These 3 cases cover all the possible actions of the algorithm and their consequences. ∎

2.4. Algorithm RMG. The randomized algorithm identifies the e and h packets just like MG does. But when it comes to decide whether e or h should be transmitted, it chooses e with probability w_e/w_h and h with the remaining probability. Its pseudocode is given in Algorithm 2.

Algorithm 2 RMG.

1: In the time step t, find the (canonical) optimal provisional schedule S^* from the pending packets in the buffer (including the new arrivals) at time t.
2: Let
- e denote the first (i.e. earliest-deadline non-dominated) packet in the buffer.
- h denote the first maximum-weight (i.e. heaviest non-dominated) packet in the buffer.

3: Send e with probability w_e/w_h and h with the remaining probability.

THEOREM 2.6. *RMG is 4/3-competitive for packet scheduling with agreeable deadlines against an oblivious adversary.*

Proof. The analysis is very similar to that of the analysis of MG. Again we note that, by Lemma 2.4, ADV transmits either e or h. Let v_{ADV} and v_{RMG} denote the expected values collected by the (modified) adversary and the algorithm at time t. We prove that

$$v_{\mathrm{ADV}} = w_h, \tag{2.1}$$

$$v_{\mathrm{RMG}} \geq \frac{3}{4} w_h, \tag{2.2}$$

which yields the result immediately.

To bound the first value we consider two cases.

1. Suppose ADV sends h.
   In this case, $v_{\mathrm{ADV}} = w_h$, because we never let ADV send any additional packet. Instead, as in the previous analysis, if RMG sends e, we replace e in ADV's buffer by h to keep the buffers of RMG and ADV identical.
2. Suppose ADV sends e.
   In this case, just like in the previous analysis, to keep the buffers of RMG and ADV identical, we let ADV send h as well and keep e in its buffer if RMG sends h. Therefore
   $$v_{\mathrm{ADV}} = w_e + \left(1 - \frac{w_e}{w_h}\right) w_h = w_h.$$

This concludes the proof of (2.1).

For the second bound, we note that v_{RMG} is simply the expected gain of RMG in the step. Thus, by using the definition of expected gain, rearranging terms and lower bounding the sum of two positive terms by one of the terms, we obtain

$$v_{\mathrm{RMG}} = \frac{w_e}{w_h} \cdot w_e + \left(1 - \frac{w_e}{w_h}\right) \cdot w_h = \frac{1}{w_h}\left(\left(w_e - \frac{w_h}{2}\right)^2 + \frac{3}{4}w_h^2\right) \geq \frac{3}{4}w_h.$$

which concludes the proof of (2.2) and the theorem. □

3. Conclusion. In this paper, we design online algorithms for buffer management for agreeable-deadline packets. Our contributions include an optimal 1.618-competitive deterministic algorithm MG and a 1.33-competitive randomized algorithm RMG (against an oblivious adversary). In analyzing these algorithms, we introduce a new analysis method, which does not rely on a potential function approach explicitly. Instead, it modifies the adversary's buffer to make it identical to the algorithm's, and assigns an appropriate credit to the adversary to account for these modifications. We expect this approach to have further applications; there is already one such example [19]. The very same example highlights the following feature of our analysis: The framework works for general instances, for both deterministic and randomized algorithms, in both oblivious and adaptive adversary model. Indeed, we use the assumptions that the instance is fixed and has agreeable deadlines only to exploit the structure of the adversary's schedule, which in turn makes it possible to obtain better competitive ratios in the analysis framework.

Note that for randomized algorithms against an oblivious adversary, there is still a gap between the 5/4 lower bound and our 4/3 upper bound. For an adaptive adversary the analogous gap is between the 4/3 lower bound and the $e/(e-1)$ upper bound, which actually holds for the general case, not just for the agreeable deadline instances. Incidentally, on 2-bounded instances our randomized algorithm RMG coincides with an optimal algorithm against an adaptive adversary [7]. However, we were unable to extend our analysis to the adaptive adversary model due to the random nature of such an adversary's schedule. We believe that shrinking either of these gaps is an interesting open problem.

REFERENCES

[1] S. Albers and M. Schmidt, *On the performance of greedy algorithms in packet buffering*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), 2004, pp. 35–44.

[2] N. Andelman, Y. Mansour, and A. Zhu, *Competitive queuing polices for QoS switches*, in Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2003, pp. 761–770.

[3] Y. Azar and Y. Richter, *Management of multi-queue switches in QoS networks*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), 2003, pp. 82–89.

[4] ———, *The zero-one principle for switching networks*, in Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC), 2004, pp. 64–71.

[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), 2004, pp. 196–207.

[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), 2004, pp. 190–210.

[7] M. Bieńkowski, M. Chrobak, and Ł. Jeż, *Randomized algorithms for buffer management with 2-bounded delay*, in Proceedings of the 6th Workshop on Approximation and Online Algorithms, 2008, pp. 92–104.

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

[9] F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, J. Sgall, and T. Tichy, *Online competitive algorithms for maximizing weighted throughput of unit jobs*, Journal of Discrete Algorithms, 4 (2006), pp. 255–276.

[10] F. Y. L. Chin and S. P. Y. Fung, *Online scheduling with partial job values: Does timesharing or randomization help?*, Algorithmica, 37 (2003), pp. 149–164.

[11] M. Chrobak, W. Jawor, J. Sgall, and T. Tichy, *Improved online algorithms for buffer management in QoS switches*, in Proceedings of the 12th Annual European Symposium on Algorithms (ESA), 2004, pp. 204–215.

[12] ———, *Improved online algorithms for buffer management in QoS switches*, ACM Transactions on Algorithms, 3 (2007), p. Article No. 50.

[13] ———, *Online scheduling of equal-length jobs: Randomization and restart help?*, SIAM Journal on Computing (SICOMP), 36 (2007), pp. 1709–1728.

[14] 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), 2006, pp. 352–363.

[15] ———, *Considering suppressed packets improves buffer management in QoS switches*, in Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007, pp. 209–218.

[16] ———, *Lower and upper bounds on FIFO buffer management in QoS switches*, Algorithmica, 53 (2009), pp. 523–548.

[17] M. H. Goldwasser, *A survey of buffer management policies for packet switches*, SIGACT News, 41 (2010), pp. 100–128.

[18] 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), 2001, pp. 434–438.

[19] Ł. Jeż, *Randomised buffer management with bounded delay against adaptive adversary*, CoRR, abs/0907.2050 (2009).

[20] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko, *Buffer overflow management in QoS switches*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC), 2001, pp. 520–529.

[21] ———, *Buffer overflow management in QoS switches*, SIAM Journal of Computing (SICOMP), 33 (2004), pp. 563–583.

[22] A. Kesselman, Y. Mansour, and R. van Stee, *Improved competitive guarantees for QoS buffering*, Algorithmica, 43 (2005), pp. 63–80.

[23] F. Li, J. Sethuraman, and C. Stein, *Better online buffer management*, in Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007, pp. 199–208.

[24] Z. Lotker and B. Patt-Shamir, *Nearly optimal FIFO buffer management for DiffServ*, in Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC), 2002, pp. 134–142.

[25] ———, *Nearly optimal FIFO buffer management for two packet classes*, Computer Networks, 42 (2003), pp. 481–492.

[26] M. Schmidt, *Packet buffering: Randomization beats deterministic algorithms*, in Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS), 2005, pp. 293–304.