Journal of Network and Computer Applications 74 (2016) 31–43

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Large profits or fast gains: A dilemma in maximizing throughput with applications to network processors

Kirill Kogan [a], Alejandro López-Ortiz [b], Sergey I. Nikolenko [c,d], Gabriel Scalosub [e], Michael Segal [e,*]

[a] IMDEA Networks Institute, Madrid, Spain
[b] School of Computer Science, University of Waterloo, Canada
[c] National Research University Higher School of Economics, St. Petersburg, Russia
[d] Steklov Mathematical Institute, St. Petersburg, Russia
[e] Department of Communication Systems Engineering, Ben-Gurion University of the Negev, Israel

ARTICLE INFO

Article history:
Received 5 June 2015
Received in revised form
9 March 2016
Accepted 29 July 2016
Available online 5 August 2016

Keywords:
Online buffer management
Online scheduling

ABSTRACT

We consider the fundamental problem of managing a bounded size queue buffer where traffic consists of packets of varying size, each packet requires several rounds of processing before it can be transmitted out, and the goal is to maximize the throughput, i.e., total size of successfully transmitted packets. Our work addresses the tension between two conflicting algorithmic approaches: favoring packets with fewer processing requirements as opposed to packets of larger size. We present a novel model for studying such systems and study the performance of online algorithms that aim to maximize throughput.

1. Introduction

Over the recent years, there has been a growing interest in understanding the effects that buffer sizing has on network performance. The main motivation for these studies is to understand the interplay between buffer size, throughput, and queueing delay. Broadly speaking, one can identify three main types of delay that contribute to packet latency: transmission and propagation delay, processing delay, and queueing delay. Recent research that advocates the usage of small buffers in core routers, aiming to reduce queueing delay in the presence of (mostly) TCP traffic, sidesteps the issue that as buffers get smaller, the effect of processing delay becomes much more pronounced (Ramaswamy et al., 2009). The importance of these phenomena is further emphasized by increasing heterogeneity of network traffic processing. The modern network edge is required to perform tasks with ever-increasing complexity including features such as advanced VPNs services, deep packet inspection, firewall, intrusion detection etc. Each of these features may require a different processing effort at the routers (Wolf and Franklin, 2000), and such features directly affect processing delay. As a result, the processing method of packets and the way how these packets are processed ("run-for-completion", processing with preemptions, etc.) may have significant impact on queueing delay and throughput; increasing the required processing per packet in some of the flows may cause increased congestion even for traffic with relatively modest burstiness characteristics.

We should note that in the general case, processing requirements are independent of packet lengths, thus decoupling the amount of work required for a router to process a packet from the throughput gained upon its successful transmission. Processing requirement and packet length are indeed two independent characteristics in the modern networks: a short packet may require complex processing policies while a long packet may simply go through almost untouched and vice versa. This independence lets us design processing policies better suited for different objective functions (e.g., by optimizing bytes transmitted per processing cycle).

This situation leads to several questions relevant to the design and implementation of router architectures. For instance, in light of heterogeneous processing requirements in the traffic, does one need to implement input buffering before a packet is handled by the network processor? If so, what should the size of such a buffer be, and what admission control policy should be applied? Another question is related to adapting common active queue management (AQM) policies so that they account for heterogeneous processing required by traffic. In this respect, the main question is whether current AQM approaches are capable of considering these characteristics; if not, what form should new policies take? In this

* Corresponding author.
E-mail addresses: kirill.kogan@imdea.org (K. Kogan), alopez-o@uwaterloo.ca (A. López-Ortiz), sergey@logic.pdmi.ras.ru (S.I. Nikolenko), sgabriel@bgu.ac.il (G. Scalosub), segal@bgu.ac.il (M. Segal).

http://dx.doi.org/10.1016/j.jnca.2016.07.010
1084-8045/

work, we initiate the study of these questions and the tradeoffs they encompass. We focus on improving our understanding of effects that processing disciplines have on throughput in cases of bounded buffers where traffic is heterogeneous in terms of both packet processing requirements and packet length.

In what follows, we adopt the terminology used to describe queue management within a router in a packet-switched network. We focus our attention on a general model for the problem where we are required to manage the admission control and scheduling units in a single bounded size queue, where arriving traffic consists of *packets*, such that each packet is labeled with its *size* (e.g., in bytes), and *processing requirement* (in processor cycles). A packet is successfully *transmitted* once the scheduling unit has scheduled the packet for processing for at least its required number of cycles, while the packet resides in the buffer. If a packet is dropped from the buffer, either upon arrival due to admission control policies or after being admitted and possibly partially, but not fully, processed (in scenarios where push-out is allowed), such a packet is irrevocably lost. We focus our attention on maximizing the *throughput* of the queue, measured by the total number of bytes of packets that are successfully transmitted by the queue.

2. Our contributions

In this work we provide a formal model for studying problems of online buffer management and online scheduling in settings where packets have both varying size and heterogeneous processing requirements, and one has a limited size buffer to store arriving packets. Our model lets us study the interplay between potentially conflicting approaches, favouring large packets and favouring packets with less required processing, in the situation where the goal is to maximize the total length of transmitted packets. At least at this point, it is unclear which one is more significant: if one policy processes longest packets first while another processes packets with minimal residual work, which is better? The offline version of this problem is NP-hard, as it encompasses Knapsack as a special case. For the more natural online setting, we propose algorithms with provable worst case performance guarantees that greedily optimize each one of these characteristics (or a combination of the both). We focus our attention on priority-based buffer management and scheduling in both *push-out* (PO) settings, where admitted packets are allowed to be pushed out of the queue prior to having its processing completed (in which case the packet does not contribute to the system's throughput), and in the *non-push-out* (NPO) case, where buffer management decisions are limited to admission control.

Specifically, we consider the following priority queueing regimes: (i) Shortest-Remaining-Processing-Time (SRPT) first, common in job scheduling environments; (ii) Longest-Packet (LP) first; (iii) Most-Effective-Packet (MEP) first, prioritizing packets by the ratio of residual processing requirement to size. We study buffer management algorithms for these priorities, proving bounds in terms of (i) maximum packet size and (ii) maximum number of processing cycles per packet. In the push-out case, to reduce the number of combinations considered, the same characteristic defines both processing order and push-out mechanism. Our results are summarized in Table 1.

3. Related work

In recent years, there has been a surge in the study of the effects of buffer size on traffic queueing delays arising in networking systems. Appenzeller et al. (2004) studied this problem in the context of statistical multiplexing, focusing mostly on TCP flows.

Table 1
Results summary: lower and upper bounds on the competitive ratio.

Algorithm	Lower bound	Upper bound
NPO, any priority	kL	$k(L+1)$
PO, SRPT priority	L	$2L$
PO, LP priority	k	$k+3$
PO, MEP priority	$\frac{9}{16}\min\{k, L\}$	–

More recently, broader aspects of these question were studied, and a comprehensive overview of perspectives on router buffer sizing can be found in Vishwanath et al. (2009).

The works (Keslassy and Kogan, 2012; Kogan et al., 2012, 2013; Azar and Gilon, 2015) considered buffer management and scheduling in the context of network processors, where arriving traffic has heterogeneous processing requirements for unit-sized packets. They study schedulers with various processing orders in both push-out and non-push-out buffer management regimes. They focused on the case where packets are of unit size and showed competitive algorithms, as well as lower bounds, for such settings. In particular, an algorithm with logarithmic competitiveness was introduced in Kogan et al. (2016) and, further, a 2-competitive algorithm was proposed in Kogan et al. (2012). Also recently, Azar and Gilon (2015) have considered unit-sized packets with heterogeneous processing requirements and proposed a different 2-competitive algorithm than (Kogan et al., 2012) for a single queue architecture with FIFO processing order and allowed push-outs.

We believe that the assumption made in Keslassy and Kogan (2012) that packets are of unit size is rather restrictive, since in real life NPs have to deal with packets of varying size, and it is unclear how one should design algorithms that ensure good throughput guarantees in such highly heterogeneous scenarios.

In the conference version of this work (Kogan et al., 2014), we stated that the PO policy with MEP-based priorities could have constant competitiveness in some cases; in particular, this policy is $\left(1+\frac{L\ln k-k-L}{B}\right)$-competitive. Here, we will show that in general the PO policy with MEP-based priority is at least $\left(\frac{9}{16}\min\{k, L\}\right)$-competitive, so it is no better than the two other priorities considered in Kogan et al. (2014) in the worst case.

Our current work can be viewed as part of a larger research effort that focuses on studying competitive algorithms for buffer management and scheduling, and specifically the study of such algorithms in bounded-buffers settings (see, e.g., a recent survey by Nikolenko and Kogan (2015) which provides an overview of the latest results). Recently Chuprikov et al. (2015) considered a weighted throughput optimization for unit-sized packets with heterogeneous values and processing requirements.

The SRPT algorithm has been studied extensively in OS scheduling for multithreaded processors, and it is well known to be optimal for mean response (Schrage, 1968). However, the SRPT algorithm as it is understood in literature is not the same as we study in this work: in the context of job scheduling (Schrage, 1968), SRPT is assumed to have an unlimited buffer and does not allow for push-out, while we use a limited buffer with push-out.

Additional objectives, models, and algorithms have been studied extensively in this context; see, e.g., Leonardi and Raz (1997), Muthukrishnan et al. (2005), Rajeev Motwani (1994), Kesselman and Kogan (2007), Kesselman et al. (2012a,b, 2010), and Kesselman (2013). A comprehensive overview of competitive online scheduling for server systems can be found in Pruhs (2007); however, OS scheduling is mostly concerned with average response time and average slowdown, while we focus on providing worst-case guarantees on the throughput. Furthermore, OS scheduling does

not allow for dropping jobs, which is an inherent aspect of our model, as implied by the fact we have a limited-size buffer, and overflowing packets must be dropped. The model considered in our work is also closely related to Job-shop scheduling problems (Brucker et al., 2006), most notably to hybrid flow-shop scheduling (Ruiz et al., 2010), in scenarios where machines have bounded buffers. However, while these works focus on system delay, our main focus is system throughput.

Recently, Sivaraman et al. (2015) explored what should be flexible to express buffer management policies for different objectives and suggested the usage of a single queue buffering architecture.

4. Model description and algorithmic framework

Consider a buffer with bounded capacity of B bytes handling the arrival of a sequence of packets. Each arriving packet p has a size $\ell(p) \in \{1, \ldots, L\}$ (in bytes) and a number of required processing cycles $r(p) \in \{1, \ldots, k\}$; both $\ell(p)$ and $r(p)$ are known for every arriving p. Note that required processing characteristics on a network processor are often highly regular and predictable for a fixed configuration of network elements (Wolf et al., 2003), so per-packet processing requirements are expected to be available and well-defined as a function of the features associated with the flow and the network element configuration. Our assumption that the size may be as small as one byte is made for simplicity and can be viewed as a scaling assumption. The values of maximal required processing k and maximal size L will play a fundamental role in our analysis; however, that none of our algorithms need to know k in advance.

The queue performs two main tasks: *buffer management*, i.e., admission control of new packets and push-out of currently stored packets, and *scheduling*, i.e., which of the currently stored packets are scheduled for processing. The scheduler will be determined by the *priority policy* employed by the queue. We assume a multi-core environment with C processors, so that at most C packets may be assigned for processing in any given time. Below we assume $C=1$; this setting suffices to show both the intrinsic difficulties of the model and our algorithmic scheme. We assume slotted time, where each time slot t consists of 3 phases: (i) *transmission*, when packets with zero remaining required processing leave the queue; (ii) *arrival*, when new packets arrive, and the buffer management unit performs both admission-control and possibly push-out; (iii) *assignment and processing*, when a single packet is assigned for processing by the scheduling unit. Fig. 1 depicts our general model. If a packet is dropped prior to being *transmitted* (i.e., while it still has a positive number of required processing cycles), it is lost; we can drop a packet either upon arrival or due to a push-out decision while it is in the buffer. A packet contributes its size to the objective function only upon being successfully transmitted. The goal of a buffer management algorithm is to maximize the overall throughput, i.e., total number of bytes transmitted.

We define a *greedy* buffer management policy as a policy that accepts all arrivals whenever there is available buffer space in the queue. We only consider *work-conserving* schedulers, i.e. schedulers that never leave the processor idle unnecessarily. An arriving packet p *pushes out* a packet q that has already been accepted into the buffer iff q is dropped in order to free up buffer space for p and p is admitted to the buffer instead. A buffer management policy is called *push-out* (PO) if it allows packets to push out currently stored packets and *non-push-out* (NPO) if it does not. For an algorithm ALG and a time slot t, we define IB_t^{ALG} as the set of packets stored in ALG's buffer at time t. The number of *processing cycles* of a packet is key to our algorithms. For a time

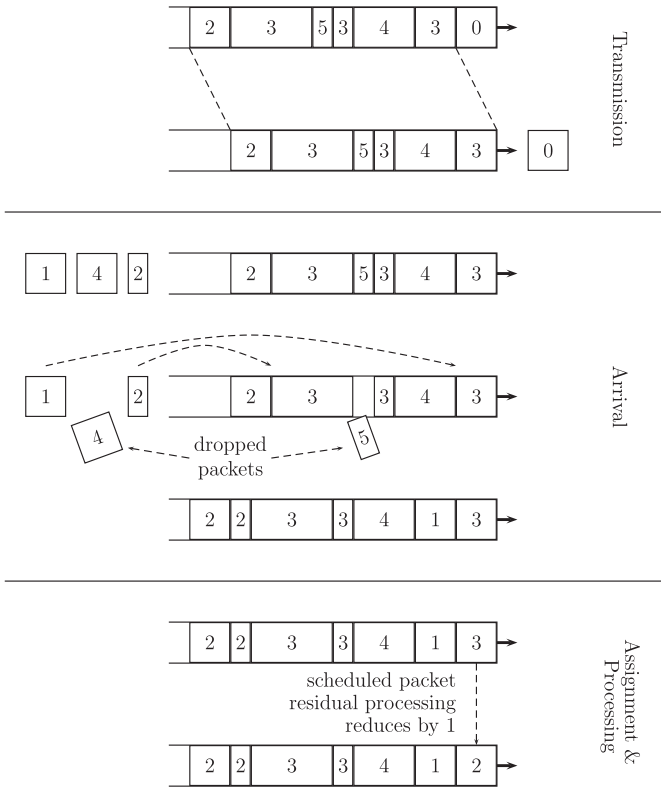


Fig. 1. An outline of the model. The top subfigure shows the transmission phase, the middle subfigure shows the arrival phase where packets might be discarded, and the bottom subfigure shows the assignment and processing phase. The length of a packet represents its size, and the number stamped on the packet represents the number of its (residual) required processing cycles.

moment t and a packet p currently stored in the queue, its number of *residual processing cycles* $r_t(p)$ is defined to be the number of processing cycles it requires before it can be successfully transmitted.

We focus our attention on *priority queueing* disciplines that define both scheduling and buffer management behavior of the queue. Specifically, we employ three disciplines that prioritize the following parameters: (i) *processing* (SRPT): the packet with the least number of residual cycles has top priority; (ii) *length* (LP): the largest packet has top priority; (iii) *processing-to-length* (MEP): the packet with the least residual cycles to size ratio has top priority.

We use competitive analysis (Sleator and Tarjan, 1985; Borodin and El-Yaniv, 1998) to evaluate performance guarantees provided by our online algorithms. An algorithm ALG is said to be *α-competitive* (for some $\alpha \geq 1$) if for any arrival sequence σ, the total length of packets successfully transmitted by ALG is at least $1/\alpha$ times the total length of packets successfully delivered by an optimal solution (denoted OPT), obtained by an offline clairvoyant algorithm.

Next we define the algorithms used below for all types of characteristics. The Non-Push-Out Algorithm (NPO) is a simple greedy work-conserving policy that accepts a packet if there is buffer space available. In the push-out case, the PO algorithm is defined in Algorithm 2. Note that PO is somewhat conservative in its use of the buffer. The reason for this will be clear from our results presented in Sections 7 and 9; the $B - 2L + 1$ position specifically was chosen to simplify analysis: it does not affect the worst-case guarantees but leads to simpler proofs of our main results.

We will sometimes use the term *value* to denote the total length of a set of packets, and our analysis will be based on comparing the mapping value obtained by an optimal solution to that of our algorithm. Specifically, we will make use of mappings

between packets transmitted by OPT and by our algorithm such that their respective values differ only by a multiplicative factor; this factor provides a bound on the competitive ratio.

Algorithm 1. NPO(p): buffer management policy.

1: **if** there is space available in the queue **then**
2: accept p
3: **end if**

Algorithm 2. PO(p): buffer management policy.

1: accept p
2: **while** the last packet q in the buffer starts above position $B - 2L + 1$ **do**
3: drop q
4: **end while**

5. Useful properties of ordered multi-sets

To facilitate our proofs, we will make use of properties of ordered (multi-)sets. These notions, as well as properties we show they satisfy, will enable us to compare the performance of our proposed algorithms with the optimal policy possible, for various priority disciplines. In the following, we consider multi-sets of real numbers, where we assume each multi-set is ordered in non-decreasing order. We will refer to such multi-sets as *ordered sets*. For every $1 \le i \le |A|$, we will further refer to element $a_i \in A$ or to $A[i]$ as the ith element in the set A, as induced by the order. Given two ordered sets A, B, we say $A \ge B$, if for every i for which both a_i and b_i exist, $a_i \ge b_i$.

The following lemma, and its corollary, will be a fundamental tool used throughout our analysis.

Lemma 1. *For any two ordered sets A, B satisfying $A \ge B$, and any two real numbers a, b such that $a \ge b$, if (i) $b \le b_{|B|}$ or (ii) $|A| \le |B|$ then the ordered sets $A' = A \cup \{a\}$, $B' = B \cup \{b\}$ satisfy $A' \ge B'$.*

Proof. We will refer to elements in A' and B' as a' and b', respectively. Assume that i and j are the positions of $a \in A'$ and $b \in B'$, respectively. I.e., $a'_i = a$ and $b'_k = b$. We need to show that for every k for which both a'_k and b'_k exist, $a'_k \ge b'_k$. We distinguish between 2 cases:

(a) $i \le j$ (see Fig. 2(a)): For all $k < i$, $a'_k = a_k$, and $b'_k = b_k$, hence by the assumption that $A \ge B$, $a'_k \ge b'_k$. By the assumption that $a \ge b$, and the fact A' and B' are ordered, for every $p \ge i$ and $q < j$ we have $a'_p \ge a'_i \ge b'_j \ge b'_q$. In particular, for every $i \le k < j$ we have $a'_k \ge b'_k$ (by taking $p = q = k$). For $k=j$, since A' and B' are ordered, and since in the current case $i \le j$, we have $a'_j \ge a'_i = a \ge b = b'_j$. For $k > j$ we have $a'_k = a_{k-1} \ge b_{k-1} = b'_k$, where the inequality follows from the assumption that $A \ge B$.

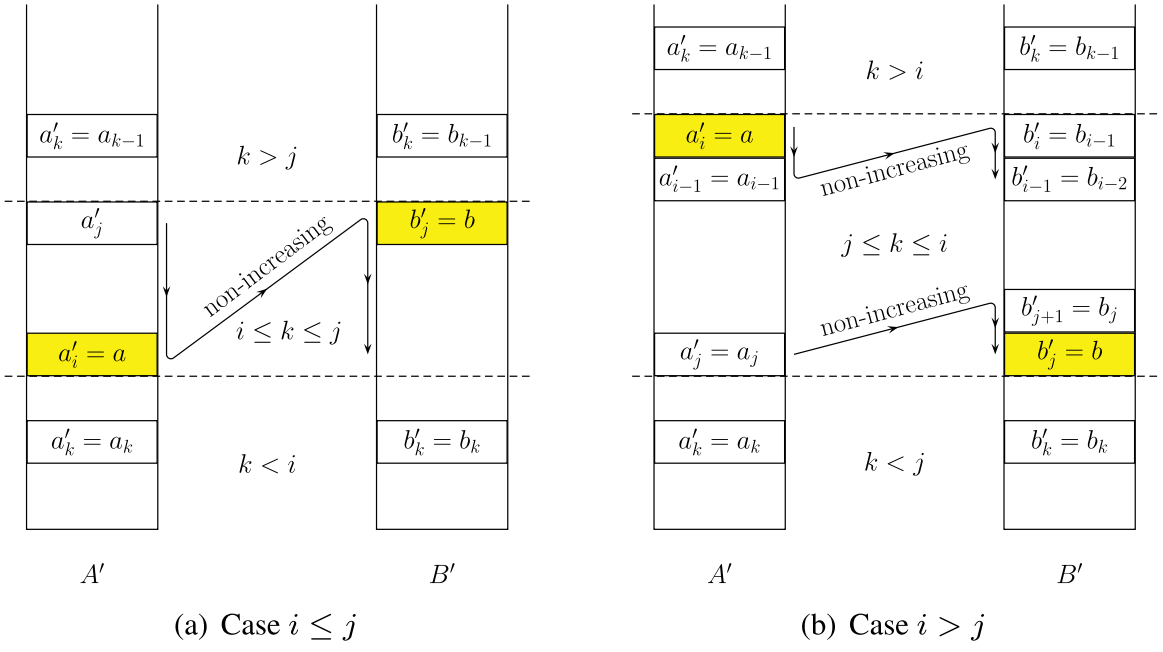
(b) $i > j$ (see Fig. 2(b)): For all $k < j$, $a'_k = a_k$, and $b'_k = b_k$, hence by the assumption that $A \ge B$, $a'_k \ge b'_k$. For $k=j$, $b'_k \le b'_{k+1} = b_k \le a_k = a'_k$, which follows from the fact that b is inserted in slot $j=k$, B' is ordered, the assumption that $A \ge B$ and $b \le b_{|B|}$ or $|A| \le |B|$. For $j < k < i$, $b'_k = b_{k-1} \le a_{k-1} \le a_k = a'_k$, which follows from the assumption that $A \ge B$. For $k=i$, $a = a'_i \ge a_{i-1} \ge b_{i-1} = b'_i$. For $k > i$, $a'_k = a_{k-1} \ge b_{k-1} = b'_k$.

We are therefore guaranteed to have $A' \ge B'$, as required. □

The following corollary shows that the same result holds if we add an item to only one of the sets.

Corollary 2. *For any two ordered sets A, B satisfying $A \ge B$, and any real number b, if (i) $b \le b_{|B|}$ or (ii) $|A| \le |B|$ then the ordered set $B' = B \cup \{b\}$ satisfies $A \ge B'$.*

Proof. Assume that b is inserted in B' in location j. Consider a virtual item a, such that $a > \max\{a_{|A|}, b\}$. We now virtually consider adding both a and b to sets A and B, respectively. By Lemma 1, it follows that the resulting sets A', B' satisfy $A' \ge B'$. Notice that the first $|A|$ elements of A' is exactly the set A (by the choice of a),



(a) Case $i \le j$ (b) Case $i > j$

Fig. 2. Cases of Lemma 1.

implying that we also have $A \geq B'$. □

6. Non-push-out policies

While non-push-out algorithms may have different priorities for the admission policy (which packets to admit from a set of simultaneously arriving packets), they cannot push already admitted packets out. As a result, the worst-case bounds are very similar for all three priorities we consider, and we simply prove a unified lower and upper bound on NPO performance for any admission policy.

Theorem 3. *NPO is at least kL-competitive and at most $k(L+1)$-competitive.*

Proof. We begin with the lower bound. To show a lower bound, we need to present a "hard" sequence of arriving packets. Consider a burst of B 1-byte packets with k processing cycles arriving on the first time slot; NPO invariably accepts them all and begins processing, while OPT is free to reject them. On the second time slot, there arrive B L-byte packets with 1 processing cycle each; they are accepted by OPT, and it begins processing. After that, every kth time slot there arrives a 1-byte packet with k processing cycle (to fill up NPO buffer), and on other time slots L-byte packets with 1 processing cycle arrive, filling up OPT queue. As a result, OPT transmits L bytes per time slot while NPO transmits 1 byte per k time slots, getting the bound in question (asymptotically, since NPO is working on the very first time slot).

To prove the upper bound, note that NPO must fill up its buffer before it drops any packets. Moreover, so long as the NPO buffer is not empty, after at most k time steps NPO must transmit its HOL packet. This means that NPO is transmitting at a rate of at least 1 byte per k time steps, while OPT can transmit at most k packets of size L each over k time slots. Hence, the number of transmitted bytes at time t for NPO is at least t/k (we assume that k divides t evenly for simplicity of exposition) while OPT transmitted at most tL bytes for a competitive ratio of kL so long as the NPO buffer is not empty.

If NPO empties its buffer first, this means that the NPO buffer was congested at some point, so NPO has transmitted at least B bytes, and OPT can transmit at most B more bytes before more packets arrive to the NPO queue. The overall ratio is therefore at most $\frac{tL+B}{t/k}$ under the condition that $t \geq B$, yielding the bound. □

Thus, the simplicity of non-push-out greedy policies does have its price. In the following sections we explore the benefits and analyze performance of push-out policies.

7. Buffer management with SRPT priorities

In this section we address the buffer management problem of when the queueing discipline gives higher priority to packets with fewer required processing cycles. We show first a lower and then an upper bound for the PO Algorithm 2 with SRPT priorities. In this and subsequent sections we focus our attention on the push-out case since non-push-out results have already been shown in Section 6 for all considered priorities.

Theorem 4. *For $B > 2L$, PO is at least L-competitive for SRPT-based priorities.*

Proof. Assume that B/L is an integer. All packets received will have a single residual pass. Consider the following sequence of arrivals. At the beginning $B - 2L + 1$ 1-byte packets arrive. PO accepts all of them. OPT drops all of them. Later on during the same time slot B/L packets of length L arrive, each with a single residual pass. PO drops all of them since their value is no better than the value of packets in its buffer, but OPT accepts all of them and thus OPT buffer is full. During each following time slot one 1-byte packet arrives, each requiring a single processing cycle, followed by one packet of size L bytes, requiring a single processing cycle. PO accepts all 1-byte packets but it does not accept any of the L-bytes packets. Thus, for each time slot when there are arrivals, OPT transmits a packet of size L, and at the same time PO transmits a 1-byte packet. At the end, OPT transmits B bytes while PO transmits $B - 2L + 1$ additional bytes. Therefore, $B + nL$ and $B - 2L + 1 + n$ bytes are transmitted by OPT and PO, respectively, where n is a number of time slots with non-empty arrivals. We obtain that for $n \gg B$, PO cannot have a competitive ratio better than L. □

Next, we show one of our main results, an upper bound for PO with SRPT priorities.

Theorem 5. *For $B > 2L$, PO is at most $4L - 2$-competitive for SRPT-based priorities.*

In what follows we assume that OPT never pushes out packets. Such an optimal solution exists since one can consider the whole input being available to OPT a priori. Thus, all packets accepted by OPT are transmitted. Our analysis will be based on describing a mapping of packets in OPT's buffer to packets transmitted by PO, such that every packet q transmitted by PO has at most $4L - 2$ bytes of OPT associated with it. To facilitate the exposition we describe packet processing as if packets arrive individually and sequentially one at a time, although in reality more than one packet might arrive at a single time step t. The mapping will be dynamically updated for each packet arrival and for each packet transmission, in both OPT and PO.

Mapping routine: During the transmission phase we distinguish between three cases:

T0 If both OPT and PO do not transmit then the mapping remains unchanged.
T1 If PO transmits a packet q then we remove its mapped image in OPT's buffer from future consideration in the mapping. The subset of these OPT packets or bytes that stay in OPT buffer at the end of transmission phase are called of type 1.
T2 If OPT transmits a packet p but its mapped packet q in PO is not transmitted then p is termed a packet of type 2. (We will show next that this case never occurs).

At time t, denote by M_t^O the ordered set of residual pass values for all non-type 1 OPT packets. All M_t^O values are grouped into blocks in the following way. A block is a minimal subset of consecutive M_t^O values starting from the lowest position that is not covered by any previous block, such that the overall length of the packets associated with the block is at least L. The minimal value in each block is called a block *representative*. Denote by R_t an ordered set of representatives at time t. In addition we denote by M_t^P an ordered set of processing cycles values of packets in PO's buffer at time t.

After the arrival at time t of a packet p we distinguish between the following cases:

A0 If p is not accepted by both OPT and PO, then the mapping remains unchanged.
A1 If after acceptance of p some PO packets were dropped then clear the mappings by step A1 between these PO packets and its mapped OPT mates. If p remains in PO's buffer and p is an ith packet in it perform a (P, i)-*mapping-shift* (see Fig. 3(a)): for each non-empty jth block b and jth PO packet q, with $j \geq i$ clear the mapping to q by step A1 and map all packets of block b to q. If p

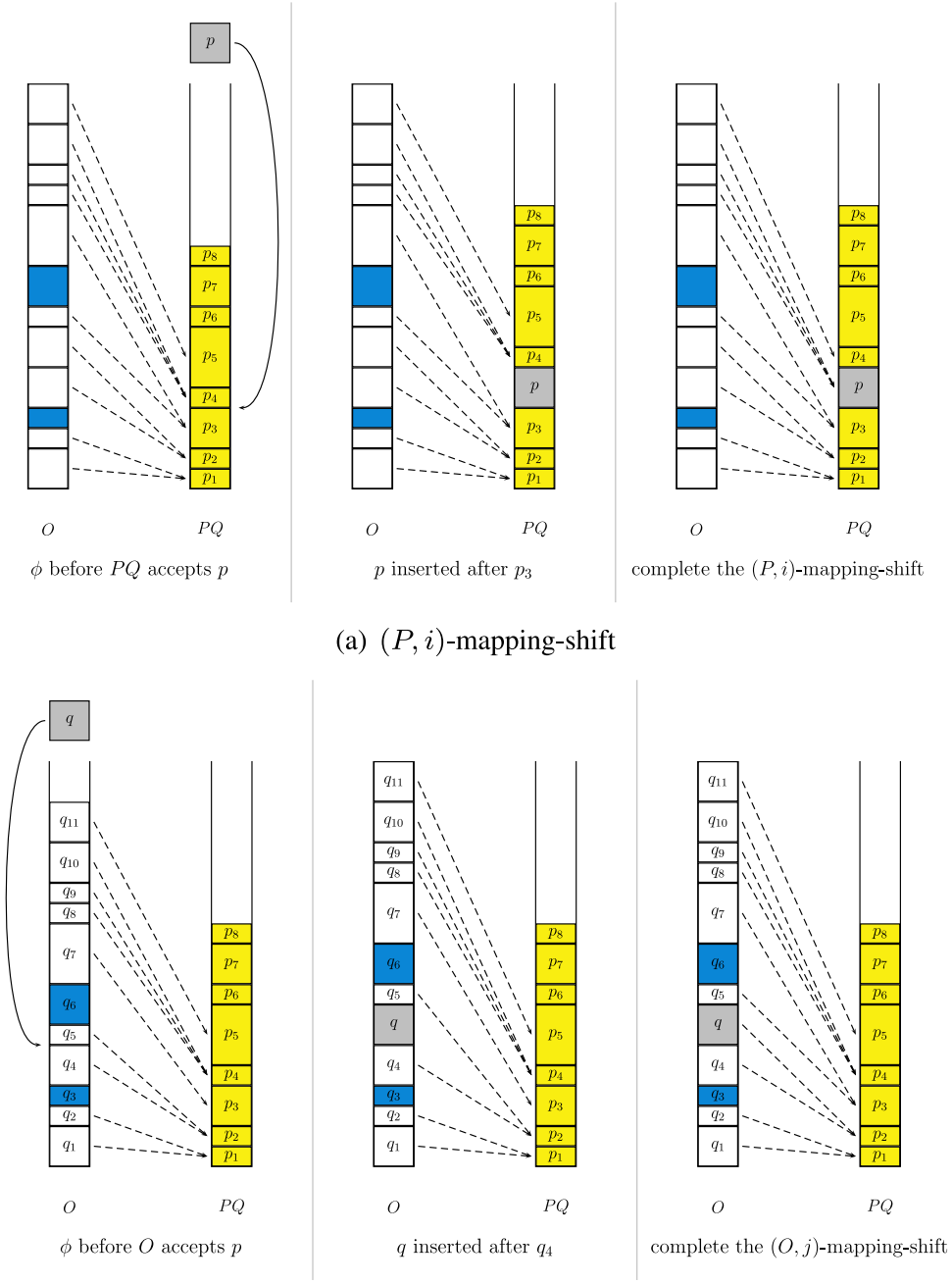


Fig. 3. Example of the mapping used in the proof of Lemma 7, and the mapping shifts performed by the analysis. The white OPT packets should be mapped and white PO packets are available for mapping. The blue OPT packets are of type 1. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this paper.)

is accepted by OPT to the jth block, perform an (O,j)-*mapping-shift* (see Fig. 3(b)): clear all mappings by step A1 between packets of the old lth block and lth PO packet (if both exist), $l \geq j$, recompute blocks starting from the jth and map packets of lth block to lth PO packet if both exist, $l \geq j$.

A2 Clear all mappings assigned by step A2. Map packets of all unmapped blocks to the HOL PO packet.

Lemma 6. *(1) The mapping is feasible.*

(2) *The total length of packets of the same block is at most* $2L - 1$.

Proof. 1. By definition PO accepts the arriving packet and all the packets with packet start above $B - 2L + 1$ are dropped. Hence, if after applying step A1 of the mapping routine there are still unmapped OPT packets then the PO buffer must contain at least one packet. Therefore, all OPT packets unmapped by step A1 are mapped by step A2.

2. In the worst case an total length of all packets in the block except the last one is $L - 1$ and the last packet of the same block has length L, so the claim follows. □

Lemma 7. *After the t-th packet arrives , if an OPT packet p is mapped to a (possibly transmitted) PO packet q then* $r_t(p) \geq r_t(q)$. *Moreover,*

all OPT packets are mapped , and at most $2L-1$ *bytes are mapped to each PO packet by step A1, and possibly at most* $2L-1$ *more bytes are mapped to the HOL packet by step A2 at any time t.*

Proof. We prove the lemma by induction on the number of arrived packets. For the base, consider that the first arriving packet p; by definition PO always accepts it. If p is dropped by OPT then the claim trivially holds. If p is accepted by OPT, it creates a new block with representative p. Clearly, $r_1(p) \geq r_1(p)$, all OPT packets are mapped, and at most L bytes are mapped to the PO packet p, and the base holds.

Assume by induction that for any time $t' < t$, after the arrival of the t'-th packet it holds that for any OPT packet p that is mapped to a (possibly transmitted) PO packet q, $r_{t'}(p) \geq r_{t'}(q)$. Moreover, all OPT packets are mapped and at most $2L-1$ OPT bytes are mapped to each PO packet by step A1. In addition at most $2L-1$ OPT bytes are mapped to the HOL packet in PO buffer at time t' by step A2.

Clearly, if a representative of a block p' mapped to a PO packet q by step A1 satisfies $r_{t'}(p') \geq r_{t'}(q)$ at time t', then for any packet p'' of the same block $r_{t'}(p'') \geq r_{t'}(q)$. To show that this holds after the tth packet arrives, it suffices to consider the ordered set of representatives R_{t-1} and its update after this arrival and show that $R_t \geq M_t^P$. By the induction hypothesis the remaining number of processing cycles of any OPT packet is at least the number of processing cycles of its PO counterpart, i.e., $R_{t-1} \geq M_{t-1}^P$, and there are no OPT packets of type 2 formed while the first $t-1$ packets were accepted. Denote by R_t^1 a set of representatives of blocks mapped by step A1. Since all packets of the same block are mapped to the same PO packet, $\left|R_{t-1}^1\right| \leq \left|M_{t-1}^P\right|$.

We denote by $t-$ the time slot just before the arrival of the t-th packet. First suppose that a transmission occurs before the t-th packet arrives, i.e., between the t-th and $(t-1)$-th packet arrivals at least one packet is transmitted by OPT or PO. By the induction hypothesis, it is impossible for OPT to transmit a packet corresponding to the first value in M_{t-1}^P before the packet whose value is the first in M_{t-1}^P, and this holds for any sequence of transmissions prior to the t-th arrival. Therefore, if PO transmits between the $t-1$-th and t-th arrival, $\left|M_{t-}^P\right|$ is reduced by one. On the other hand, $\left|M_{t-}^O\right|$ is reduced by the number of packets in the first block (if any) mapped by step A1 to the packet sent by PO. Hence, $\left|R_{t-}^1\right| \leq \left|M_{t-}^P\right|$. Moreover, any value mapped by step A2 to a packet transmitted by PO is removed from M_{t-}^O upon this transmission (by the definition of M_t^O which consists of non-type 1 packets only). Thus, $R_{t-} = R_{t-}^1$ and $\left|R_{t-}\right| \leq \left|M_{t-}^P\right|$ in this case, and the claim holds at time $t-$, in particular $R_{t-} \geq M_{t-}^P$.

Consider now the arrival of the t-th packet p. We distinguish the following cases.

Case 0 OPT does not accept p and PO accepts and immediately drops p. We are done.

Case 1 PO does not drop p, OPT does not accept p. In this case, $R_t = R_{t-}$ and it suffices to show that $R_{t-} \geq M_t^P$.

Case 1.1 $\left|R_{t-}\right| \geq \left|M_{t-}^P\right|$: Since $\left|R_{t-}^1\right| \leq \left|M_{t-}^P\right|$, some OPT packets represented in R_{t-} are mapped by step A2. In this case, the last packet in PO buffer occupies the $(B-2L+1)$-th byte (each block has length $\geq L$, each block is mapped to a single PO packet, and all bytes in PO buffer are available for mapping). PO does not drop p, so the value of $r_t(p)$ is at most the last value in M_t^P. Since OPT does not accept p, by Corollary 2(i), $R_{t-} \geq M_{t-}^P \cup \left\{r_t(p)\right\} = M_t^P$.

Case 1.2 $\left|R_{t-}^1\right| \leq \left|M_{t-}^P\right|$: Again, since in this case OPT does not accept p, by Corollary 2(ii), $R_{t-} \geq M_{t-}^P \cup \left\{r_t(p)\right\} = M_t^P$.

Case 2 OPT accepts p, PO drops p. In this case $M_t^P = M_{t-}^P$, and $r_t(p)$ is larger than any value in M_t^P. Let l be the position of p in OPT buffer. For any $m \geq l$ the mth OPT packet has more residual cycles than any value in M_t^P, so for any OPT packet p' mapped to PO packet q by step A1 $r_t(p') \geq r_t(q)$, and we have $R_t \geq M_t^P$.

Case 3 OPT accepts p, PO does not drop p. If $\left|R_{t-}\right| \leq \left|M_{t-}^P\right|$ or $\left|R_{t-}\right| \geq \left|M_{t-}^P\right|$, then similar to the Cases 1.1 and 1.2, by Lemma 1 we have that $R' = R_{t-} \cup \left\{r_t(p)\right\} \geq M_{t-}^P \cup \left\{r_t(p)\right\} = M_t^P$. Therefore, in this case it suffices to show that $R_t \geq R'$, which in turn implies $R_t \geq M_t^P$. Let j denote the index of the block where p is inserted in OPT. We have to consider two possibilities for the position of p's number of processing cycles in R', which could be either the jth or $(j+1)$-st.

Case 3.1 $R'[j] = r_t(p)$: In this case p now serves as the representative of block j, i.e., $R_t[j] = r_t(p)$. If $\ell(p) = L$ then p forms a full block and $R_t[m] = R_{t-}^1[m-1]$ for all $m > j$. Therefore, $R_t \geq R'$ (actually, in this case we have strict equality). Otherwise, $\ell(p) < L$, and at least as many residual processing cycles as the $(l+1)$th element will join the jth block after recomputation (since such a block must add at least L to total length). We therefore have $R_t[m] \geq R_{t-}^1[m]$ for all $m > j$. Since $R'[m] = R_{t-}[m-1]$, $m > j$, this case follows.

Case 3.2 $R'[j+1] = r_t(p)$: In this case the representative of block j remains unchanged, i.e., $R'[j] = R_t[j] = R_{t-}[j]$ and $R'[m] = R_{t-}[m-1]$, $m > j+1$. Since p belongs to the jth block after acceptance and $r_t(p)$ is not a representative of the block, then $R_t[j+1] \geq r_t(p)$. Since after recomputation representatives will move up for no more than one block in R_t compared to R_{t-1}, $R_t[m] \geq R_{t-}[m-1]$, $m > j+1$. Therefore, $R_t \geq R'$ and this case follows.

Now let us show that there are sufficiently many PO packets to map all of OPT packets such that at most $2L-1$ bytes are assigned by step A1 to each transmitted packet of PO and additionally at most $2L-1$ bytes are assigned to the HOL packet of PO by step A2. Recall that the claim holds for time $t-$. Consider the arrival of the t-th packet p. If PO accepts p, then the claim holds, since this new packet can support the block changes (and possible addition) that may potentially occur if OPT also accepts p. If PO does not accept p then by the definition of PO this can only happen if the buffer occupancy of PO is at least $B-2L+1$. Clearly, the total length of a block mapped by step A1 to any PO packet is at most $2L-1$ (by definition). Furthermore, since we have shown that $R_t \geq M_t^P$, and by definition the blocks are of total length at least L, it must follow that the total length of packets in PO covers at least this amount of total length of packets in OPT mapped to PO by step A1. It follows that the remaining total length of packets in the buffer of OPT that are not mapped by step A1 can be at most $2L-1$ (the possibly unused space in PO). It follows that the total length of packets mapped to the HOL packet of PO by step A2 is at most $2L-1$, as required. □

Theorem 5 now follows immediately from Lemma 7. Next we generalize the previous mapping and show how to improve the upper bound of PO for sufficiently large buffers.

Theorem 8. *PO is at most* $\frac{(2L-1)(N+1)}{N}$*-competitive for SRPT-based priorities , where* $N = \left\lceil \frac{B-2L+1}{2L-1} \right\rceil$.

The idea is to redistribute bytes mapped by step A2 between different PO packets. Let $N = \left\lceil \frac{B-2L+1}{2L-1} \right\rceil$. We consider an updated version of step A2 that maps at most $\frac{2L-1}{N}$ value to each PO packet. The mapping routine is unchanged during the transmission phase and now it operates on M_t^O as follows. Denote by M_t^O at time t the ordered set of values of processing cycles of type 1 OPT packets that are not mapped by the step A2 as defined below. We now exclude from future consideration by step A1 all OPT packets

mapped by step A2 even before their PO mates are transmitted. The definition of block, representative, R_t, and M_t^P remain unchanged.When a packet p arrives at time t, steps A0 and A1 remain unchanged. Next we define the steps that do change.

A2 If prior to t there are no bytes mapped by step A2 and after the t-th A1 step there are still Y unmapped OPT bytes then a jth portion of $\frac{Y}{N}$ bytes unmapped by step A1 map to PO packet whose mapped block contains the $((j-1)(2L-1)+1)$-st byte x, $1 \le j \le N$. We say that x "defines" a mapping of this portion of still unmapped bytes. Let Y be the overall length mapped by step A2 prior to time t and still there are Y_0 unmapped bytes after applying step A1 during time t. Let the mapping of the Yth byte assigned by step A2 be the lth byte in the OPT buffer. Map each jth portion of $\frac{Y_0}{N}$ still unmapped by step A1 byte to PO packet whose mapped block contains the $(j(2L-1)+l+1)$st byte, $1 \le j \le N$. Observe that both these bytes can be remapped to the other PO packet during the (O,j)-*mapping-shift*.
A3 Values unmapped by steps A1 and A2 are assigned to the HOL PO packet. We will show that step A3 is never applied and is required only for completeness.

The mapping is feasible since during arrivals the PO buffer contains at least one packet and any value that is unmapped by steps A1 and A2 is assigned by step A3 to the HOL PO packet. Lemma 6(2) remains the same. The next lemma is very similar to Lemma 7. Namely, if an OPT packet p is mapped by step A1 to a (possibly transmitted) PO packet q then $r_t(p) \ge r_t(q)$. The fact that the total value assigned to each PO packet is at most $\frac{(2L-1)(N+1)}{N}$ follows from the fact that for each OPT packet p that is mapped by step A1 to a PO packet q at any time t, $r_t(p) \ge r_t(q)$, the maximal block size is $2L-1$ bytes. Theorem 8 follows immediately from Lemma 9.

Lemma 9. *After arrival of the t-th packet , if an OPT packet p is mapped to a (possibly transmitted) PO packet q then $r_t(p) \ge r_t(q)$. Moreover, all OPT packets are mapped and at most $\frac{(2L-1)(N+1)}{N}$ value is mapped to each PO packet at time t , where $N = \left\lceil \frac{B-2L+1}{2L-1} \right\rceil$.*

Corollary 10. *If $B > 4L^2 - 2L$ then PO is at most $2L$-competitive for SRPT-based priorities .*

8. Buffer management with LP priorities

We begin with a lower bound for the PO algorithm with LP-based priorities and then proceed to an upper bound of PO with LP-based priorities.

Theorem 11. *PO is at least k-competitive for LP-based priorities on a sufficiently long sequence .*

Proof. Here, we will consider a push-out version of OPT for simplicity of description. Assume $\frac{B}{L}$ be an integer value. Consider a cycle of L iterations of the first type and later sequence of $n > 0$ iterations of the second type (defined below). Each iteration of the first type contains $k-1$ time slots. At the beginning of the ith iteration of the first type $\left\lceil \frac{B}{i} \right\rceil$ packets of i bytes with k processing cycles arrive and later during the same time slot $\left\lceil \frac{B}{i} \right\rceil$ packets of i bytes with 1 processing cycles arrive. OPT drops the first subsequence and accepts the second. On the other hand, PO accepts the first subsequence and drops the second. So during each iteration of the first type OPT transmits $i(k-1)$ bytes but PO transmits zero bytes. At the beginning of the next iteration of the first type both algorithms push out already admitted packets that still remain in their buffers.

After the Lth iteration, both buffers are nearly full with packets of size L, but with k processing cycles in the case of PO and one residual pass in the case of OPT. Now a sequence of the second type starts. After the last transmission by PO, $k+1$ packets arrive in the following order: first one L-byte packet with k passes and thereafter k packets of length L with a single residual pass. The first packet is accepted by PO and dropped by OPT. The latter $k-1$ packets are dropped by PO and accepted by OPT. Each buffer is completely full again. So during each iteration of the second type OPT transmits kL bytes but PO only L bytes. After n iterations of the second type the overall transmission of OPT is $\frac{L(1+L)(k-1)}{2} + knL + B$ while PO transmits $Ln + B$ bytes. Thus, the lower bound on competitive ratio of PO is $\frac{L(1+L)(k-1)+2kLn+2B}{2(Ln+B)}$. □

Theorem 12. *PO is at most $(k+3)$-competitive for LP-based priorities with sufficiently big buffers.*

Sketch. The mapping routine during the transmission phase remains the same as in Section 7.

Mapping routine ϕ: During the transmission phase we distinguish between the three following cases:

T0 If neither OPT nor PO transmits then the mapping remains unchanged.
T1 If PO transmits a packet q then we remove its mapped image in OPT's buffer from future consideration in the mapping. A subset of these OPT packets or bytes that stays in the OPT buffer at the end of transmission are called of type 1.
T2 If OPT transmits a packet p but its mapped packet q in PO is not transmitted then p is termed a packet of type 2.

During the arrival of a packet p at time t steps A0, A2 and A3 are the same as in Theorem 8. At time t, denote by M_t^O a set of non-type 1 packets sojourns in OPT buffer and not mapped by step A2. In addition is ordered in non-increasing order of packet length. All M_t^O packets are grouped into blocks in the following way. Let q be an ith packet in PO buffer at time t. An ith block is defined in the following way. Consider a minimal set B_0 of packets starting from the lowest position that are represented in M_t^O and not covered by any other block whose overall required work is at least $r_t(q)$. If the overall length of all packets in B_0 is at least $\ell(q)$ then B_0 forms a block. Otherwise, add to B_0 a minimal set of packets B_1 starting from the first packet that is represented in M_t^O and not covered by B_0 such that the overall length of packets in $B_0 \cup B_1$ will be at least $\ell(q)$. In this case a set of packets that is covered by $B_0 \cup B_1$ defines a block. Denote by $\ell(X)$ the overall length of packets and by $r_t(X)$ the overall required work in a set of packets X at time t. A block b that is mapped to a PO packet q is called *fully mapped* to a packet q at time t if $\ell(b) \ge \ell(q)$ and $r_t(b) \ge r_t(q)$. Observe that it is possible that $\ell(B_0)$ will be less than its PO counterpart. In this case OPT may later accept packets that will not be accepted by PO.

We define a new step A1 where the blocks are recomputed after a (P,i)-*mapping-shift*.

A1 If after p is accepted some PO packets were dropped then clear the mappings by step A1 between these PO packets and its OPT counterparts. If p remains in PO buffer and p is the ith packet in PO buffer, perform a (P,i)-*mapping-shift*: clear all mappings by step A1 between packets of the old lth block in OPT buffer and lth packet in PO buffer, $l \ge j$, recompute blocks from jth and map packets of lth block to the lth PO packet if both exist, $l \ge j$. If p is accepted by OPT to the jth block, perform an (O,j)-*mapping-shift*: clear all mappings by step A1 between packets in OPT buffer of the old lth block and lth packet in PO

buffer (if both exist), $l \geq j$, recompute blocks from jth and map packets of lth block to lth PO packet if both exist, $l \geq j$.

Clearly, the mapping is feasible since if OPT accepts some packet that is not accepted by PO, PO buffer contains at least one packet. Since affected blocks are recomputed after each (P,i)-*mapping-shift* and (O,j)-*mapping-shift* and by definition of a block b that is mapped to a PO packet q at time t, $\ell(b) \geq \ell(q)$ and $r_t(b) \geq r_t(q)$. Thus, we will consider sufficiently big buffers where $\frac{2L-1}{B}$ tends to zero and because of the above properties of the block step A2 will introduce at most additional ϵ value for each PO packet. Hence, for each packet q transmitted by PO, OPT transmits at most $(k+1)l(q)+\epsilon$ by steps A1 and A2. Denote by T the total number of bytes transmitted by PO and by P the total number of bytes transmitted by OPT during processing of pushed-out PO packets. Thus, the competitive ratio is at most $\frac{(k+1+\epsilon)T+P}{T}$. Now let us estimate P and substitute it into the previous expression. For each pushed-out by PO packet p denote by $T(p)$ a number of time slots when p was HOL before it was pushed-out. Clearly, that the process of push-outs of packets that have positive $T(p)$ will be stopped once all packets will have a maximal packet length L or it can continue during each time slot when there is at least one packet in PO buffer of length smaller than L. Moreover, if push-out happens the buffer occupancy is at least $B-2L+1$. Denote by $\mathcal{P}$ a set of PO packets pushed-out during this interval of time. So for each $B-2L+1$ bytes transmitted by PO, P is bounded by $\sum_{p\in\mathcal{P}} T(p)l(p) \leq \sum_{p\in\mathcal{P}} kl(p)] \leq kL(L+1)/2$. Thus, PO is at most $\frac{2(k+1+\epsilon)B+kL(L+1)}{2(B-2L+1)}$-competitive. For the buffers that are significantly bigger than $kL(L+1)$, PO is at most $k+3$-competitive. □

9. Buffer management with MEP priorities

In this section we study the performance of a BM implementing PQ, where priorities are set in accordance with the non-increasing order of processing cycles divided by packet length. This priority is dubbed the *Most Effective Packet* first priority (MEP), or the *effective-ratio* priority. Recall that our objective here is to maximize the number of bytes transmitted in total. Non-push-out results are similar to Section 6.

The following theorem provides a lower bound on the performance of the push-out MEP policy. Note that it is significantly larger (worse) than the lower bound of $\left(1+\frac{L\ln k-k-L}{B}\right)$ previously proven in Kogan et al. (2014) (Theorem 13); while we had hoped that MEP priorities might have good competitiveness guarantees, the following result shows a linear lower bound, making MEP basically no better than the other policies in the worst case.

Theorem 13. *PO with MEP-based priorities is at least* $\left(\frac{9}{16}\min\{k,L\}\right)$*-competitive.*

Proof. We show the bound for the case of $B=k=L=n$. On the first time slot, there arrives an $\frac{n}{2}$-byte packet with n processing cycles followed by a 1-byte packet with 2 processing cycles. PO begins processing the larger packet, while OPT drops it and begins processing the 2-byte packet. On the second time slot, there arrives an $\frac{n+2}{2}$-byte packet with n processing cycles followed by a 1-byte packet with 2 processing cycles. Since $\frac{n+2}{2n} > \frac{n}{2(n-1)}$, it pushes out the previous large packet, and PO begins processing the large packet. This is repeated until on the $\frac{n}{4}$-th time slot, there arrives an n-byte packet with n processing cycles. By this point, PO has not processed a single packet, and its buffer contains only the last n-byte packet, while OPT has already processed $\frac{n}{8}$ 1-byte packets and has $\frac{n}{8}$ more of them in the queue. In a time slot, there arrive n, 1-byte packets with 1 processing cycle each; PO does not accept them since their ratio is worse than $\frac{n}{n-1}$; OPT buffer is now full of these 1-byte packets. For the next $\frac{n}{2}-1$ steps, PO keeps processing the large packet while OPT keeps processing the 1-byte packets. Finally, when PO has a packet with n bytes and $\frac{n}{2}-1$ processing cycles, a packet with 2 bytes and 1 processing cycle arrives, replacing it in PO buffer, and then both algorithms finish their packets. As a result of this sequence, OPT has processed $n+\frac{n}{8}$ bytes in total while PO has processed 2 bytes, getting the bound. □

10. Simulation study

10.1. General remarks

In order to obtain a better understanding of the differences between our proposed solutions, we conducted a simulation study where we evaluate the performance of each policy in terms of throughput and address the effect of variable processing requirements on the average delay in the system.

Publicly available traffic traces (such as CAIDA) do not contain, to the best of our knowledge, information on the processing requirements of packets. Furthermore, these requirements are difficult to extract since they depend on the specific hardware and NP configuration of the network elements. Another handicap of such traces is that they provide no information about time-scale, and specifically, how long should a time-slot last. This information is essential in our model in order to determine both the number of processing cycles per time-slot, as well as traffic burstiness. We therefore perform our simulations on synthetic traces. Our simulation results are based on traffic composed of the interleaving of 100 independent sources, with each source generated by an on–off bursty process modeled by a Markov-modulated Poisson process (MMPP). During every time slot, each source has probability 0.05 to be switched on, and once switched on, probability 0.2 to be switched back off. When a source is on, it emits packets with intensity λ_{on}, which represents one of the parameters governing traffic generation. Each generated packet is assigned two parameters: (i) required processing chosen uniformly at random from $\{1,\ldots,k\}$ (k being the maximum amount of processing required by any packet), and (ii) packet length, chosen uniformly at random from $\{1,\ldots,L\}$ (L being the maximum length of a packet in the system). Each of our results follows from simulating the system for 5, 000, 000 time slots; we allowed different parameters to vary in each set of simulations in order to better understand the effect each parameter has on system performance and further validate our analytic results and algorithmic insights.

We simulated the throughput performance of our three proposed policies, all based on the greedy algorithm depicted in Algorithm 2: (i) SRPT, (ii) LP, and (iii) MEP. In order to obtain a better qualitative differentiation between the policies, we compared their throughput performance with that of a "virtual" policy, which serves as an approximate upper bound on the optimal throughput possible. This virtual policy essentially transforms each arriving packet requiring $k' \leq k$ processing cycles, and having length $\ell' \leq K$, into k' distinct packets, each requiring one processing cycle, and having length ℓ'/k', using the LP/MEP as the scheduling and admission criteria (they are equivalent for such virtual inputs). Clearly the performance of this virtual policy serves as an approximate upper bound on the performance of the optimal policy, since this policy profits from any partial processing of a packet. We use this approximation since finding the actual optimal algorithm would be computationally prohibitive: even identifying the best set of packets to store in a single time step is equivalent to the knapsack problem which is NP-hard. In Figs. 4(a), 5(a) and 6(a), which demonstrate the throughput performance of the system, the y-axis represents the ratio between the throughput obtained by a

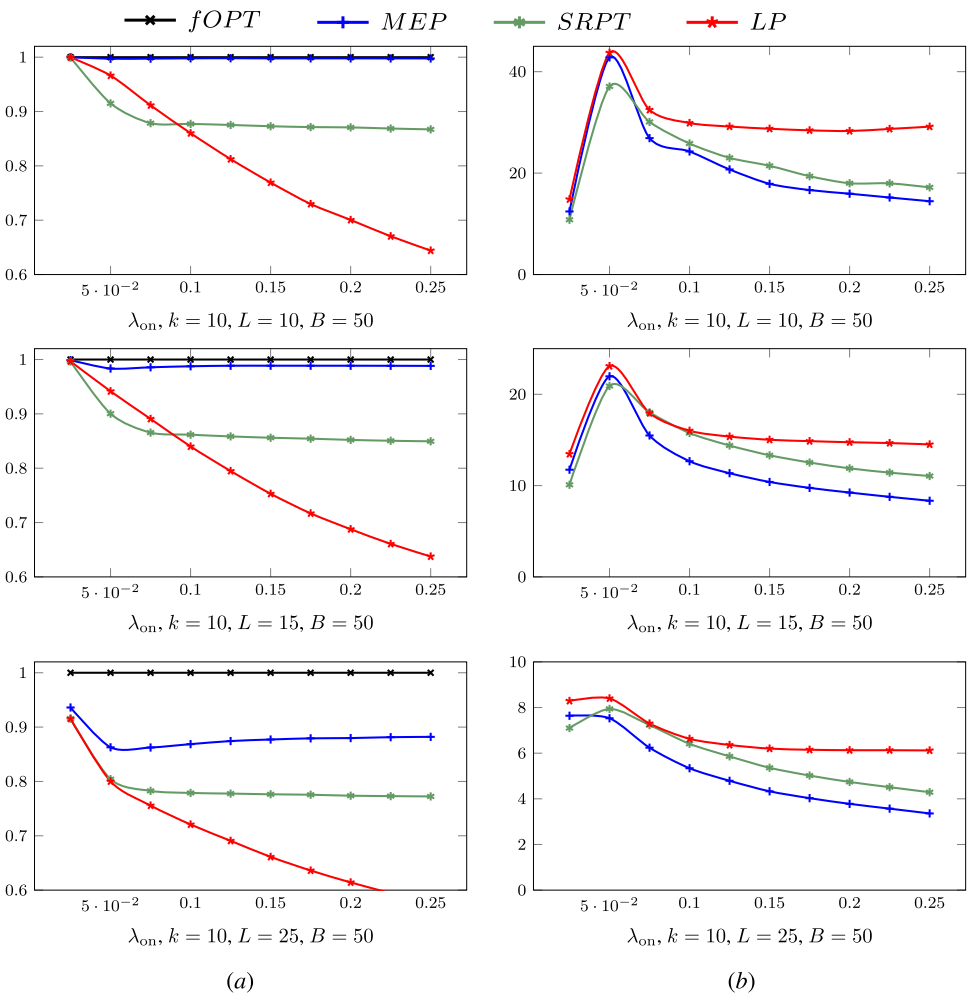


Fig. 4. Throughput performance (a) and latency (b) as a function of incoming stream intensity λ_{on} for three different values of maximal packet length L.

policy and the throughput obtained by the virtual policy (which serves as an approximate upper bound on the optimal policy).

Another set of results produced by our simulation study deals with the average queuing delay of packets for each of the policies considered. As mentioned in the introduction, queueing delay has long been known to be directly related to the buffer size available for the queue. Our work tries to shed light on the role of variable processing requirements as a major factor affecting queueing delay in such heterogeneous environments and relate this latency performance to that of the attainable throughput. In Figs. 4(b), 5(b) and 6(b), which demonstrate the average latency in the system, the y-axis represents the average latency (in time slots) overall packets delivered.

We present here only a small sample of our results, aiming to explore the effect of various parameters examined in our study. Specifically, we consider the effect of offered-load, average number of processing required by a packet, buffer size, and average packet length. For each of the first three parameters here mentioned, we present a cross section of the effect of average packet length by providing three plots corresponding to maximum allowed packet length values $L = 10, 15, 25$.

10.2. Varying traffic intensity

Fig. 4 shows the system performance as a function of increased average load, where we increase the rate of each independent source by increasing the parameter λ_{on} which governs packet intensity during a burst period. Fig. 4(a) shows that, in general, MEP is the best policy (this will always be the case throughout our simulations). However, when examining the other two policies, although as traffic intensity increases SRPT significantly outperforms LP, under low load conditions and small values of L, LP outperforms SRPT. This indicates that under moderate load conditions, and when packet length variability is small, it is best to prioritize longer packets rather than by their processing requirements. When either as traffic intensity increases (or packet length variability grows), the system will be prone to increased congestion, whose alleviation is possibly by preferring packets which take a shorter time to process. As for the latency, Fig. 4(b) shows that average latency increases up to a certain point, and then steadily decreases. This increase occurs in moderate load conditions, where all algorithms are "forced" to accept non-favorable packets. However, as traffic intensity increases, all algorithms have a better selection of packet to accept, and each will focus on its more preferable packets, thus resulting in decreasing packet latency.

10.3. Increasingly heterogeneous processing requirements

Fig. 5 shows the system performance as we allow packet processing requirement to increase, both in value and in variability. As demonstrated in Fig. 5(a), while the MEP policy outperforms both other policies, for relatively small L we observe a transition from LP to SRPT as the second best policy. One can see that while the average number of processing cycles is relatively low, the LP policy outperforms the SRPT policy, while as the average number of required processing increases beyond some threshold, SRPT

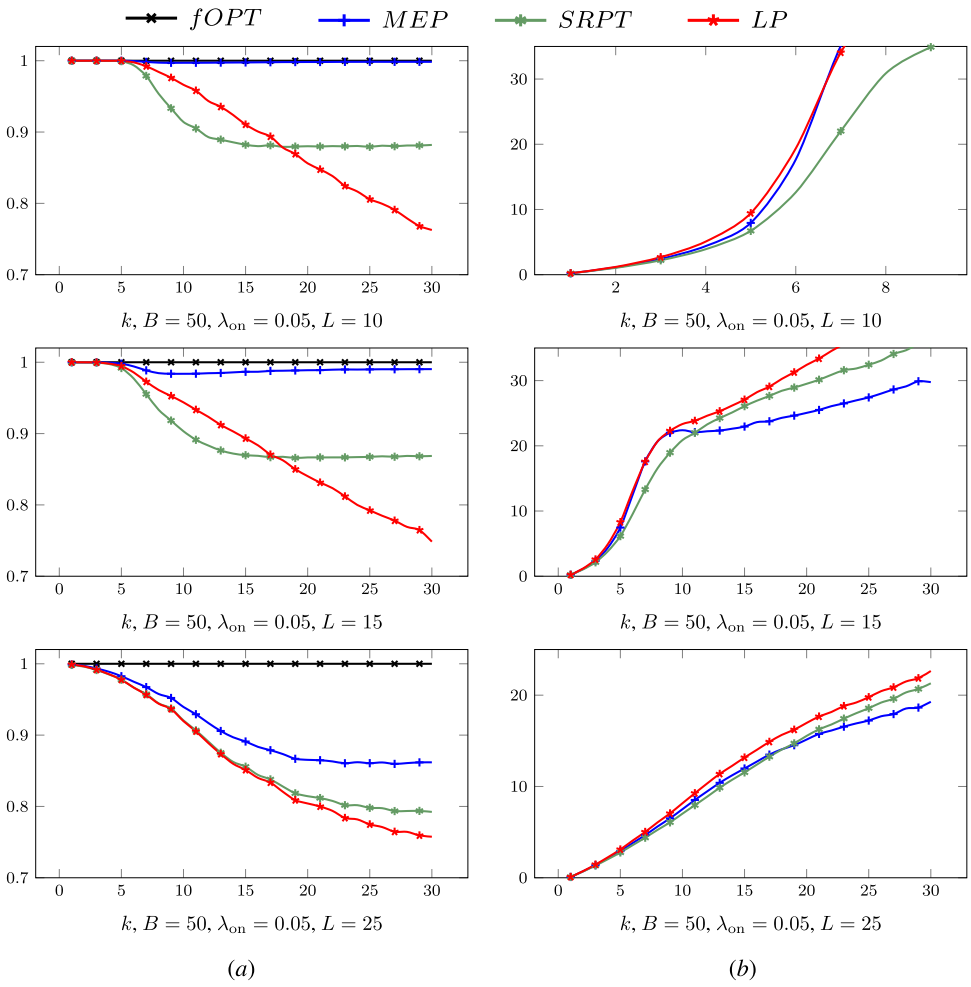


Fig. 5. Throughput performance (a) and latency (b) as a function of maximal required work k for three different values of maximal packet length L.

becomes superior to LP. This behavior is similar to that observed in the study of the effect of traffic-intensity on the performance in Section 10.2. This coincides with the intuition that the actual notion of load in the system is actually the product of the average required processing and packet arrival rate. The simulation results presenting the effect of increasing load and increasing required processing on the system's throughput are in accordance with the results obtained in our analytic study, which show that the ratio between the parameters k and L indeed corresponds to which of the policies is expected to be superior. The latency graphs here (Fig. 5(b)) are mostly strictly increasing, with latency becoming pronounced as the overall arrival load (in the sense just described) topping the system's processing service rate (this occurs at around k=5). When examining the differences in latency as average packet length increases, one can see that the average latency (for the same values of k) is inversely proportional to the average packet length. This is due to the fact that every successful transmission when packets are larger leaves reduces the delay of packets remaining in the queue.

10.4. The effect of buffer size

Fig. 6 shows the effect buffer size has on system performance. Fig. 6(a) shows that buffer size has relatively little effect on the differences in throughput: all three policies relatively quickly achieve their corresponding maximal performance and stay there as buffer size grows further. This is due to the fact that beyond a certain point, packet arrival rate is smoothed by the availability of buffer space. In terms of latency, Fig. 6(b) shows a steady increase in latency, which should be ascribed to queueing delay. However, the LP policy exhibits the best performance in these scenarios since it favors the transmission of longer packets first, which alleviate the latency sensed by the remaining packets in the buffer.

In general, our results clearly show that the MEP policy is better than both other policies with respect to throughput. Note that in terms of latency the best policy (MEP) does not necessarily outperform other policies: since it processes more packets, some of them must wait for their turn longer.

Our simulation results and the insights they provide can serve as a rule of thumb in choosing the best policy for a specific network scenario, depending on expected traffic characteristics.

11. Conclusion

Increasingly heterogeneous packet processing requirements in modern networks pose novel design challenges to NP architects. In this work we study the impact of two important characteristics, maximal required processing k and maximal packet size L, and show the significance of the relationship between k and L. We introduce three different priority regimes for processing: SRPT, LP, and MEP, and study their performance in queues with bounded buffers. We present results for both non-push-out, as well as push-out buffer management algorithms, which give guarantees on the worst-case performance of such algorithms, without resorting to any assumptions on the process generating the traffic. Due to this

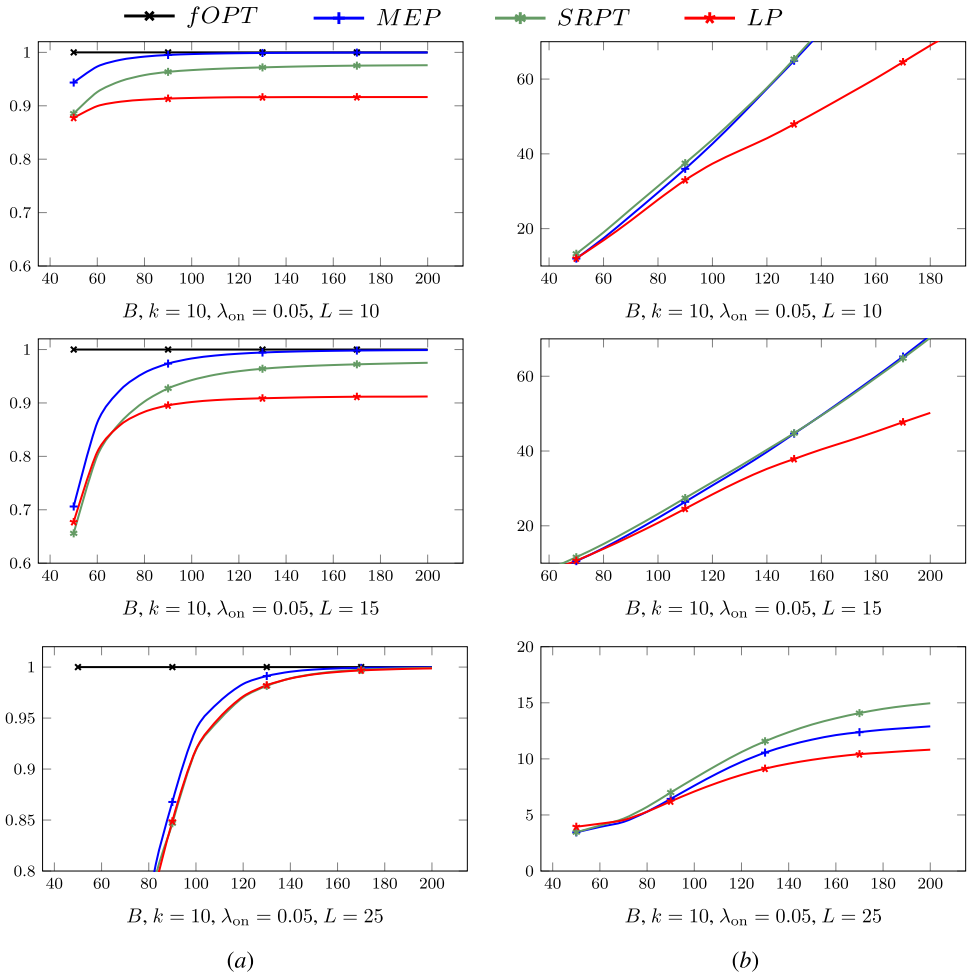


Fig. 6. Throughput performance (a) and latency (b) as a function of buffer size B for three different values of maximal packet length L.

approach, results can be globally applicable in various networking environments which may deal with highly heterogenous traffic patterns.

Our results show that implementing a push-out mechanism, although potentially costly in terms of vendor implementation, has a significant impact on the system's performance, primarily in terms of throughput. In general, in this case two characteristics (size and processing requirements) introduce a natural tradeoff between "large profits" from processing large packets and "fast gains" from processing packets with small processing requirements. Interestingly, our results indicate that from the point of view of worst case guarantees, it suffices to optimize only one of these characteristics, and MEP priorities do not bring any significant improvements.

Straightforward remaining open questions include closing the gaps between the upper and lower bounds shown in Table 1. Another interesting direction would be to consider weighted throughput where each packet also has an intrinsic value; this will cover some other practical cases such as differentiation of service frameworks.

Acknowledgments

The work of Sergey Nikolenko was partially supported by the Government of the Russian Federation Grant 14. Z50.31.0030 and the Russian Presidential Grant for Young pH. D.s MK-7287.2016.1. The work of Gabriel Scalosub was supported by the Israel Science Foundation (Grant no. 1036/14). Work by Michael Segal has been supported by Israel Science Foundation (Grant no. 317/15), by IBM Corporation and by Israel Ministry of Economy and Industry.

References

Guido Appenzeller, Isaac Keslassy, Nick McKeown, 2004. Sizing router buffers. In: SIGCOMM, pp. 281–292.

Yossi Azar, Oren Gilon, 2015. Buffer management for packets with processing times. In: Algorithms—ESA 2015—23rd Annual European Symposium, Patras, Greece, September 14–16, 2015, Proceedings, pp. 47–58.

Borodin, Allan, El-Yaniv, Ran, 1998. Online Computation and Competitive Analysis. Cambridge University Press, Cambridge, UK.

Brucker, Peter, Heitmann, Silvia, Hurink, Johann, Nieberg, Tim, 2006. Job-shop scheduling with limited capacity buffers. OR Spectr 28 (2), 151–176.

Pavel, Chuprikov, Nikolenko, Sergey I., Kogan, Kirill, 2015. Priority queueing with multiple packet characteristics. In: 2015 IEEE Conference on Computer Communications, INFOCOM 2015, Kowloon, Hong Kong, April 26–May 1, 2015, pp. 1418–1426.

CAIDA The Cooperative Association for Internet Data Analysis. [Online] ⟨http://www.caida.org/⟩.

Keslassy, Isaac, Kogan, Kirill, Scalosub, Gabriel, Segal, Michael, 2012. Providing performance guarantees in multipass network processors. IEEE/ACM Trans Netw 20 (6), 1895–1909.

Kesselman, Alexander, Kogan, Kirill, 2007. Nonpreemptive scheduling of optical switches. IEEE Trans Commun 55 (6), 1212–1219.

Kesselman, Alexander, Kogan, Kirill, Segal, Michael, 2010. Packet mode and qos algorithms for buffered crossbar switches with fifo queuing. Distrib Comput 23 (3), 163–175.

Kesselman, Alexander, Kogan, Kirill, Segal, Michael, 2012. Best effort and priority queuing policies for buffered crossbar switches. Chic J Theory Comput Sci 2012.

Kesselman, Alexander, Kogan, Kirill, Segal, Michael, 2012. Improved competitive performance bounds for cioq switches. Algorithmica 63 (1–2), 411–424.

Kesselman, Alexander, Patt-Shamir, Boaz, Scalosub, Gabriel, 2013. Competitive buffer management with packet dependencies. Theory Comput Sci 489–490,

75–87.
Kogan, Kirill, López-Ortiz, Alejandro, Nikolenko, Sergey I., Sirotkin, Alexander, 2013. Multi-queued network processors for packets with heterogeneous processing requirements. In: Proceedings of the 5th International Conference on Communication Systems and Networks (COMSNETS 2013), pp. 1–10.
Kogan, Kirill, López-Ortiz, Alejandro, Nikolenko, Sergey I., Sirotkin, Alexander V., 2012. A taxonomy of semi-FIFO policies. In: Proceedings of the 31st IEEE International Performance Computing and Communications Conference (IPCCC 2012), pp. 295–304.
Kogan, Kirill, López-Ortiz, Alejandro, Nikolenko, Sergey I, Sirotkin, Alexander V., 2016. Online scheduling fifo policies with admission and push-out. Theory Comput. Syst. 58 (2), 322–344.
Kogan, Kirill, López-Ortiz, Alejandro, Nikolenko, Sergey I., Scalosub, Gabriel , Segal, Michael, 2014. Balancing work and size with bounded buffers. In: Proceedings of the 6th International Conference on Communication Systems and Networks (COMSNETS 2014), pp. 1–8.
Leonardi, Stefano, Raz, Danny, 1997. Approximating total flow time on parallel machines. In: STOC, pp. 110–119.
Motwani, Rajeev, Phillips, Steven, Torng, Eric, 1994. Non-clairvoyant scheduling. Theory Comput Sci 130 (1), 17–47.
Muthukrishnan, S., Rajaraman, Rajmohan, Shaheen, Anthony, Gehrke, Johannes E., 2005. Online scheduling to minimize average stretch. SIAM J Comput 34 (2), 433–452.
Nikolenko, Sergey I., Kogan, Kirill, 2015. Single and multiple buffer processing. In: Encyclopedia of Algorithms. Springer, Philadelphia.
Pruhs, Kirk, 2007. Competitive online scheduling for server systems. SIGMETRICS Perform Eval Rev 34 (4), 52–58.
Ramaswamy, Ramaswamy, Weng, Ning, Wolf, Tilman, 2009. Analysis of network processing workloads. J Syst Archit —Embed Syst Des 55 (10–12), 421–433.
Ruiz, Rubén, Vázquez-Rodrígue, José Antonio, 2010. The hybrid flow shop scheduling problem. Eur. J. Oper. Res. 205(1), 1–18.
Schrage, Linus, 1968. A proof of the optimality of the shortest remaining processing time discipline. Oper Res 16, 687–690.
Sivaraman, Anirudh, Subramanian, Suvinay, Agrawal, Anurag, Chole, Sharad, Chuang, Shang-Tse, Edsall, Tom, Alizadeh, Mohammad, Katti, Sachin, McKeown, Nick, Balakrishnan, Hari, 2015. Towards programmable packet scheduling. In: Proceedings of the 14th ACM Workshop on Hot Topics in Networks, Philadelphia, PA, USA, November 16–17, 2015, pp. 23:1–23:7.
Sleator, Daniel Dominic, Tarjan, Robert Endre, 1985. Amortized efficiency of list update and paging rules. Commun. ACM, 28(2), 202–208.
Vishwanath, Arun, Sivaraman, Vijay, Thottan, Marina, 2009. Perspectives on router buffer sizing: recent results and open problems. Comput Commun Rev 39 (2), 34–39.
Tilman Wolf, Mark A. Franklin, 2000. Commbench—a telecommunications benchmark for network processors. In: ISPASS, pp. 154–162.
Wolf, Tilman, Pappu, Prashanth, Franklin, Mark A., 2003. Predictive scheduling of network processors. Comput Netw 41 (5), 601–621.