Balancing Work and Size with Bounded Buffers

Kirill Kogan*, Alejandro López-Ortiz†, Sergey I. Nikolenko‡,§, Gabriel Scalosub¶ and Michael Segal¶

* School of Computer Science, Purdue University, Email: kkogan@cs.purdue.edu
† School of Computer Science, University of Waterloo, Email: alopez-o@uwaterloo.ca
‡ Steklov Mathematical Institute, Email: sergey@logic.pdmi.ras.ru
§ National Research University Higher School of Economics, St. Petersburg, Russia
¶ Dept. of Comm. Sys. Eng., Ben-Gurion University of the Negev, Email: {sgabriel,segal}@cse.bgu.ac.il

***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.**

I. 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 [12]. 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 [17], 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. Furthermore, required processing characteristics on a network processor are often highly regular and predictable for a fixed configuration of network elements [18], 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.

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 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.

II. 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 I. Due to space constraints most proofs are omitted; they can be found in the full version of the paper [7].

III. Related Work

In recent years there has been a surge in the study of the effect of buffer size on the traffic queueing delay arising in the system. Appenzeller et al. [1] studied this problem in the context of statistical multiplexing, focusing mostly on TCP flows. More recently, broader aspects of these question were studied. A comprehensive overview of perspectives on router buffer sizing can be found in [16]. Keslassy et al. [6] were the first to consider buffer management and scheduling in the context of network processors, where arriving traffic has heterogeneous processing requirements. They study both FIFO with recycles and SRPT priority schedulers 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. They further introduced the notion of push-out costs which serves to balance the aggressiveness demonstrated by the the buffer management unit. We believe the assumption made in [6] 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. The work of Keslassy et al. as well as 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 Goldwasser [5] which provides an excellent overview of this field).

The SRPT algorithm has been studied extensively in OS scheduling for multithreaded processors, and it is well known to be optimal for mean response [14]. Additional objectives, models, and algorithms have been studied extensively in this context; see, e.g., [8]–[10]. A comprehensive overview of competitive online scheduling for server systems can be found in [11]; 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 [3], most notably to hybrid flow-shop scheduling [13], in scenarios where machines have bounded buffers. However, while these works focus on system delay, our main focus is system throughput.

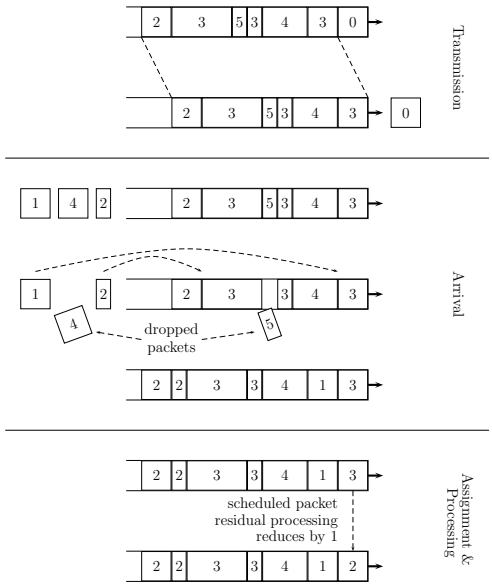


Fig. 1. An outline of the model. The top subfigure shows the transmission phase, the middle subfigure shows the arrival phase where packets might be discarded, and the bottom subfigure shows the assignment and processing phase. The length of a packet represents its size, and the number stamped on the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

IV. 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.[1] 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

[1]For a motivation why this information may be available, see [18]. Our assumption that the size may be as small as one byte is made for simplicity and can be viewed as a scaling assumption.

Algorithm	NPO, any priority	PO, SRPT priority	PO, LP priority	PO, MEP priority
Lower bound	kL	L	k	$1+(L\ln k-k-L)/B$
Upper bound	$k(L+1)$	$2L$	$(k+3)$	–

TABLE I. RESULTS SUMMARY: LOWER AND UPPER BOUNDS ON THE COMPETITIVE RATIO.

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. Figure 1 (see Appendix) 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 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 behaviour 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 [2], [15] 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 VII and IX. 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

```
if there is space available in the queue then
    accept p
end if
```

Algorithm 2 PO(p): Buffer Management Policy

```
accept p
while the last packet q in the buffer starts above position B − 2L + 1 do
    drop q
end while
```

V. USEFUL PROPERTIES OF ORDERED MULTISETS

In our proofs, we will make use of some properties of ordered (multi)sets in order to compare the performance of proposed algorithms with the optimal policy for various priority disciplines. In what follows, we consider multisets of real numbers, assuming that each multiset is ordered in non-decreasing order; we refer to such multisets as *ordered sets*. For every $1 \leq i \leq |A|$, we will further refer to element $a_i \in A$ or $A[i]$ as the i-th element in the set A in non-decreasing order. Given two ordered sets A, B, we say $A \geq B$ if $a_i \geq b_i$ for every i such that both a_i and b_i exist. The following lemma and its corollary will be needed as fundamental tools throughout our analysis; proofs can be found in [7].

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

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

VI. 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 (see [7] for proofs).

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

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

VII. 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 VI for all considered priorities.

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

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; for detailed proofs, see [7].

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).

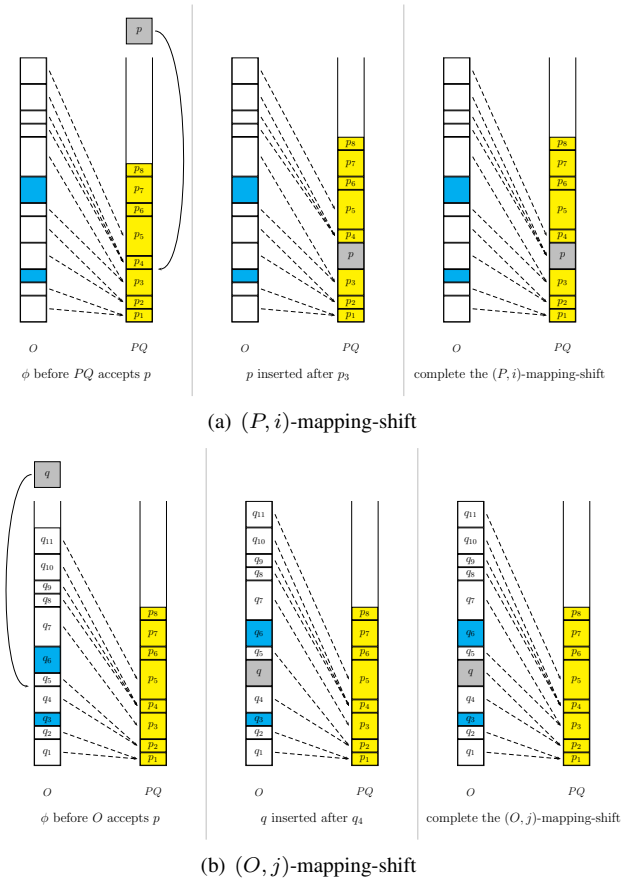
At time t, denote by M_t^O the ordered set of residual pass values for all non-type 1 OPT packets. All M_t^O values are grouped into blocks in the following way. A block is a minimal subset of consecutive M_t^O values starting from the lowest position that is not covered by any previous block, such that 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, the mapping remains unchanged.

A1 If some PO packets were dropped after p was accepted, clear all A1 mappings between these PO packets and their OPT mates. If p remains the i-th packet in PO's buffer, perform a (P,i)-*mapping-shift* (see Fig. 2(a) and [7] for more details): for each nonempty j-th block b and j-th PO packet q, $j \geq i$, clear the mapping to q by step A1 and map all packets of block b to q. If p is accepted by OPT to the j-th block, perform an (O,j)-*mapping-shift* (Fig. 2(b)): clear all A1 mappings between packets of the old l-th block and l-th PO packet (if both exist), $l \geq j$, recompute blocks starting from the j-th, map packets of l-th block to l-th 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.



(a) (P,i)-mapping-shift

(b) (O,j)-mapping-shift

Fig. 2. Example of the mapping used in Lemma 7 with mapping shifts used in the analysis. White OPT packets should be mapped, white PO packets are available for mapping. Blue OPT packets are of type 1.

Lemma 6: (1) The mapping is feasible. (2) The total length of packets in one block is at most $2L-1$.

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.

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 $N = \lceil \frac{B-2L+1}{2L-1} \rceil$.

The idea is to redistribute bytes mapped by step A2 between different PO packets. Let $N = \lceil \frac{B-2L+1}{2L-1} \rceil$. 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 j-th 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 Y-th byte assigned by step A2 be the l-th byte in the OPT buffer. Map each j-th 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 $N = \lceil \frac{B-2L+1}{2L-1} \rceil$.

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

VIII. Buffer Management with LP Priorities

We begin with a lower bound for the PO algorithm with LP-based priorities (see the proof in [7]) 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.

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

The mapping routine is unchanged during the transmission phase as in Section VII. 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 M_t^O 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 i-th packet in PO buffer at time t. An i-th 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.

Next we define a new step A1 where the blocks are recomputed after a (P,i)-*mapping-shift*. The rest of the proof of Theorem 12 can be found in [7].

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 i-th packet in PO buffer, perform a (P,i)-*mapping-shift*: clear all mappings by step A1 between packets of the old l-th block in OPT buffer and l-th packet in PO buffer, $l \ge j$, recompute blocks from j-th and map packets of l-th block to the l-th PO packet if both exist, $l \ge j$. If p is accepted by OPT to the j-th block, perform an (O,j)-*mapping-shift*: clear all mappings by step A1 between packets in OPT buffer of the old l-th block and l-th packet in PO buffer (if both exist), $l \ge j$, recompute blocks from j-th and map packets of l-th block to l-th PO packet if both exist, $l \ge j$.

IX. 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 VI. The following theorem (proven in the Appendix) provides a lower bound on the performance of the push-out MEP policy.

Theorem 13: PO with MEP-based priorities is at least $\left(1 + \frac{L \ln k - k - L}{B}\right)$-competitive.

Note that Theorem 13 provides a nontrivial lower bound for $L > \frac{k}{\ln k - 1}$. A nontrivial upper bound for PO with MEP priorities remains an interesting open problem.

X. Simulation study

A. 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 [4]) 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 Figures 3(a), 4(a), and 5(a), which demonstrate the throughput performance of the system, the y-axis represents the ratio between the throughput obtained by a 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 Figures 3(b), 4(b), and 5(b), which demonstrate the average latency in the system, the y-axis represents the average latency (in time slots) over all 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 $L = 10, 15, 25$.

B. Varying traffic intensity

Figure 3 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. Figure 3(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, Figure 3(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.

C. Increasingly heterogeneous processing requirements

Figure 4 shows the system performance as we allow packet processing requirement to increase, both in value and in variability. As demonstrated in Figure 4(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 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 X-B. This coincides with the intuition that the actual notion of load in the system is actually the product

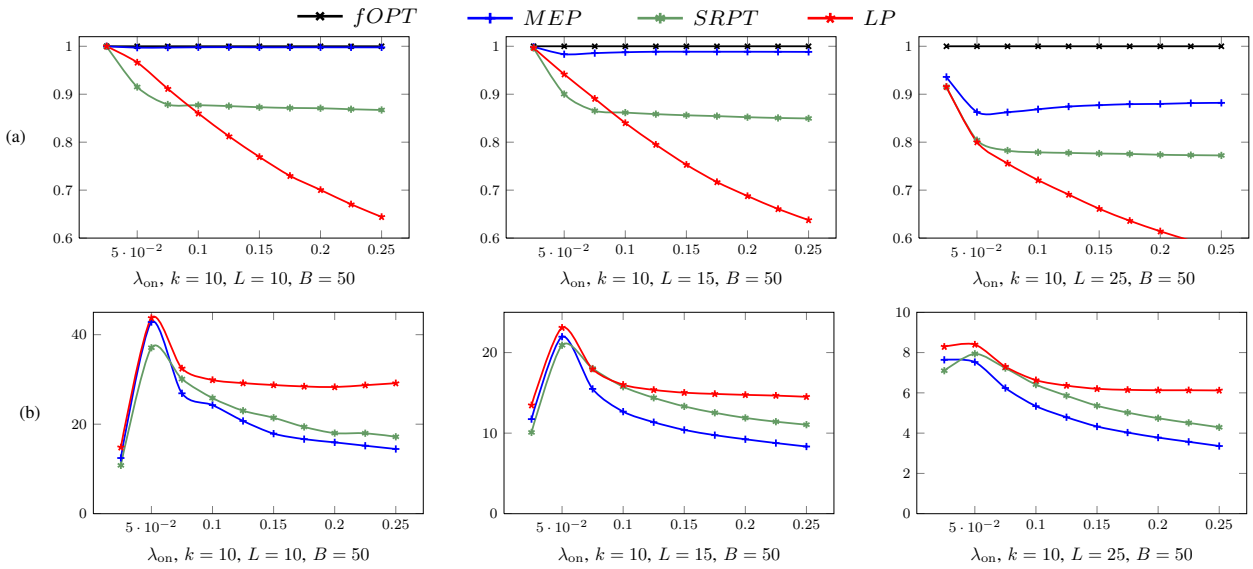


Fig. 3. Throughput performance (a) and latency (b) as a function of incoming stream intensity λ_{on} for three different values of maximal packet length L.

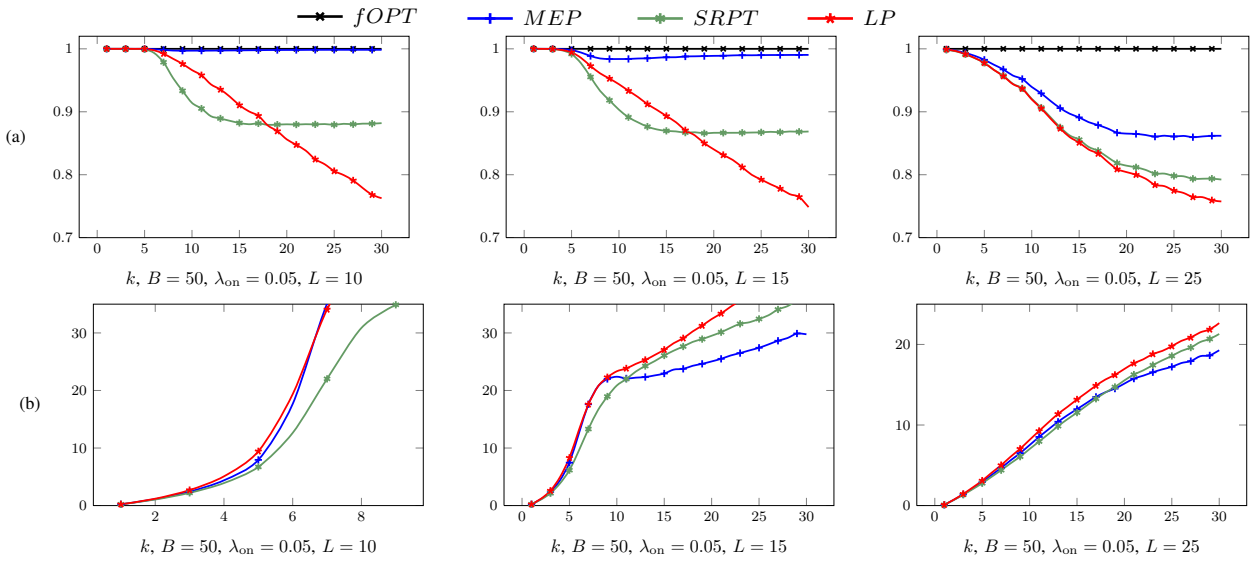


Fig. 4. Throughput performance (a) and latency (b) as a function of maximal required work k for three different values of maximal packet length L.

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 (Figure 4(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, once 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.

D. The effect of buffer size

Figure 5 shows the effect buffer size has on system performance. Fig. 5(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 an beyond a certain point, packet arrival rate is smoothed by the availability of buffer space. In terms of latency, Figure 5(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

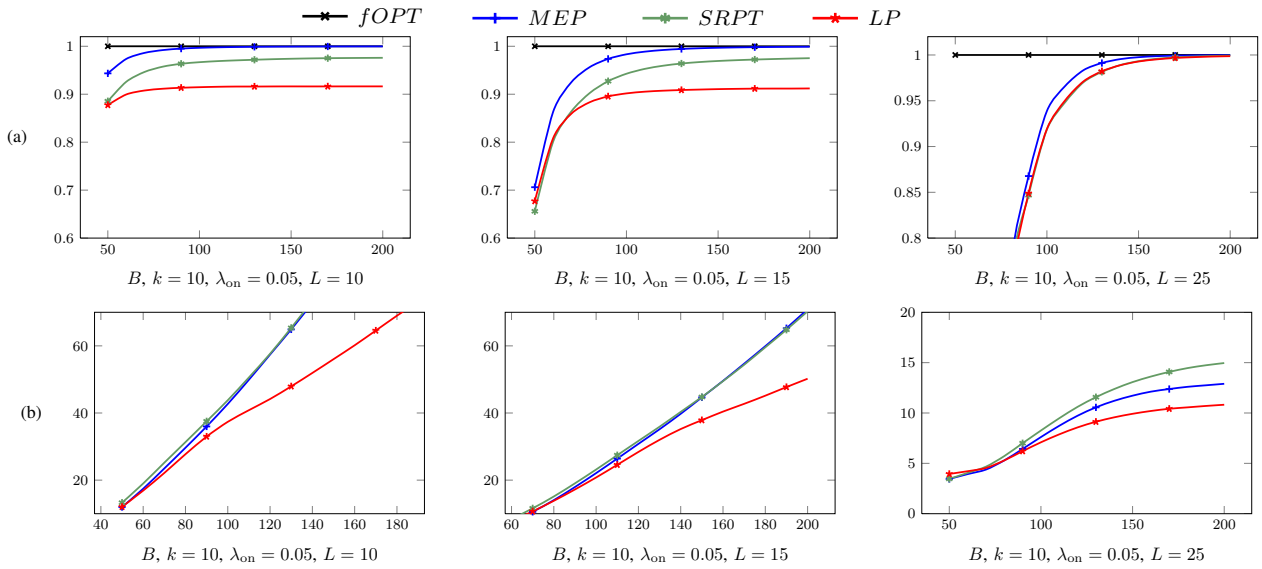


Fig. 5. Throughput performance (a) and latency (b) as a function of buffer size B for three different values of maximal packet length L.

packets, some of them must wait for their turn longer.

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

XI. 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 approach, are 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. Remaining open questions include closing the gaps between the upper and lower bounds (shown in Table I), and predominantly determining the performance of the PO algorithm with MEP priorities.

References

[1] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. In *SIGCOMM*, pages 281–292, 2004.

[2] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[3] Peter Brucker, Silvia Heitmann, Johann Hurink, and Tim Nieberg. Job-shop scheduling with limited capacity buffers. *OR Spectrum*, 28(2):151–176, 2006.

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

[5] Michael Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41(1):100–128, 2010.

[6] Isaac Keslassy, Kirill Kogan, Gabriel Scalosub, and Michael Segal. Providing performance guarantees in multipass network processors. In *INFOCOM*, pages 3191–3199, 2011.

[7] Kirill Kogan, Alejandro López-Ortiz, Sergey Nikolenko, Gabriel Scalosub, and Michael Segal. Balancing work and size with bounded buffers. *CoRR*, abs/1202.5755, 2012.

[8] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. In *STOC*, pages 110–119, 1997.

[9] Rajeev Motwani, Steven Phillips, and Eric Torng. Non-clairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.

[10] S. Muthukrishnan, Rajmohan Rajaraman, Anthony Shaheen, and Johannes E. Gehrke. Online scheduling to minimize average stretch. *SIAM Journal on Computing*, 34(2):433–452, 2005.

[11] Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.

[12] Ramaswamy Ramaswamy, Ning Weng, and Tilman Wolf. Analysis of network processing workloads. *Journal of Systems Architecture – Embedded Systems Design*, 55(10–12):421–433, 2009.

[13] Rubén Ruiz and José Antonio Vázquez-Rodrígue. The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1):1–18, 2010.

[14] Linus Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:687–690, 1968.

[15] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

[16] Arun Vishwanath, Vijay Sivaraman, and Marina Thottan. Perspectives on router buffer sizing: recent results and open problems. *Computer Communication Review*, 39(2):34–39, 2009.

[17] Tilman Wolf and Mark A. Franklin. Commbench – a telecommunications benchmark for network processors. In *ISPASS*, pages 154–162, 2000.

[18] Tilman Wolf, Prashanth Pappu, and Mark A. Franklin. Predictive scheduling of network processors. *Computer Networks*, 41(5):601–621, 2003.