Providing Performance Guarantees in Multipass Network Processors

Isaac Keslassy, Kirill Kogan, Gabriel Scalosub, Michael Segal

***Abstract*—Current network processors (NPs) increasingly deal with packets with heterogeneous processing times. In such an environment packets that require many processing cycles delay low latency traffic, because the common approach in today's NPs is to employ run-to-completion processing. These difficulties have led to the emergence of the Multipass NP architecture, where after a processing cycle ends, all processed packets are recycled into the buffer and re-compete for processing resources.**

In this work we provide a model that captures many of the characteristics of this architecture, and consider several scheduling and buffer management algorithms that are specially designed to optimize the performance of multipass network processors. In particular, we provide analytical guarantees for the throughput performance of our algorithms. We further conduct a comprehensive simulation study, which validates our results.

I. Introduction

A. Background

Multi-core Network Processors (NPs) are widely used to perform complex packet processing tasks in modern high-speed routers. NPs are able to address such diverse functions as forwarding, classification, protocol conversion, Deep Packet Inspection (DPI), intrusion detection, Secure Socket Layers (SSL), Network Address Translation (NAT), firewalling, and traffic engineering. Such tasks are usually referred to as *features*. They are often implemented using many processing cores. These cores are either arranged as a pool of identical cores (e.g., the Cavium CN68XX [10] or the AMCC nP7310 [3]), as a long pipeline of cores (e.g., the Xelerated X11 [33]), or as a combination of both (e.g., the EZChip NP-4 [13] or the Netronome NFP-32xx [25]).

Such architectures are very efficient for simple traffic mixes. However, following operator demands, packet processing needs are becoming more heterogeneous and rely on a growing number of more complex features, such as advanced Virtual Private Network (VPN) encryption (like IPsec-VPN and SSL-VPN), Lempel-Ziv-Stac (LZS) decompression, VoIP Session Border Controller (SBC), video Call Admission Control (CAC), per-subscriber queueing, and hierarchical classification for QoS [10], [12], [15], [23].

These features are increasingly challenging for traditional architectures, posing implementation, fairness, and benchmarking issues. First, longer and more complex features require either deeper pipeline lengths (e.g., 512 PISC processor cores in the Xelerated HX3XX series [33]) or longer processing times in run-to-completion cores. Second, a few packets with many features can delay, and even temporarily starve, the later packets. In fact, given limited high-speed buffering, this might lead to large drop rates upon congestion. This was illustrated in the *Christmas tree packet* DoS (Denial-of-Service) attack, in which each packet "lights up" several IP options processing bits [23]. Finally, and maybe more significantly, typical benchmarking tests used to rely on a simple stream of minimum-sized packets with only a basic IP forwarding service to measure the "worst-case throughput" of an NP [23], [28]. As benchmarking tests start to measure throughput given more advanced processing features, the impact of these features will be even more highlighted.

In view of the increasing impact of the packets with heavy features, another NP architecture has emerged as a leading alternative in the industry: the *Multipass NP* architecture. In this architecture, the processing time of a packet is divided into several time intervals, called *passes* or *cycles*. Intuitively, when a packet arrives to the NP, it is sent to a processing core. Then, after the core completes its processing pass, the packet is *recycled* into the set of packets awaiting processing. And so on, until all the packet passes are completed.

The main motivation for the multipass design results from the need to provide better QoS guarantees and traffic differentiation. Today, most NPs implement a *run-to-completion* paradigm. Therefore, heavy packets with long processing times until completion may significantly delay packets from low-latency flows, possibly resulting in the starvation of such flows. The multipass NP architecture is targeted at tackling this problem, by ensuring a partial and dynamic *packet isolation* property that can shield low-latency flows from heavy packets. In addition to the fact that it avoids the run-to-completion paradigm, it also does not require the NP designer to define a large pipeline length in advance. This is especially useful for NPs with different possible markets. Furthermore, note that in multipass NPs, actually recycling packets would involve complex interconnections and large buffers. Therefore, to decrease the cost of recycling, packets practically stay buffered and small control messages go through recycling instead.

This NP architecture with recycling has for instance been

A preliminary version of this paper was presented at IEEE Infocom, 2011. The authors would like to thank Isask'har (Zigi) Walter for his comments. This work was partly supported by US Air Force European Office of Aerospace Research and Development, grant FA8655-09-1-3016, Deutsche Telecom, European project FLAVIA, Israeli Ministry of Industry, Trade and Labor (consortium CORNET), as well as by the European Research Council Starting Grant No. 210389.

I. Keslassy is with the Dept. of Electrical Engineering, Technion, Haifa, Israel. Email: isaac@ee.technion.ac.il.

K. Kogan is with Computer Science Dept., Waterloo University, Canada. Email: kirill.kogan@gmail.com.

G. Scalosub and M. Segal are with the Dept. of Communication Systems Engineering, Ben Gurion University, Beer-Sheva, Israel. Email: {sgabriel,segal}@bgu.ac.il.

implemented in the recent Cisco QuantumFlow NP [12]. Forming the heart of Cisco's most recent ASR 1000 edge routers, this 40-core NP might become the most widespread among high-speed routers. Also, although not strictly multipass NP architectures, several NP architectures in the literature already allow for recycling of complex packets, such as [31] for IP control packets, and [16] mostly for MPLS packets using a folded pipeline architecture.

Given a heterogeneous set of packet processing times, the *scheduler* plays a significant role in the multipass NP architecture. This is because it should make sure that heavy packets with many passes do not monopolize the cores and starve packets with fewer passes.

To the best of our knowledge, despite the emergence of the multipass NP architecture, there has not yet been any analysis of its scheduler performance in the literature. In particular, NP schedulers are typically *designed for the worst-case throughput* to support a guaranteed wire rate (see Section 2.2 in [28]). But little is known regarding the worst-case throughput of the various possible multipass NP schedulers.

B. Our Contributions

In this paper, we analyze the performance of scheduling and buffer management policies in multipass NPs, and provide guarantees as to their worst-case throughput. Our solutions enable dealing with the various requirements posed to the scheduler (such as delay, throughput, and implementation complexity), and illustrate tradeoffs as to the scheduler's ability to fulfill these requirements. Our analysis also makes it possible for the designer of future multipass NPs to have analytical worst-case guarantees on the NP performance, which are applicable to traffic with complex processing needs.

We consider settings where each arriving packet requires some number of processing passes, and study the interplay of three factors:

- *The scheduling policy*: we study both FIFO buffers, and Priority Queues (where priority is determined by the number of remaining passes required).
- *The buffer management policy:* we design and evaluate both preemptive policies (where packets residing in the buffer can be discarded), and non-preemptive policies.
- *The implementation cost:* Our model allows for a copying cost of packets into the buffer that reflects the impact multiple accesses to the buffer have on the system's throughput.

We design and analyze algorithms that aim at maximizing the overall value obtained by the system, which is affected by both the packet-level throughput (considered as benefit) and the copying cost (considered as penalty). We note that the concept of copying cost used in our model may be applied to capture various types of "costs" related to admitting a new packet to the buffer, including processing, power consumption, latency, memory access bandwidth, etc. In particular, we shall see that this cost may serve as a parameter that can be tuned by the operator in order to balance between various aspects pertaining to the system's performance. We further note that our model can also be used to model cold-cache penalties. A detailed description of our model is given in Section II.

For our analytical results, we use competitive analysis to evaluate the performance of our proposed policies. For the case where no copying cost is incurred, we design and analyze buffer management algorithms for both FIFO- and PQ-based environments. We show that non-preemptive architectures may suffer form extremely large performance degradation compared to the optimal performance possible. On the other hand, we prove that natural buffer management policies for PQ-based environments are optimal when preemption is allowed, and further show that FIFO-based environments endowed with preemption, although they are not optimal, can obtain a reasonable guaranteed throughput compared to the optimal performance possible, which depends only on the maximum number of passes a packet requires. These results are presented in Section III. For the case where the system incurs a strictly positive copying cost, we devise competitive buffer management algorithms for PQ-based environments, and provide an elaborate analysis of their performance guarantees. These results are presented in Section IV.

To complete our study, we present a simulation study that further validates our results and provides additional insights as to the performance of multicore NPs. Specifically, our results show that the design criteria governing our algorithms, which are intended to optimize towards the worst-case scenario, exhibit very good performance also for simulated average-case traffic. In addition, our simulation study shows that the number of available cores has a striking non-trivial effect on the performance of the various policies we propose. These results are presented in Section V.

Our work gives rise to a multitude of questions and possible extensions. We discuss these further in Section VI.

C. Related Work

As mentioned above, recycling is not new in NPs and has previously appeared in the literature, especially for particularly complex packets that cannot be processed using a typical pipelining scheme For instance, in the Open Network Lab's NP [31], the XScale recycles IP control packets and other exceptional packets, and also enables the use of plugins for special processing. Multipass NP power consumption in a folded pipeline architecture has also been a previous topic of study [16]. However, to our knowledge, there is no previous work in the literature that discusses the scheduling and buffer management policies in multipass NPs. Finally, [8] further studies the impact of caches on performance. However, none of these papers considers the impact of recycling in their models. Moreover, no paper analyzes the impact of the packet admission control policy on the worst-case NP performance.

There is also a long history of OS scheduling for multithreaded processors. A comprehensive overview of competitive online scheduling for server systems is provided in [26]. For instance, the SRPT (Shortest Remaining Processing Time) algorithm always runs the job with the least amount of remaining processing time, and it is well known that SRPT is optimal for average response [24]. Additional objectives, models, and algorithms have been studied extensively in this context (e.g., [7], [21], [24], to name but a few). There has been significant interest in the research community in addressing shared resource contention on multi-threaded processors. Conceptually, contention-aware schedulers map threads to cores in such a way as to minimize shared resource contention amongst the threads and improve performance. Contention-aware thread schedulers have been studied recently in [6], [20], [30], [34], and have shown significant improvement in throughput, fairness, and predictability over contention-unaware schedulers. Another related line of research that takes into account some notion of "throughput" in the context of OS scheduling focuses on assigning jobs to processors with fluctuating speeds, which are not known precisely to the scheduler [11]. When comparing this body of research with the framework of NPs one should note that OS scheduling is mostly concerned with average response time, average slowdown, etc., while NP scheduling is targeted at providing (worst-case) guarantees on the throughput. In addition, NP scheduling is unique in that it inherently has a limited-size buffer.

Another large body of research related to our work focuses on competitive packet scheduling and buffer management, mostly for various switching architectures, such as Output-Queued (OQ) switches (e.g., [1], [18]), shared memory switches with OQs (e.g., [14], [19]), and merging buffers (e.g., [17]). This body of work has also studied more general multi-queue switches (e.g., [2], [4], [5]).

II. Model Description

A. Multipass NP Architecture

Figure 1 illustrates the multipass NP architectural model used in this paper. It is a simplified model of the Cisco QuantumFlow NP architecture [12]. The three major modules in our model are: (a) the *Input Buffer* (IB), (b) the *Scheduler Module* (SM), and (c) a set of C cores or *Packet Processing Elements* (PPEs).

First, the *IB module* is used to buffer incoming packets. The IB holds a buffer that can contain at most B packets. It obeys a given Buffering Model (BM), as defined later. Second, the *SM module* has two main functionalities in our model: the *buffer management*, as later described, and the *assignment of packets to PPEs*, by binding each PPE with its corresponding IB packet. Each *PPE element* is a processing core that works on a specific packet stored in the IB for one cycle (predefined period of time), also referred to as a time slot. For simplicity we assume that each PPE is single threaded.

We divide time into discrete time slots, where each step consists of four phases: (i) *transmission*, in which completed packets leave the NP and incomplete control packets for those with remaining passes are recycled, (ii) *arrival*, in which the SM performs its buffer management task considering newly arrived packets and recycled control packets (observe that recycled control packets are admitted to IB before new arrivals), (iii) *scheduling*, in which C head-of-queue packets are designated for processing, and (iv) *processing*, in which the SM assigns a designated packet to each PPE, and packet processing takes place.

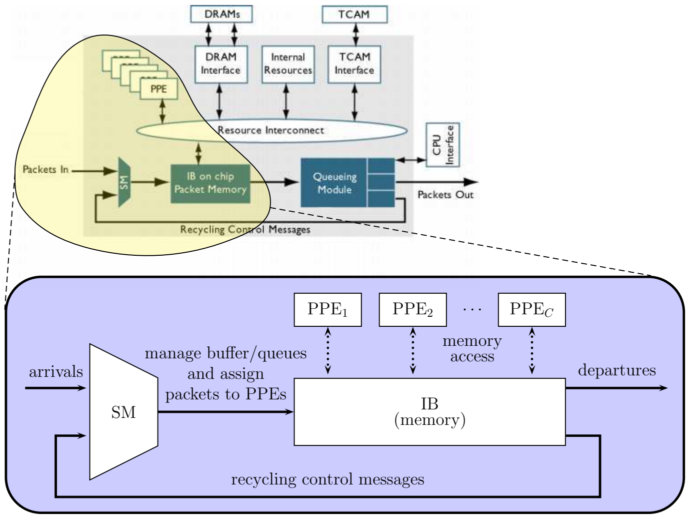


Fig. 1. An outline of the architecture model, as an abstraction of a standard Multipass NP Architecture (see, e.g. [12]).

We assume arbitrary packet arrival (i.e., it is not governed by any specific stochastic process, and may even be adversarial). We also assume that all packets have unit size. Each arriving packet p is further stamped with the number of *passes* it requires from the NP, denoted $r(p)$. This number is essentially the number of times the packet should be assigned to a PPE if it is to be successfully delivered. The availability of this information relies on [32], which shows that "processing on an NP is highly regular and predictable. Therefore it is possible to use processing time predictions in admission control and scheduling decisions." In practice, this number of passes $r(p)$ might only be an approximation, or only be known after the first pass, even though we assume it in this paper to be known from the start. In terms of maintaining packet order within a given flow[1], we note that other modules within the NP architecture are responsible for handling any reordering issues that may arise (see, e.g., [22], [27]).

B. Problem Statement and Objectives

In the NP multipass architecture, *new packets incur higher costs than recycled packets*. New packets admitted to the buffer monopolize part of the memory link capacity to enter the memory, and therefore require more capacity in the memory access implementation of an NP. Each new packet also needs to update many pointers and associated structures at link speeds. These costs are substantially higher than the costs associated with recycled control packets corresponding to packets already stored in the buffer.

To reflect the value of throughput, we assume that each departed packet has unit value. However, to reflect the cost of admitting new packets, each newly admitted packet is also assumed to incur a fixed *copying cost* cost of $\alpha \in [0, 1)$ for copying it to IB. Finally, we measure the final value as the total throughput value minus the total copying cost.

[1]By *flow* we mean any subsequence of packets associated by the NP to correspond to a single logical connection. For instance, a sequence identified by the common header 5-tuple.

Any specific architecture corresponding to our model can be summarized by a 4-tuple (B, BM, α, C), where B denotes the buffer size available for IB, BM is the buffering model (in this paper it will usually be PQ or FIFO), α is the copying cost, and C is the number of available PPEs.

Our objective is the following: *given a* (B, BM, α, C)*-architecture, and given some finite arrival sequence, maximize the value of successfully delivered packets.*

For the case where $\alpha = 0$, the overall value of successfully delivered packets is equal to the system's packet-level throughput. For the case where $\alpha > 0$ the overall value of successfully delivered packets equals the throughput minus the overall copying cost incurred by admitting packets to IB.

Our goal is to provide performance guarantees for various scheduling and buffer management algorithms. We use competitive analysis [9], [29] to evaluate the performance guarantees provided by online algorithms. An algorithm ALG is said to be *c-competitive* (for some $c \geq 1$) if for any arrival sequence σ, the overall value of packets successfully delivered by ALG is at least $1/c$ times the overall value of packets successfully delivered by an optimal solution, denoted OPT, obtained by a possibly offline clairvoyant algorithm, which successfully delivers the maximum number of packets that is possible for the given arrival sequence.

C. Further Notation and Algorithmic Framework

We will define a *greedy* buffer management policy as a policy that accepts all arrivals whenever there is available buffer space (in the IB). Throughout this paper we only look at *work-conserving* schedulers, i.e. schedulers that never leave a processor idle unnecessarily.

We will say that an arriving packet p *preempts* a packet q that has already been accepted into the IB module iff q is dropped and p is admitted to the buffer instead. A buffer management policy is called *preemptive* whenever it allows for preemptions.

For any algorithm ALG and any time-slot t, we define IB_t^{ALG} as the set of packets stored in IB of algorithm ALG at time t.

We assume that the original number of passes required by any packet is in a finite range $\{1, \dots, k\}$. The value of k will play a fundamental role in our analysis. We note, however, that none of our algorithms needs to know k in advance.

The number of *residual passes* of a packet is key to several of our algorithms. Formally, for every time t, and every packet p currently stored in IB, its number of residual passes, denoted by $r_t(p)$, is defined to be the number of processing passes it requires before it can be successfully delivered.

Most of our algorithms will take the general form depicted in Algorithm 1, where the specific subroutine determining whether or not preemption takes place will depend on the algorithm. We will focus our attention on two natural BMs:

1) FIFO: In this policy packets are served in FIFO order, i.e. the C head-of-line packets are chosen for assignment to the PPEs. Upon completion of a processing round by the PPEs, all the packets that have been processed in this round and still require further processing passes are queued at the tail of the IB queue.
2) Priority Queueing (PQ): In this policy packets are served in non-increasing order of residual passes, i.e., C packets with the minimum number of residual passes are chosen for assignment to the PPEs in every time slot.

We assume that the queue order is also maintained according to the BM preference order.

The generic algorithmic setting for the buffer management policy of the SM is defined in Algorithm 1. The specific algorithms discussed in the following sections will differ according to the decision made by the DecideIfPreempt procedure, which decides what packet to discard in case of overflow.

Algorithm 1 ALG: Buffer Management Policy
```
1: upon the arrival of packet $p$:
2: if the buffer is not full then
3:     accept packet
4: else
5:     DecideIfPreempt(ALG,$p$)
6: end if
```

III. Buffer Management with no Copying Cost ($\alpha = 0$)

A. Non-preemptive Policies

In this section we consider *non-preemptive greedy buffer management policies.* Essentially, the subroutine DecideIfPreempt for such policies simply rejects the pending packet. The following theorem provides a lower bound on the performance of such non-preemptive policies for *FIFO schedulers.*

Theorem 1. *The competitive ratio of any non-preemptive greedy buffer management policy* $(B, FIFO, 0, C)$*-system is at least* $\frac{k}{C}$*, where* k *is a maximal number of passes required by any packet.*

Proof: Assume for simplicity that B/C is an integer. Consider the following set of arrivals. During the first time slot arrive B packets with maximal number of passes k. Since online algorithm A is greedy it accepts all of them. OPT does not accept these packets. During the next kB/C time slots the buffer of A is full since it is FIFO and non preemptive. During this time interval arrive kB/C packets with a single required pass and all of them are transmitted by OPT. ■

The following theorem provides a lower bound for *PQ schedulers.*

Theorem 2. *The competitive ratio of any non-preemptive greedy buffer management policy* $(B, PQ, 0, C)$*-system is at least* $k-1$*, where* k *is a maximal number of passes required by any packet.*

Proof: We will show more specifically that the competitive ratio is at least $(k-1)(1-\varepsilon)$, for any $\varepsilon > 0$. Assume for simplicity that k/C is an integer. At time 0 we have the arrival of B packets, each requiring k passes, and at time 1 we have the arrival of $(k-1)C$ packets, each requiring a single pass. Consider the sequence of time slots $a_i = ik - 1$, for $i = 1, \dots, \ell$. At any time a_i we have the arrival of C packets, each requiring k passes. At any time a_i+1 we have the arrival of $(k-1)C$ packets, each requiring a single pass.

We now turn to analyze the performance of any greedy PQ policy given the above arrival sequence. At time 0 all B packets are accepted (due to greediness), and none of the arrivals at time 1 can be accommodated into the buffer. It is easy to show by induction by the arrival of packets at time a_1, C of the packets accepted by at time 0 are delivered, and the buffer has room to accommodate the newly arriving C packets at time a_1, and none of the packets arriving at time a_1+1 can be accommodated into the buffer. It is therefore easy to show by induction that for any $i=1,\dots,\ell-1$, the number of packets delivered by the algorithm between a_i and a_{i+1} is exactly C, the algorithm accepts all packets arriving at a_{i+1}, and cannot accept any of the packets arriving at time $a_{i+1}+1$. This gives an overall throughput of at most $B+\ell C$ packets.

Let us now turn to describe a feasible solution whose throughput serves as a lower bound on OPT . This solution would accept $B-(k-1)C$ packets that arrive at time 0, $(k-1)C$ packets that arrive at time 1, and for every $i=1,\dots,\ell$, accepts the packets that arrive at time a_i+1. We first show that the number of packets residing in the buffer of this solution never exceeds the buffer capacity of B. By definition, the overall number of packets accepted by time 1 is B. Furthermore, the algorithm delivers all packets that arrived at time 1 by time $1+(k-1)=k=a_1+1$ (since each of them requires a single pass), and can therefore accommodate the newly arriving packets at time a_1+1. It is easy to show by induction that for any $i=1,\dots,\ell-1$, the number of packets delivered by this solution between a_i+1 and $a_{i+1}+1$ is exactly $(k-1)C$, and the solution can accept all packets arriving at $a_{i+1}+1$. It follows that this solution has an overall throughput of $B-(k-1)C+\ell(k-1)C$.

Combining the above we obtain that the competitive ratio of any greedy buffer management policy in a $(B,PQ,0,C)$-system is at least

$$\frac{B-(k-1)C+\ell(k-1)C}{B+\ell C}=\frac{B+(\ell-1)(k-1)C}{B+\ell C}$$

which tends to $k-1$ as ℓ grows to infinity, thus completing the proof.

■

Next, we demonstrate upper bounds of two specific non-preemptive policies that are based on queueing model $X\in\{PQ,FIFO\}$.

Theorem 3. *X is $k_a/k_{\min}$-competitive in a $(B,X,0,C)$-system, where k_a is an average number of required passes of the n packets accepted by X, and $k_{\min}$ is an average number of passes of the n packets with the least number of required passes in the input sequence σ.*

Proof: We can assume without loss of generality that there is an optimal algorithm OPT never preempts an accepted packet. We will omit ∞ from the name of X_∞ algorithm for simplicity of the future notation. For any algorithm ALG, denote by $W(\mathrm{ALG},\sigma)$ the amount of processed work (in passes) made by algorithm ALG, given input sequence σ. In what follows we will usually omit the σ notation and simply refer to $W(\mathrm{ALG})$.

We first claim that that $W(OPT)\le W(X)$. For this purpose we will define two reference algorithms (and virtual inputs) OPT^* and X^* whose behavior are defined by the next template ALG^*. For every packet accepted to the buffer of ALG or recycled by ALG, ALG^* accepts a single packet requiring one pass. ALG^* is work conserving, and therefore always processes (and transmits) a packet on each available core whenever the buffer is not empty. Note that at any time slot t, both ALG^* and ALG buffers contain the same amount of packets. Moreover, The number of packets transmitted by ALG^* is exactly $W(\mathrm{ALG})$. It therefore suffices to show that OPT^* does not outperform X^*.

Assume by contradiction that OPT^* does outperform X^*. Clearly in such a case there must be at least one time slot in which OPT^* transmits more packets than X^*. Let t be the minimal such time slot. It must be the case that there is an arrival phase that occurred at some time prior to t where OPT^*'s buffer contains more packets than X^* buffer. Let $t'<t$ be the minimal such time. The algorithm X^* does not accept a new virtual packet to its buffer only if X is idle (i.e., X does not accept new packets, nor recycles any packets). It must be the case that during t', OPT^* accepts some packets to its buffer (by minimality of t' and the fact that OPT^* is work conserving). It therefore follows that during t' the set of arriving packets is not empty. Since during t', X does not accept all virtual arrivals, its buffer is full at the end of arrival phase t' (since X is greedy). Recall that at any time slot, the buffer of X^* and X contain the same amount of packets. Hence, X^* buffer is also full at the end of arrival phase t'. From the other side at the end of arrival phase t', OPT^* buffer contains more packet than its X^* mate. Since X^* and OPT^* buffers of the same size, we get a contradiction, thus establishing that $W(OPT)\le W(X)$.

Let $n(\mathrm{ALG},\sigma)$ be a number of packets that are accepted by algorithm ALG. When the arrival sequence σ is clear from the context, we will simply use n^{ALG} to denote this number. Since OPT never preempts n^{OPT} is the number of packets transmitted by OPT. Denote by k^{ALG} the average number of required passes of packets accepted by ALG. It follows that for any nonpreemptive algorithm, $n^{\mathrm{ALG}}=W(\mathrm{ALG})/k^{\mathrm{ALG}}$. Let k' denote the average number of passes required by the n^{OPT} with the least number of passes in the input sequence. Clearly, $k'\le k^{OPT}$. Since $n^X\le n^{OPT}$, we have $k_{min}\le k'$ (recall that $k_{\min}$ is an average number of passes of the n^X packets with the least number of required passes in the input sequence σ). $n^{OPT}\le W(OPT)/k_{\min}$. Combining this with our previous claim we obtain that $n^{OPT}\le W(OPT)/k_{\min}\le W(X)/k_{\min}=n^Xk^X/k_{\min}$, thus completing the proof. It should be noted that the above argument is independent of C, and therefore our claim holds for any $C>1$. ■

As demonstrated by the above results, the simplicity of non-preemptive greedy policies does have its price. In the following sections we explore the benefits of introducing preemptive policies, and provide an analysis of their performance.

B. Preemptive Policies

For the case where $\alpha = 0$, we focus our attention on the intuitive rule for preemption, which states that a newly arrived packet p should preempt a buffered packet q at time t if $r_t(p) < r_t(q)$. This rule is formalized in Algorithm 2, which gives a formal definition of the DECIDEIFPREEMPT procedure of Algorithm 1.

Algorithm 2 DECIDEIFPREEMPT(ALG,p)

```
1: $i \leftarrow$ first packet in $IB_t^{ALG}$ s.t. $r_t(p_i) = \max_{i'} \{r_t(p_{i'})\}$
2:                              ▷ first in the order implied by the BM
3: if $r(p) < r_t(p_i)$ then
4:     drop $p_i$ and accept $p$
5: else
6:     reject $p$
7: end if
```

In what follows we consider the performance of the above preemption rule for two specific BMs, namely: $ALG \in \{PQ, FIFO\}$.

1) Preemptive Priority Queueing: In this section we study the performance of a BM implementing PQ, where priorities are set in accordance with the non increasing order of residual passes.[2] We refer to this algorithm as PQ_1.[3] The following theorem provides some guarantee as to its performance. As mentioned earlier, although SRPT was shown to be optimal when one tries to minimize the average response time, the objective in our case is different and those results do not readily apply to our settings.

Theorem 4. *PQ_1 is optimal in a $(B, PQ, 0, C)$-system.*

Proof: Assume some arrival sequence σ, and let OPT denote an optimal solution for σ. For every time t, every algorithm A, and every integer $\ell \in \{1, \ldots, |IB_t^{OPT}|\}$, let $P_t^A(\ell)$ denote the set of ℓ head-of-queue packets in IB_t^A (recall we can assume without loss of generality that packets in IB are ordered according to A's buffering model). Consider algorithm PQ_1. Recall that for any such ℓ, the head-of-line packets in $IB_t^{PQ_1}$ have the minimal number of residual passes. Consider the following volume function

$$\Phi_t^A(\ell) = \sum_{p \in P_t^A(\ell)} r_t^A(p)$$

where $r_t^A(p)$ denotes the residual number of passes of packet p in IB_t^A. $\Phi_t^A(\ell)$ measures the amount of remaining work required for processing the ℓ head-of-queue packets in the buffer of A at time t.

We will prove that for any time t and any integer $\ell \in \{1, \ldots, |IB_t^{OPT}|\}$,

$$\Phi_t^{PQ_1}(\ell) \leq \Phi_t^{OPT}(\ell), \tag{1}$$

i.e., the amount of residual work necessary by PQ_1 for processing the ℓ head-of-queue packets is at most that required by OPT . Note that it is sufficient to consider ϕ as defined at the end of each time step, although the inequality holds also after each of the phases within a time step. By proving that the inequality in Equation (1) holds for any ℓ, we would obtain in that

$$\phi_t^{PQ_1}(1) \leq \phi_t^{OPT}(1),$$

which implies that whenever OPT processes a 0-residual passes packet, so does PQ_1. Therefore, if Equation (1) holds, the throughput of PQ_1 is at least that of OPT , completing the proof.

We now turn to prove Equation (1). First note that without loss of generality we can assume that OPT is both work-conserving, i.e., never idles when its buffer is non-empty, and also non-preemptive. Since PQ_1 does not perform any admission control, and always accepts packets when it has room, then at any time t, $IB_t^{PQ_1} \geq IB_t^{OPT}$.

The proof follows by induction on t. For $t = 0$, the claim clearly holds since by definition PQ_1 accepts the maximal size set of packets of minimal passes (up to the buffer capacity limit B) among all arrivals at $t = 0$, and stores them in non-decreasing order of residual passes. Assume the claim holds for $t' < t$, and consider time t. We will show the inequality holds after each of the phases within time step t. For the transmission phase, note that by the induction hypothesis, for every ℓ $\Phi_{t-1}^{PQ_1}(\ell) \leq \Phi_{t-1}^{OPT}(\ell)$, and since both PQ_1 and OPT are work-conserving, both sides of the inequality reduce by the amount of processing done in the transmission phase (the same for both PQ_1 and OPT , unless the buffer of OPT becomes empty, in which case the inequality is trivially true since we are only concerned with $\ell \leq |IB_t^{OPT}|$, and $|IB_t^{OPT}| = 0$ in this case). This implies that the inequality holds after the transmission phase. Consider now the arrival phase within time step t. We will consider the buffer management decisions made by PQ_1 as if they were done in a series of phases. In the first phase, assume PQ_1 retains in its buffer $P_t^{PQ_1}(|IB_t^{OPT}|)$, and let $R_t^{PQ_1}$ denote the remaining packets in its buffer, i.e., packets in positions greater than $|IB_t^{OPT}|$ that are currently in $IB_t^{PQ_1}$ (which are set aside at this point). In the second phase, assume PQ_1 accepts the set of packets A_t^{OPT}, the set of packets accepted by OPT at time t (recall that by our assumption OPT is non-preemptive, hence by the fact that OPT does not overflow, PQ_1 also has room to accept these packets in this phase). Denote by $\widetilde{IB}_t^{PQ_1} = P_t^{PQ_1}(|IB_t^{OPT}|) \cup A_t^{OPT}$. For this buffer occupancy (when sorted by non-decreasing residual passes order), by the induction hypothesis on $P_t^{PQ_1}(|IB_t^{OPT}|)$, the inequality holds since the added packets are exactly those accepted by OPT at time t. Note that $\left|\widetilde{IB}_t^{PQ_1}\right| = |IB_t^{OPT}|$, where IB_t^{OPT} is considered at the end of time step t. Now let PQ_1 consider the remaining packets pending, by first considering packets in $R_t^{PQ_1}$, and afterwards considering any additional packets that arrived at time t (and were not accepted by OPT). Let $IB_t^{PQ_1}$ denote the buffer of PQ_1 after this sequence of events. First note that this sequence mimics exactly the behavior of PQ_1 (up to accepting packets that are equivalent to those accepted by OPT). Let p_ℓ ($\tilde{p}_\ell$) denote the packet in position ℓ in $IB_t^{PQ_1}$ ($\left|\widetilde{IB}_t^{PQ_1}\right|$). By the priority-based preemption rule of PQ_1, for every position $\ell \in \{1, \ldots, |IB_t^{OPT}|\}$, $r_t(p_\ell) \leq r_t(\tilde{p}_\ell)$.

[2]Packet p has a higher priority than packet q at time t if $r_t(p) < r_t(q)$.

[3]The reason for choosing the subscript 1 would become clear in section IV.

This follows from the fact that the queues are ordered in non-decreasing order of residual passes, and the candidate packets considered by PQ_1 that resulted in the buffer configuration $IB_t^{PQ_1}$ is a superset of $\left|\widetilde{IB}_t^{PQ_1}\right|$. It therefore follows that for any $\ell \in \{1, \ldots, |IB_t^{OPT}|\}$, $\Phi_t^{PQ_1}(\ell) \leq \Phi_t^{OPT}(\ell)$, thus completing the proof. ■

The above theorem provides concrete motivation for using a priority queuing buffering model. It also enables using PQ_1 as a benchmark for optimality.

On the other hand, priority queueing has many drawbacks in terms of the difficulty in providing delay guarantees and in terms of implementation. For instance, low-priority packets may be delayed arbitrarily for an arbitrarily long amount of time due to the steady arrival of low-priority packets. Therefore it is of interest to study BMs that ensure such scenarios do not occur. One such predominant BM is using FIFO queueing, which is discussed in the following section.

2) Preemptive FIFO: In this section we analyze the preemptive policy depicted in Algorithm 2, where the BM implements FIFO queueing. We refer to this algorithm as $FIFO_1$. FIFO has many attractive features, including bounded delay, and it is easy to implement. We first begin with providing the counterpart to Theorem 4, which shows that as opposed to priority queueing, the performance of $FIFO_1$ can be rather far from optimal.

Theorem 5. *$FIFO_1$ has competitive ratio $\Omega(\frac{\log k}{C})$ in a $(B, FIFO, 0, C)$-system, for $k = \Theta(\frac{B}{C})$.*

Proof: Assume for simplicity that B/C is an integer, and further assume that $k+1 = \frac{B}{C}$. Consider the following arrival sequence: for $i = 0, \ldots, k$ we have B packets with $k-i$ required passes arriving at time $t_i = iB/C$.

Let us first consider the performance of $FIFO_1$ given the above arrival sequence. At time t_0 $FIFO_1$ accepts B packets, each with k required passes. Call this set A. It is easy to see that for every $i = 1, \ldots, k$ at time t_i all the packets in A are still in $FIFO_1$'s buffer, and each has $k-i$ residual passes. Hence, $FIFO_1$ never has a reason to preempt any of the packets in A. It follows that at time t_k the buffer holds B packets with 1 residual passes and can eventually only deliver B packets.

We now turn to consider the performance of an optimal policy for the above arrival sequence. We first show a policy that delivers $(1 + \frac{1}{C})B - 1$ packets out of the above arrival sequence (implying a lower bound of $1 + \frac{1}{C} - \frac{1}{B}$ on the competitive ratio). We then refine our analysis to prove the required result. We henceforth start by considering the conservative policy that for any $i = 0, \ldots, k-1$ accepts a single packet at time t_i, and further accepts all B packets arriving at time t_k. First note that the above policy is feasible: since for any i, $t_{i+1} - t_i = \frac{B}{C} \geq k+1$, if a policy accepts a single packet at time t_i, then this packet will be delivered by time t_{i+1}. This implies that the buffer is empty at time t_{i+1}, implying in turn the feasibility of the above policy. Since the policy accepts $k = \frac{B}{C} - 1$ packets by time t_k, and B packets at time t_k, we have a total throughput of $(1 + \frac{1}{C})B - 1$ as required.

Let us now turn to refine our analysis, and present a better policy that implies the required result, and is based upon the simple policy just described. Recall that at time t_i, the buffer is empty, and we have B packets with $k-i$ required passes arriving. Our new policy accepts $\lfloor \frac{B}{C(k-i+1)} \rfloor$ of these packets. We will show that this new policy ensures that the buffer is empty just before the arrival phase at any time t_{i+1}. The overall number of time steps required to deliver a set of $\lfloor \frac{B}{C(k-i+1)} \rfloor$ packets, each requiring $k-i$ passes, is $\lfloor \frac{B}{C(k-i+1)} \rfloor \cdot (k-i+1) \leq \frac{B}{C} = t_{i+1} - t_i$, which implies that by time t_{i+1} the buffer is indeed empty. Note that since $k = \frac{B}{C} - 1$, $\lfloor \frac{B}{C(k-i+1)} \rfloor \geq 1$ for every $i = 0, \ldots, k$. We can now evaluate the performance of this new policy. The overall number of packets accepted (and delivered) by the policy is

$$\begin{array}{rcl} \sum_{i=0}^{k} \lfloor \frac{B}{C(k-i+1)} \rfloor & \geq & \sum_{i=0}^{k} (\frac{B}{C(k-i+1)} - 1) \\ & = & \frac{B}{C} \sum_{j=1}^{k+1} \frac{1}{j} - (k+1) \\ & = & \frac{B}{C} \cdot H_{k+1} - (k+1) \end{array}$$

where H_n is the n-th harmonic number which satisfies $H_n = \Theta(\log n)$. Since $B = \Theta(k)$, the result follows. ■

IV. Buffer Management with Copying Cost ($\alpha > 0$)

In this section we consider the more involved case where each packet admitted to the buffer incurs *copying cost* α. In this model, it is preferable to perform as few preemptions as possible, since preemptions increase the costs, but do not contribute to the overall throughput. We recall that the performance of an algorithm in this model is defined as the algorithm's throughput, from which we subtract the overall copying cost incurred due to admitting distinct packets to the buffer.

A. Characterization of the Optimal Algorithm

We first note that if we consider algorithm PQ_1 described in the previous section, which is optimal for the case where $\alpha = 0$, we are guaranteed to have it produce the maximum throughput possible given the arrival sequence. If we further consider a slightly distorted model where PQ_1 is allowed to "pay" its copying cost only upon the successful delivery of a packet, we essentially obtain an optimal solution also for cases where $\alpha > 0$, because in that case PQ_1 never pays a useless cost of α for a packet that it ends up dropping . This is formalized in the following theorem:

Theorem 6. *PQ_1 that pays the copying cost only for transmitted packets is optimal for the (B, PQ, α, C)-architecture, for any $\alpha \in [0, 1)$.*

The theorem can also be seen with a different perspective. Intuitively, a PQ_1 scheduler would be optimal if it knew in advance which packets are winners and only accepted those packets. More formally, we can reach optimality by combining PQ_1 with a buffer admission control policy that would only accept the packets that ultimately depart using a given optimal scheduling policy.

B. Optimizing Priority Queuing

Given a copying cost $\alpha < 1$, we will define a value $\beta = \beta(\alpha) \geq 1$ (the precise value of β will be derived from our analysis below), which will be used in defining the preemption-rule DECIDEIFPREEMPT(PQ_β,p), as specified in Algorithm 3. The algorithm essentially preempts a packet q in favor of a newly arrived packet p only if p has significantly fewer residual passes than q's residual passes (where significantly means at most a fraction of $1/\beta$). Note that the special case where $\beta = 1$ coincided with algorithm PQ_1 described in section III-B (hence the subscript 1).

Algorithm 3 DECIDEIFPREEMPT(PQ_β,p)

```
1: $p_B \leftarrow$ last packet in $IB_t^{PQ_\beta}$
2:                         ▷ note that $r_t(p_B) = \max_{p' \in IB_t^{PQ_\beta}} r_t(p')$
3: if $r_t(p) < \frac{r_t(p_B)}{\beta}$ then
4:     drop $p_B$ and accept $p$
5: else
6:     reject $p$
7: end if
```

We now turn to analyze the performance of the algorithm which uses PQ_β-preemption. We first prove an upper bound on the performance of the algorithm, for any value of β. We can then optimize the value of $\beta = \beta(\alpha)$ so as to yield the best possible upper bound. We focus our attention in this analytical section on the case where $C = 1$. We note that for any two values of $\beta, \beta' \geq k$, the algorithm would behave the same, i.e., be non-preemptive. Furthermore, one should bear in mind that our algorithm does not require knowing the value of k in advance. Specifically, it might be that the value of β used by the algorithm is such that $\beta > k$ (being unaware of the real value of k). However, in our *analysis* we may assume that $\beta \leq k$, since any algorithm which uses a value of $\beta > k$ is equivalent to an algorithm that uses a value of k for β. This having been said, the optimal value of $\beta = \beta(\alpha, k)$, that minimizes the competitive ratio, does depend on the value of k, and in order to find this optimal value, k has to be given in advance.

Theorem 7. *For $C = 1$, algorithm PQ_β has a competitive ratio of*

$$\frac{(1 + \log_{\frac{\beta}{\beta-1}}(k/2) + 2\log_\beta k)(1-\alpha)}{1 - \alpha \log_\beta k},$$

for $B \geq 2$, $\beta > 1$, $\alpha < \min(1, 1/\log_\beta k)$.

The proof outline of the theorem is provided below. By optimizing the value of β one can obtain the minimum value for the competitive ratio, depending on the value of α. Figure 2 illustrates the performance guarantee as a function of β for various values of α, in the case where $k = 100$. Intuitively, as α gets larger, PQ_β pays more and more to accept packets that end up being preempted. Therefore, these useless costs increasingly weaken its performance guarantees when compared to the optimal offline algorithm. For example, the figure shows that for the case where $k = 100$ and $\alpha = 0.2$, the best competitive ratio for PQ_β is obtained by choosing $\beta^* \sim 5.43$, implying that PQ_{β^*} is 45.06-competitive.

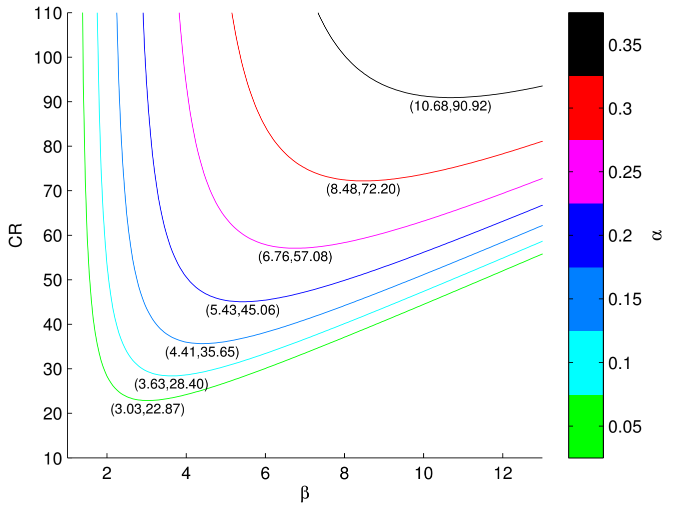


Fig. 2. The competitive ratio guarantee (y-axis) as a function of β (x-axis), for various values of α (color-coded). For each value of α considered, the plot specifies the optimal value of β, and the resulting competitive ratio guarantee. These guarantees are for $k = 100$.

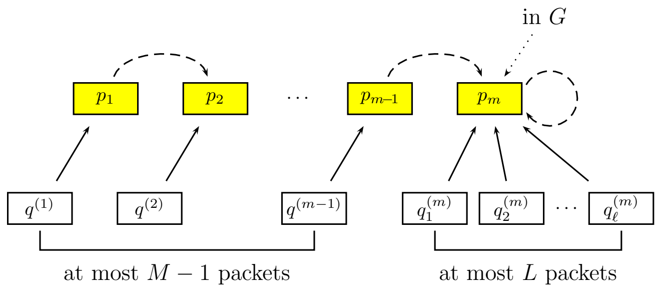


Fig. 3. Outline of mapping χ. Packet p_1 is admitted to the buffer upon arrival without preempting any packet, and henceforth packet p_{i+1} preempts packet p_i. The mapping ψ along the preemption sequence is depicted by dashed arrows. Such a sequence ends at a packet p_m which is successfully delivered by PQ_β. Mapping ϕ, depicted by solid arrows, maps at most 1 packet to any packet that is preempted in the sequence, and possibly additional ℓ packets to the last packet of the sequence which is successfully delivered by PQ_β. This gives an overall number of $2(m-1) + \ell \leq 2(M-1) + L$ packets mapped to any single packet successfully delivered by PQ_β.

C. Proof Intuition for Theorem 7

In the remainder of this section, we will focus on the proof of Theorem 7. We note that some of the proofs of the lemmas presented in this section appear in the appendix. We will denote by G the set of packets successfully delivered by PQ_β, and by O be the set of packets successfully delivered by some optimal solution OPT . Consider a partition of the set of packets $O \setminus G = A_1 \cup A_2$, such that A_1 is the set of packets dropped by PQ_β upon arrival, and A_2 is the remaining set of packets, consisting of packets that were originally accepted, but at some point were preempted by more favorable packets. It follows that $O = A_1 \cup A_2 \cup (G \cap O)$.

Our analysis will be based on describing a mapping of packets in O to packets in G, such that every packet in G piggybacks a bounded number of packets of O. Our mapping will be devised in several steps.

First, we define a mapping $\phi : A_1 \mapsto A_2 \cup G$ such that for every $p \in A_2$, $\left|\phi^{-1}(p)\right| \leq 1$, and for every $p \in G$, $\left|\phi^{-1}(p)\right| \leq$

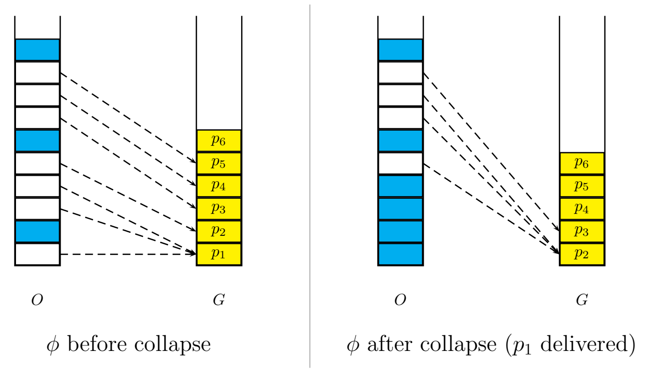


Fig. 5. Outline of a mapping-collapse. Upon the delivery of the HOL packet in G, p_1, the largest set of live A_1 packets closest to the head of queue in G (but no more than L) are mapped to the new HOL packet, p_2, and remaining packets of A_1 in O's buffer are shifted downwards appropriately. In this example we take $L = 3$.

L, for some value of L to be determined later (see Lemma 12). We then define a mapping $\psi : A_2 \cup G \mapsto G$ such that for every $p \in G$, $\left|\psi^{-1}(p)\right| \leq M$, for some value of M to be determined later (see Lemma 14). By composing these two mappings we obtain a mapping $\chi : O \setminus G \mapsto G$ such that for every $p \in G$, $\left|\chi^{-1}(p)\right| \leq 2(M-1)+L$, i.e., there are at most $2(M-1)+L$ packets from $O \setminus G$ mapped to any single packet in $p \in G$ by χ. Figure 3 gives an outline of the resulting mapping χ.

It is important to note that this mapping is done in hindsight, as part of the analysis, and is not part of the algorithm's definition. We can therefore assume that for our analysis, we know for every packet arrival which algorithm(s) would eventually successfully deliver this packet.

D. The Basic Mapping ϕ

Our goal in this section is to define a mapping $\phi : A_1 \mapsto A_2 \cup G$ such that for every $p \in A_2$, $\left|\phi^{-1}(p)\right| \leq 1$, and for every $p \in G$, $\left|\phi^{-1}(p)\right| \leq L$, for some value of L to be determined later (see Lemma 12). For every time t, we will denote the ordered set of packets residing in the buffer of PQ_β at t by p_1^t, p_2^t, and so on. Recall that since the buffer size is at most B, such a sequence is of length at most B. For clarity, in cases where the time step in question t is clear from the context, we will omit the subscript t, and refer to the packets in the buffer of PQ_β by $p_1, p_2, \ldots$. We will further define the *load of* p_i *at* t by $n_t(p_i) = \left|\phi^{-1}(p_i)\right|$, i.e. the number of packets currently mapped to packet p_i. In order to avoid ambiguity as for the reference time, t should be interpreted as the arrival time of a single packet. If more than one packet arrive in a time slot, these notations should be considered for every packet independently, in the sequence in which they arrive (although they might share the same actual time slot).

The mapping will be dynamically updated at each event of packet arrival, or packet transmission from the buffer of G, as follows: Assume packet p arrives at time t. We distinguish between 3 cases:

1) If p is not in both O and G (i.e., neither PQ_β nor OPT deliver it successfully), then the mapping remains unchanged.
2) If $p \in A_1$, and it is assigned to buffer slot j in the buffer of O upon arrival, perform an (O, j)*-mapping-shift* (see detailed description below).
3) If $p \in A_2 \cup G$, and it is assigned to buffer slot i in the buffer of G upon arrival (i.e., after its acceptance to the buffer we have $p_i = p$), perform a (G, i)*-mapping-shift* (see detailed description below).

The last case to consider is the case of a packet being successfully delivered by G. In this case we perform a *mapping-collapse* onto the head-of-line (HOL) packet in G (see detailed description below).

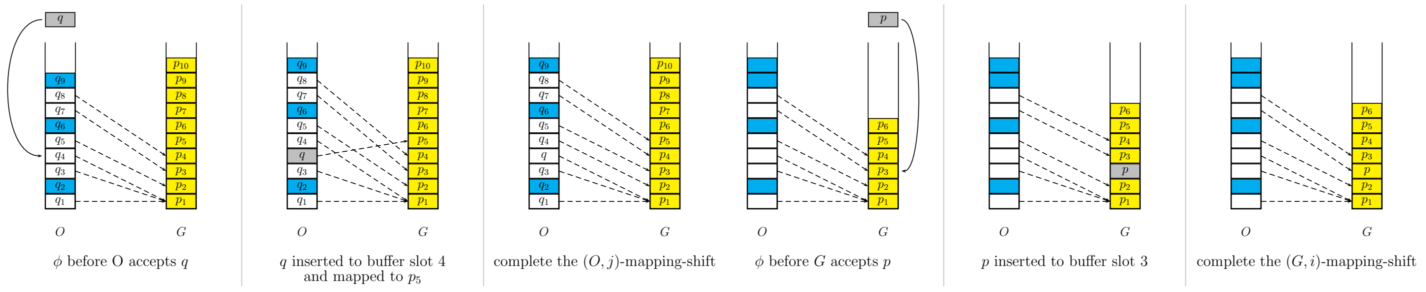
At any given time, we will consider the set of *live* packets in A_1 that are currently in the buffer of O, where this set is updated dynamically as follows: Every packet $q \in A_1$ is alive upon arrival. A live packet q ceases to be alive the moment $\phi(q)$ is completed (either by being preempted, or by being delivered). All remappings described henceforth only apply to live packets. Specifically, for every event causing a change or update in the mapping occurring in any time t, packets in A_1 which are no longer alive at t are essentially considered by the following procedures as packets which are in $A_2 \cup (G \cap O)$ (i.e., their mappings do not change, and they are sidestepped when shifting mappings).

We first give some definitions. We say a mapping is *sequential* if for every $i < i'$, the set of packets mapped to the packet in slot i would leave OPT before the set of packets mapped to the packet in slot i' (assuming both of these slots are non-empty). We further say a mapping is i*-prefix-full* if every packet in slot $i' \leq i$ has packets mapped to it and every packet in slot $i' > i$ has no packets mapped to it, and furthermore if $i > 1$ then the HOL packet in G has L packets mapped to it. See Figure 6 for an example of a mapping satisfying these two properties.

In order to finalize the description of ϕ, it remains to explain the notion of mapping-shifts, and mapping-collapse. An (O, j)*-mapping-shift* works as follows: If the HOL packet in G has less than L packets currently mapped to it, we map the arriving packet $p \in A_1$ to the HOL packet in G. Otherwise, we find the minimal index i of a packet in the buffer of G to which no packet is mapped to, and map packet p to this packet. If there is no such packet in the buffer of G (i.e., the HOL packet has load L, and every other packet in the buffer of G has load exactly 1), then we map p to the last packet in G. Clearly this mapping is feasible, i.e., whenever a packet $p \in A_1$ arrives, there is a packet in G to which we can map p. In order to complete this mapping-shift, we swap mappings (without changing the number of packets mapped to to any packet in G) such that the resulting mapping is sequential. See Figure 4(a) for an example of an (O, j)-mapping-shift.

A (G, i)*-mapping-shift* is simpler and works as follows: for any non-empty buffer-slot $j > i$, remap any packets mapped to p_j, to p_{j-1}, in sequence, starting from $j = i+1$. Figure 4(b) gives an example of a (G, i)-mapping-shift.

We now turn to describe the effect of a *mapping-collapse*, as illustrated in Figure 5. Upon the successful delivery of the HOL packet in G, the packet which was just in the second position in G's buffer, becomes the HOL. This packet may have at most 1 packet mapped to it (according to the definition


O
G
ϕ before O accepts q
q inserted to buffer slot 4 and mapped to p_5
complete the (O,j)-mapping-shift
ϕ before G accepts p
p inserted to buffer slot 3
complete the (G,i)-mapping-shift

(a) (O,j)-mapping-shift ($q \in A_1$ is admitted by O to slot $j = 5$)   (b) (G,i)-mapping-shift (p is admitted by G to slot $i = 3$)

Fig. 4. Outline of mapping-shifts. The new packet is inserted into the corresponding buffer slot, and the mapping is shifted accordingly. Cyan packets are packets that are either in $A_2 \cup (O \cap G)$, or packets in A_1 that are no longer alive. White packets are live A_1 packets, which might be affected by changes in the mappings. In both examples $L = 3$.

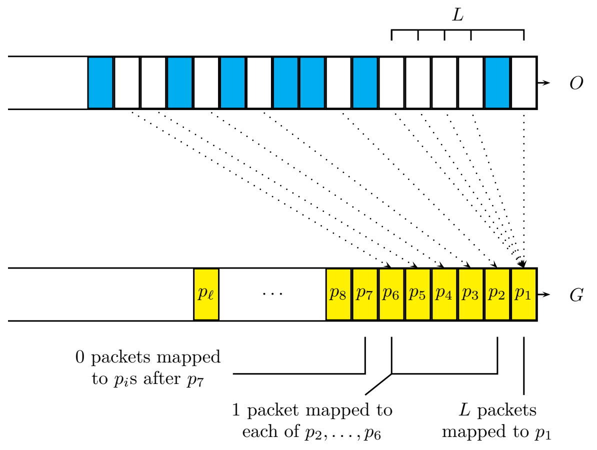

L
O
G
p_ℓ
p_8 p_7 p_6 p_5 p_4 p_3 p_2 p_1
0 packets mapped to p_is after p_7
1 packet mapped to each of $p_2, \ldots, p_6$
L packets mapped to p_1

Fig. 6. An example of a mapping which is 6-prefix-full and sequential. Cyan packets are packets that are either in $A_2 \cup (O \cap G)$ or A_1 packets that are no longer alive. Mapped white packets are in A_1. In the example above, $L = 5$ packets are mapped to p_1.

of the mapping). Upon its becoming the HOL packet, we remap the largest set of live packets in A_1 currently in the buffer of O, to the new HOL packet, such that there are at most L packets mapped to it. If we have remapped r such packets, and there remain additional packets in A_1 currently in the buffer of O, then we remap each of these packets r positions downward, such that the resulting mapping is i-prefix-full for some buffer position $i \in \{1, \ldots, B\}$.

We say a mapping satisfies the *head-of-line-before-OPT* (HOBO) property w.r.t. L, if at any time t, if the HOL packet in G has L packets mapped to it, then the last of these packets would leave O no earlier than this HOL packet would leave G.

The following lemma shows that if L satisifes the HOBO property, then except for maybe the HOL packet in the buffer of G, any other packet in the buffer has load of at most 1. This follows by definition for all such non-HOL packets, save possibly for the last packet in the buffer, which is the focus of the lemma.

Lemma 8. *If L satisfies the HOBO property, then at most one O packet is mapped to the last packet in G.*

The above lemma essentially guarantees that if we choose L such that the HOBO property is maintained, then each packet in G buffer except the first has at most one mapping.

The following lemma ensures that upon every event which affects the mapping, every live packet has sufficiently many residual passes, compared to the packet to which it is mapped to.

Lemma 9. *For every $i \in \{1, \ldots, B\}$, let p_i denote the packet residing in slot i in the buffer of G. For every such i, if q is (re)mapped to p_i at time t, then $r_t^O(q) \geq \frac{1}{\beta} r_t^G(p_i)$.*

In what follows, we assume that $B \geq 2$. For every packet $p \in G \cup A_2$, consider the set of packets mapped to p when p is completed. If $p \in A_2$, i.e., it is preempted by G at some time t, then since preemption always takes place from the last slot in the buffer of G, and since $B \geq 2$, by the definition of the mapping there could be at most one packet mapped to p when it is completed. We thus have the following lemma.

Lemma 10. *For every $p \in A_2$, $\phi^{-1}(p) \leq 1$ when p is preempted.*

If $p \in G$, let s_p be the latest time in which p becomes the HOL packet in G, and let s_p^* denote the time in which it is delivered by G. We further consider the set of packets mapped to p upon its delivery, and denote this set by $q_1, \ldots, q_\ell$, where the order is implied by the order in which these packets are mapped to p. This set of packets may be split into two sets: $q_1, \ldots, q_i$, which are the packets mapped to p due to the mapping-collapse at time s_p, and the set $q_{i+1}, \ldots, q_\ell$ which are additional packets mapped to p (which can only be due to O-shifts occurring at times $t \in H_p = [s_p, s_p^*)$). For every such packet q_i, let t_i denote the time in which it is mapped to p during the interval H_p. We hereby introduce the following notation: let $x_t^i = r_t^O(q_i)$, and let $y_t = r_t^G(p)$. Using this notation, Lemma 9 implies that at any time t_i, $x_t^i \geq \frac{1}{\beta} y_t$. Figure 7 provides a graphical description of the packets mapped to p along time.

In what follows we provide an analysis which will eventually enable us to determine the value of L used in the definition of the mapping. Consider any value $\tilde{L}$ which satisfies the following property (using the notation introduced above):

For every $p \in G$, if it has load $\tilde{L}$ upon delivery, there exists

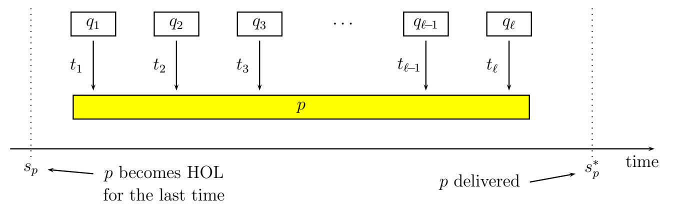


Fig. 7. An outline of the mappings to the HOL packet p in G during the interval $H_p = [s_p, s_p^*)$. For each i, packet q_i is mapped to p at time t_i.

some $t \in H_p$ such that

$$y_t \leq \sum_{1 \leq j \leq \tilde{L} | t_j \geq t} x_{t_j}^j. \tag{2}$$

Any value $\tilde{L}$ satisfying this property is said to be *HOBO-compliant*.

We first note that there exists an HOBO-compliant value of $\tilde{L}$. Assume we take in the our mapping $L = k$. This implies that during H_p, if p has load L, then there are $L = k$ distinct packets mapped to p during H_p, where each of these packets has some strictly positive number of residual passes. This implies that for $t = s_p$, Equation (2) holds, since $y_t \leq k \leq \sum_{j=1}^{k} x_{t_j}^j$. It follows that $\tilde{L} = k$ is HOBO-compliant. It is worthwhile to note that by definition, if $\tilde{L}$ is HOBO-compliant, then every $L \geq \tilde{L}$ is also HOBO-compliant (since the right hand side of Equation (2) can only increase, where the left hand side remains unchanged).

Next, we prove that for any value $\tilde{L}$ which is HOBO-compliant, satisfies HOBO.

Lemma 11. *If $\tilde{L}$ is HOBO-compliant, then $\tilde{L}$ satisfies HOBO.*

From now on, we assume without loss of generality that $\tilde{L}$ is the *minimal* integer that is HOBO-compliant. The following lemma provides an upper bound on such a minimal $\tilde{L}$.

Lemma 12. *$\tilde{L}$ satisfies $\tilde{L} \leq 2 + \log_{\frac{\beta}{\beta-1}} (k/2)$.*

We thus have the following corollary:

Corollary 13. *For $L = 2 + \log_{\frac{\beta}{\beta-1}} (k/2)$, for every $p \in G$, $\phi^{-1}(p) \leq L$ when p is delivered.*

E. The Mapping ψ

In this section we define a mapping $\psi : A_2 \cap G \mapsto G$ such that for every $p \in G$, $\left|\psi^{-1}(p)\right| \leq \log_\beta k$, i.e., there are at most $\log_\beta k$ packets from $A_2 \cap G$ mapped to any single packet in $p \in G$ by ψ.

The mapping essentially follows a preemption sequence of packets, up to a packet that is successfully delivered by G. Formally, it is defined by backward recursion as follows: if $p \in G$, then $\psi(p) = p$. Otherwise $p \in A_2$ is preempted in favor of some packet $q \in A_2 \cup G$, such that $r(p) > \beta r(q)$, in which case we define $\psi(p) = \psi(q)$. The following lemma follows immediately from the geometric decrease in the number of residual passes along a preemption sequence:

Lemma 14. *For every $p \in G$, $\left|\psi^{-1}(p)\right| \leq \log_\beta k$.*

F. Putting it All Together

We are now in a position to prove our main theorem.

Proof of Theorem 7: Our proof essentially relies on determining the value of L in the description of mapping ϕ. We set $L = 2 + \log_{\frac{\beta}{\beta-1}} (k/2)$, as suggested by Lemma 12 and Corollary 13.

By composing the mappings ϕ and ψ we obtain a mapping $\chi : A_1 \cap A_2 \mapsto G$ such that for every $p \in G$, $\left|\chi^{-1}(p)\right| \leq 2(\log_\beta k - 1) + L = L - 2 + 2\log_\beta k$. This follows from the fact that every packet along the preemption sequence except for the last piggybacks at most 1 packets by ϕ (Lemma 10), and the last packet in the preemption sequence piggybacks at most L packets by ϕ (Corollary 13). One should also take into account all the packets in the preemption sequence itself which are accounted for by ψ (save the last one, which is successfully delivered by G). Again, see Figure 3 for an illustration of the mapping χ.

All that remains is to bound the value obtained by the optimal solution, compared to the value obtained by by PQ_β. Assuming $\alpha < \frac{1}{\log_\beta k}$, one can see that the overall payments made by the algorithm in any preemption sequence sum to at most $\alpha \log_\beta k < 1$ (since payment is made only for packets in $A_2 \cup G$), and hence they do not exceed the unit profit obtained by delivering the last packet in the sequence. It follows that any packet delivered by our algorithm contributes at least a value of $1 - \alpha \log_\beta k$. For every such packet, the optimal solution may obtain a value of at most $(L - 2 + 2\log_\beta k + 1)(1 - \alpha) = (2 + \log_{\frac{\beta}{\beta-1}} (k/2) - 1 + 2\log_\beta k)(1 - \alpha)$. Note the additional value of 1 in the denominator, which accounts for packets in $O \cap G$. The result follows. ■

The following theorem introduces lower bound of PQ_β and its relationship to the values of k, α and β.

Theorem 15. *PQ_β is at least $max(\frac{(1-\alpha)(\beta - \log_\beta(k-1)+2)}{1-\alpha(\log_\beta(k-1)-1)}, \beta)$ competitive for $\beta > n - 2$ and $k = \beta^n + 1$, where $n > 1$ and $\alpha < \frac{1}{n-1}$.*

Proof: For simplicity we assume that B/C is an integer. Denote by OPT an algorithm that we compare with PQ_β. Next, we describe the arrival sequence that consists of periods. Each such period continues $x + 1$ time slots. During the first $x = \beta - n + 2$ time slots the next C packets are processed and transmitted by PQ_β. There are no arrivals during the first x time slots. Denote by F an available space in PQ_β buffer (in packets) at the beginning of any arrival phase.

For $\beta > n-2$, during $(x+1)$-th time slot arrive $nF + C(\beta - n + 2)$ packets that are ordered in the following way. The first nF packets are the following (ordered by residual passes): $\underbrace{\beta^n + 1}_{F}, \underbrace{\beta^{n-1}}_{F}, \underbrace{\beta^{n-2} - 1}_{F}, \ldots \underbrace{\beta - n + 2}_{F}$. The last $C(\beta - n + 2)$ packets have a single residual pass.

Since F is the amount of available space at the beginning of arrival phase, after acceptance of the first F packets PQ_β buffer is full. Thus, every next F packets preempt $n - 1$ times the previously accepted packets and after processing of the first nF packets each one of the newly accepted packets has $\beta - n + 2$ residual passes with $1 - \alpha(n - 1)$ value. Since the last $C(\beta - n + 2)$ packets have a single residual pass, then all of them are rejected by PQ_β but accepted by OPT. Therefore, during the period of time C packets are transmitted by PQ_β with overall $C(1-\alpha(n-1))$ value. From the other side OPT transmits $C(\beta-n+2)$ packets with the overall $C(1-\alpha)(\beta-n+2)$ value. Thus, PQ_β is $\frac{(1-\alpha)(\beta-n+2)}{1-\alpha(n-1)}$ competitive, where $\alpha(n-1) < 1$.

In the case when $\alpha \to 0$ the effect of preemptions on the gained value by PQ_β is reduced. In this case consider the following arrival sequence. During each arrival phase the input consists of B packets with β residual passes and C packets with a single residual pass and the order of the packets is non increasing by residual passes. Since PQ_β is greedy it fills up its buffer by the packets with β residual passes and does not accept any of the packets with a single residual pass. From the other side OPT accepts C packets with a single residual pass. So the competitive ratio of PQ_β is $\frac{(1-\alpha)\beta}{1-\alpha} = \beta$ if a number N of such arrival phases satisfies $N >> B$. ■

V. Simulation Study

In this section, we compare the performance of the family of algorithms PQ_β for various values of β (defined in Section IV), as well as algorithms PQ_1 and $FIFO_1$ (defined in Section III-B), and the non-preemptive algorithm that uses PQ (defined in Section III-A), which we dub PQ_∞ (this notation is used to maintain consistency with our notation of PQ_β).

When considering the family of algorithms PQ_β, we consider several values for β, and do not restrict ourselves to the optimal values implied by our analysis. The reason for this is that our analysis is targeted at bounding the worst case performance, and it is instructive to evaluate the performance of the algorithms using different values of β for simulated traffic that is not necessarily worst-case.

Our traffic is generated using an ON-OFF Markov modulated Poisson process (MMPP), which is targeted at producing bursty traffic. The choice of parameters is governed by the average arrival load, which is determined by the product of the average packet arrival rate and the average number of passes required by packets. For a choice of parameters yielding an average packet arrival rate of λ, where every packet has its required number of passes chosen uniformly at random within the range $[1, k]$, we obtain an average arrival load (in terms of required passes) of $\lambda \cdot \frac{k+1}{2}$.

Figures 8 and 9 provide the results of our simulations. The Y-axis in all figures represents the ratio between the algorithms' performance and the optimal performance possible given the arrival sequence. For the case where $\alpha = 0$ the optimal performance is obtained by PQ_1 (as proved in Theorem 4), whereas for $\alpha > 0$ the optimal performance is obtained by the algorithm that incurs the copying cost only upon transmission (as proved in Theorem 6).

We conduct two sets of simulations; one targeted at a better understanding of the dependence on the number of recycles, and the other targeted at evaluating the power of having multiple cores. We note that the standard deviation throughout our simulation study never exceeds 0.05 (deviation bars are omitted from the figures for readability).

A. *Variable Maximum Number of Required Passes*

In the first set of simulations we set the average arrival rate to be $\lambda = 0.3$. By performing simulations for variable values of the maximum number of required passes k in the range $[4, 24]$, we essentially evaluate the performance of our algorithms in settings ranging from underload (average arrival load of 0.75) to extreme overload (average arrival load of 3.75), which enables validating the performance of our algorithms in various traffic scenarios. For every choice of parameters, we conducted 20 rounds of simulation, where each round consisted of simulating the arrival of 1000 packets. Throughout our simulations we used a buffer of size $B = 20$, and restricted our attention to the single-core case, i.e., $C = 1$.

For $\alpha = 0$, Figure 8(a) shows that the performance of PQ_β degrades as β increases. This behavior is of course expected, since the optimal performance is known to be obtained by algorithm PQ_1 which preempts whenever some gain can be obtained. The non-preemptive algorithm (PQ_∞) has poor performance, and the performance of $FIFO_1$ lays in between the performance of the algorithms PQ_β and the non-preemptive algorithm. When further considering the performance of the algorithms for increasing values of α, in Figures 8(a)-8(c), and most notably in Figure 8(c), an interesting phenomenon is exhibited: the performance of all algorithms (especially $FIFO_1$) degrades substantially, save the performance of the non-preemptive algorithm which is maintained essentially unaltered. It should also be noted that all algorithms exhibit a performance far superior to the upper bound given in Theorem 7.

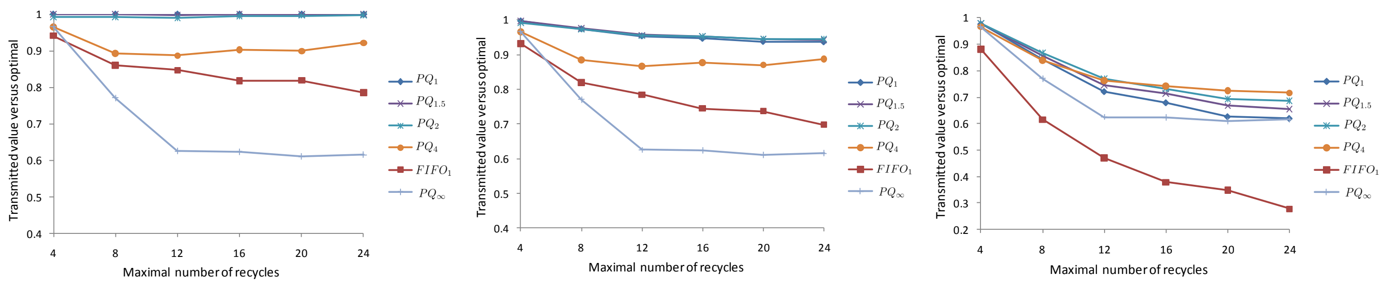
One of the most interesting aspects arising from our simulation results is the fact that they seem to imply that our *worst-case* analysis has been beneficial in designing algorithms that work well also *on average*. This can be seen especially by comparing Figures 8(b) and 8(c): the results show that when α changes, the value of β for which PQ_β performs best also changes (specifically, compare $PQ_{1.5}$ and PQ_2). This change is in accordance with the value of β that optimizes the competitive ratio, which is a *worst-case* bound derived from our analysis (see, e.g., optimal values of β in Figure 2).

B. *Variable Number of Cores*

In this set of simulations we evaluated the performance of our algorithms for variable values of C in the range $[1, 25]$. For each choice of parameters, we conducted 20 rounds of simulation, where each round consisted of simulating the arrival of 1000 packets. Throughout our simulations we used a buffer of size $B = 20$, and used $k = 16$ as the maximum number of passes required by any packet.

Figure 9(a) presents the results for a constant traffic arrival rate of $\lambda = 3$. Not surprisingly, the performance of all algorithms improves drastically as the number of cores increases. The increase in the number of cores essentially provides the network processor with a speedup proportional to the number of cores (assuming the average arrival rate remains constant).

We further evaluate the performance of our algorithms for increasing number of cores, while simultaneously increasing the average arrival rate (set to $\lambda = 0.3 \cdot C$, for each value of



(a) $\alpha = 0$ (b) $\alpha = 0.1$ (c) $\alpha = 0.4$

Fig. 8. Performance ratio of online algorithms versus an optimal offline algorithm for different values of α, as a function of the maximum number of passes k required by a packet k. The results presented are for a single core (i.e., $C = 1$). The average arrival rate of the simulated traffic for each value of k is fixed to 0.3 (packets per time slot).

C), such that the ratio between the speedup and the arrival rate remains constant. The results of this set of simulations is presented in Figures 9(b) and and 9(c), for $\alpha = 0$ and $\alpha = 0.4$, respectively. Contrarily to what may have been expected, the performance of some of the algorithms is not monotonically non-decreasing as the number of cores increases. Furthermore, the performance of some of the algorithms, and especially the non-preemptive algorithm PQ_∞, decreases drastically as the number of cores increases (up to a certain point), when compared to the optimal performance possible. Only once the number of cores is sufficiently large (which occurs when $C \geq 14$), do all algorithms exhibit a steady improvement in performance as the number of cores further increases. This is due to the fact that for such a large number of cores, almost all packets in the buffer are scheduled in every time slot (recall that the buffer used in our simulations has a size of $B = 20$). It is interesting to note that this behavior trend is independent of the value of α for both $FIFO_1$ and PQ_∞. These results provide further motivation, beyond the worst-case lower bounds presented in Section III-A, for adopting preemptive buffer management policies in multi-core, multipass NPs, and shows the vulnerability of architectures based on FIFO buffers.

VI. Discussion

The increasingly-heterogeneous packet-processing needs of NP traffic are posing design challenges to NP architects. In this paper we provide performance guarantees for various algorithms within the multipass NP architecture, and further validate these results by simulations.

Specifically, we consider the performance of FIFO and Priority-based scheduling disciplines. For non-preemptive buffering regimes we show that both scheduling disciplines may suffer significant performance degradation, whereas when preemptive policies are allowed we demonstrate that the priority-based approach can achieve optimal throughput, whereas FIFO may result in mild performance degradation. We then consider a more general model where preemptions incur a cost, in which we devise and analyze a priority-based algorithm whose performance is at most a logarithmic factor far from the optimal performance possible. We also conduct a comprehensive simulation study, which further validates our results.

Our results can be extended in several directions to reflect current NP constraints. Our work which focuses on unit-sized packets and homogeneous PPEs can be considered as a first step towards solutions which more generally deal with variable packet sizes and heterogeneous PPEs. In addition, it would be interesting to study non-greedy algorithms which are equipped with an admission control mechanism that aims at maximizing the guaranteed NP throughput. Last, it would be interesting to see the impact of moving the computation of the number of passes needed for each packet from the entrance of the NP to PPEs during the first pass. This is especially interesting because the first pass often corresponds to processing features that lead to the early dropping of packets, such as ACL processing.

VII. Acknowledgements

We thank the reviewers for their helpful comments that helped improved the paper.

References

[1] William Aiello, Rafail Ostrovsky, Eyal Kushilevitz, and Adi Rosen. Dynamic routing on networks with fixed-size buffers. In *SODA*, pages 771–780, 2003.

[2] Susanne Albers and Markus Schmidt. On the performance of greedy algorithms in packet buffering. *SIAM Journal on Computing*, 35(2):278–304, 2005.

[3] AMCC. np7310 10 gbps network processor, product brief, 2010. [Online] http://www.appliedmicro.com/MyAMCC/jsp/public/productDetail/product_detail.jsp?productID=nP7310.

[4] Yossi Azar and Arik Litichevskey. Maximizing throughput in multi-queue switches. *Algorithmica*, 45(1):69–90, 2006.

[5] Yossi Azar and Yossi Richter. Management of multi-queue switches in qos networks. *Algorithmica*, 43(1-2):81–96, 2005.

[6] M. Banikazemi, D. Poff, and B. Abali. Pam: a novel performance/power aware meta-scheduler for multi-core systems. In *SC*, pages 1–12, 2008.

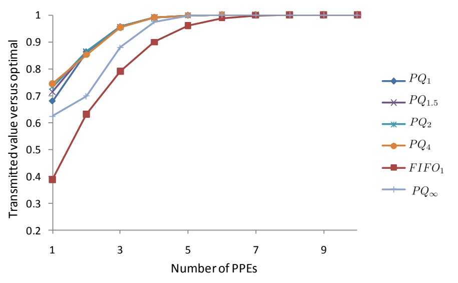
[7] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk Pruhs. On-line weighted flow time and deadline scheduling. *Journal of Discrete Algorithms*, 4(3):339–352, 2006.

[8] Michela Becchi, Mark A. Franklin, and Patrick Crowley. Performance/area efficiency in chip multiprocessors with micro-caches. In *Proceedings of the 4th ACM International Conference on Computing Frontiers (CF)*, pages 247–258, 2007.

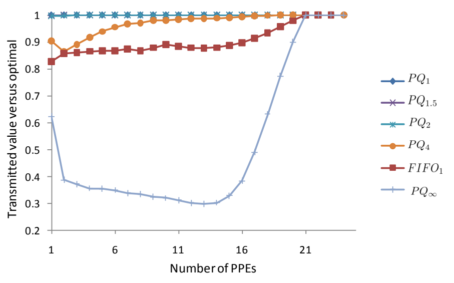
[9] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[10] Cavium. Octeon ii cn68xx multi-core mips64 processors, product brief, 2010. [Online] http://www.caviumnetworks.com/OCTEON-II_CN68XX.html.

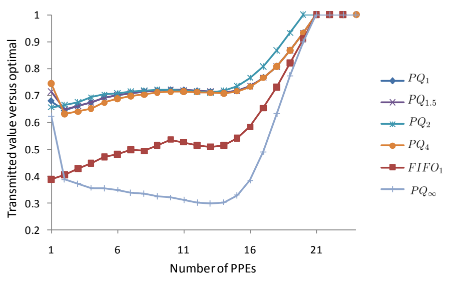
[11] Carri W. Chan and Nicholas Bambos. Throughput loss in task scheduling due to server state uncertainty. In *Proceedings of the 4th International ICST Conference On Performance Evaluation Methodologies And Tools (VALUETOOLS)*, 2009.



(a) Constant rate $\lambda = 0.3$ ($\alpha = 0.4, k = 16$)



(b) $\alpha = 0$ ($\lambda = 0.3 \cdot C, k = 16$)



(c) $\alpha = 0.4$ ($\lambda = 0.3 \cdot C, k = 16$)

Fig. 9. Performance ratio of online algorithms versus an optimal offline algorithm for different values of α, as a function of the number of cores C. In Figure 9(a), the arrival rate is kept constant at $\lambda = 0.3$, regardless of the number of cores C. In all other figures, the average arrival rate of the simulated traffic for each value of C is proportional to the number of cores (set to $\lambda = 0.3 \cdot C$).

[12] Cisco. The cisco quantumflow processor, product brief, 2010. [Online] http://www.cisco.com/en/US/prod/collateral/routers/ps9343/solution_overview_c22-448936.html.

[13] EZChip. Np-4 network processor, product brief, 2010. [Online] http://www.ezchip.com/p_np4.htm.

[14] Ellen L. Hahne, Alexander Kesselman, and Yishay Mansour. Competitive buffer management for shared-memory switches. In *SPAA*, pages 53–58, 2001.

[15] Juniper. Junos trio, white paper, 2009. [Online] http://www.juniper.net/us/en/local/pdf/whitepapers/2000331-en.pdf.

[16] Kimon Karras, Thomas Wild, and Andreas Herkersdorf. A folded pipeline network processor architecture for 100 gbit/s networks. In *ANCS*, 2010.

[17] Alexander Kesselman, Zvi Lotker, Yishai Mansour, and Boaz Patt-Shamir. Buffer overflows of merging streams. In *ESA*, pages 349–360, 2003.

[18] Alexander Kesselman, Zvi Lotker, Boaz Patt-Shamir, Yishai Mansour, Baruch Schieber, and Maxim Sviridenko. Buffer overflow management in qos switches. *SIAM Journal on Computing*, 33(3):563–583, 2004.

[19] Alexander Kesselman and Yishai Mansour. Harmonic buffer management policy for shared memory switches. *Theoretical Computer Science*, 324(2-3):161–182, 2004.

[20] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn. Using OS observations to improve performance in multicore systems. *IEEE Micro*, 28(3):54–66, 2008.

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

[22] Michael Meitinger, Rainer Ohlendorf, Thomas Wild, and Andreas Herkersdorf. A hardware packet re-sequencer unit for network processors. In *ARCS*, pages 85–97, 2008.

[23] Jayaram Mudigonda, Harrick M. Vin, and Raj Yavatkar. A case for data caching in network processors. Unpublished manuscript.

[24] 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.

[25] Netronome. Nfp-32xx network flow processor, product brief, 2010. [Online] http://www.netronome.com/pages/network-flow-processors.

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

[27] Joy Kuri S. Govind, R. Govindarajan. Packet reordering in network processors. In *IPDPS*, 2007.

[28] Timothy Sherwood, George Varghese, and Brad Calder. A pipelined memory architecture for high throughput network processors. In *Proceedings of the 30th International Symposium on Computer Architecture (ISCA)*, pages 288–299, 2003.

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

[30] K. Tian, Y. Jiang, and X. Shen. A study on optimally co-scheduling jobs of different lengths on chip multiprocessors. In *CF*, pages 41–50, 2009.

[31] Charlie Wiseman, John DeHart, Mart Haitjema, Shakir James, Fred Kuhns, Jing Lu, Jyoti Parwatikar, Ritun Patney, Michael Wilson, Ken Wong, and David Zar. Remotely accessible network processor-based router for network experimentation. In *ANCS*, pages 20–29, 2008.

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

[33] Xelerated. X11 family of network processors, product brief, 2010. [Online] http://www.xelerated.com/Uploads/Files/67.pdf.

[34] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *ASPLOS*, pages 129–142, 2010.

Appendix

Proof of Lemma 8: Assume $q \in A_1$ arrives at time t. It follows that upon its arrival, the buffer of G is full (since otherwise it would have accepted q). Assume by contradiction that the HOL packet in G's buffer has load L, and that every other packet in G's buffer has load exactly 1. Since L satisfied the HOBO property, we are guaranteed to have that the last packet mapped to the HOL packet in G is delivered by O no earlier than the HOL packet in G is delivered by G. In particular, at time t the last packet mapped to the HOL packet in G has not yet been delivered by O, and it resides in the buffer of O. Since the mapping is maintained sequential, we are guaranteed to have that all packets in the buffer of O mapped to packets other than the HOL packet in G are also residing in the buffer of O. It follows that the buffer occupancy of O is at least that of G, which by the fact that the buffer of G is full, implies that the buffer of O is full upon the arrival of q, contradicting the fact that O accepts q upon its arrival (since recall we assumed without loss of generality that O never preempts an accepted packet). ■

Proof of Lemma 9: We prove by induction on the sequence of events (essentially, induction on t) that in every event which causes a (re)mapping of q to some packet p_i at time t, the property $r_t^O(q) \geq \frac{1}{\beta} r_t^G(p_i)$ holds. First note that every packet $q \in A_1$ arriving at time t', causes an O-shift, which implies that at time t', $r_t^O(q) \geq \frac{1}{\beta} r_t^G(p_j)$ for all $j \in \{1, \ldots, B\}$, and in particular this is true for the packet p_i to which q is mapped to upon its arrival at time t'. We now turn to the induction step, and prove the property holds for every event causing a remapping of q. For an O-shift affecting the mapping of q, this happens upon the arrival of a packet q' at some time t, such that $r_t^O(q') \leq r_t^O(q)$ (since by definition, an (O, j)-shift might cause remapping of packets only in positions $j' \geq j$. By combining this observation with the fact that an O-shift occurred upon the arrival of q' implies $r_t^O(q') \geq \frac{1}{\beta} r_t^G(p_j)$ for all $j \in \{1, \ldots, B\}$, we are guaranteed to the have the property hold in case of an O-shift. For a G-shift affecting the mapping of q, note that this can only occur by changing the target of the mapping from being packet p_{i+1}, to being packet p_i (by the definition of G-shifts). In this case we have $r_t^O(q) \geq \frac{1}{\beta} r_t^G(p_{i+1}) \geq \frac{1}{\beta} r_t^G(p_i)$, since G maintains its buffer in non decreasing order of residual passes. The last case to consider is the that where we have a mapping-collapse affecting the mapping of q. In this case, q is originally mapped to some packet p_{i+m} (for some m), and after the collapse is mapped to packet p_i. In this case as well, similarly to the case of the G shift, since we remap to packets which are closer to G's head-of-queue, the induction hypothesis implies the property is maintained. ■

Proof of Lemma 11: Consider the time $t \in H_p$ for which Equation (2) holds. For such a t, the sum of residual passes of packets mapped to p as of time t (non of which is already delivered by O) is at least the number of residual passes remaining for p at time t. Since by definition p is the HOL packet throughout H_p, and is delivered by the end of this interval, this implies that the last of the packets mapped to p cannot be delivered by O before p is delivered by G. ■

Proof of Lemma 12: By definition, $\tilde{L}$ is the minimal value for which for any packet $p \in G$ such that has load L upon its delivery, there exists some time $t \in [s_p, s_p^*]$ for which $y_t \leq \sum_{j \leq \tilde{L} | t_j \geq t} x_{t_j}^j$.

Therefore, if we consider $\tilde{L} - 1$, then there exists some packet $p \in G$ that has load $\tilde{L} - 1$, yet for every time $t \in [s_p, s_p^*]$, $y_t > \sum_{j \leq \tilde{L} | t_j \geq t} x_{t_j}^j$. In particular this holds for every time $t = t_i$, for $i = 1, \ldots, \tilde{L} - 1$. We now abuse notation and let $y_i = y_{t_i}$, and also let $x_i = x_{t_i}^i$. We further let $z_i = \sum_{j=i}^{L-1} x_j$. Using the above notation, we have for every $i \in \{1, \ldots, L-1\}$, $y_i > z_i$, i.e. $y_i \geq z_i + 1$.

In addition, by Lemma 9 we have for every $i = 1, \ldots, L-2$, $x_i = z_i - z_{i+1} \geq \frac{1}{\beta} \cdot y_i$. Therefore, combining both equations, we obtain: $z_{i+1} \leq z_i - \frac{1}{\beta} \cdot y_i \leq z_i - \frac{1}{\beta} \cdot (z_i + 1)$, which translates to $z_{i+1} + 1 \leq \left(1 - \frac{1}{\beta}\right) \cdot (z_i + 1)$.

By iteration, we get: $z_{L-1} + 1 \leq \left(1 - \frac{1}{\beta}\right)^{L-2} \cdot (z_1 + 1)$. Finally, we use $z_{L-1} + 1 = x_{L-1} + 1 \geq 2$ and $z_1 + 1 \leq y_1 \leq k$. Therefore we obtain $2 \leq k \cdot \left(1 - \frac{1}{\beta}\right)^{L-2}$. Rearranging the terms we obtain the required result. ■

Proof of Corollary 13: By our choice of L, Lemma 12 implies that L is HOBO-compliant. For such an L, Lemmas 11 and 8 ensure that the mapping ϕ is feasible. Since every $p \in G$ is the HOL packet of G upon delivery, it follows by the definition of the mapping that $\phi^{-1}(p) \leq L$, as required. ■

PLACE PHOTO HERE

Isaac Keslassy (M'02, SM'11) received his M.S. and Ph.D. degrees in Electrical Engineering from Stanford University, Stanford, CA, in 2000 and 2004, respectively.

He is currently an associate professor in the Electrical Engineering department of the Technion, Haifa, Israel. His recent research interests include the design and analysis of high-performance routers and on-chip networks. The recipient of the European Research Council Starting Grant, the Yigal Alon Fellowship and the ATS-WD Career Development Chair, he is an associate editor for the IEEE/ACM Transactions on Networking.

PLACE PHOTO HERE

Kirill Kogan finished the B.Sc., M.Sc in Computer Science and Ph.D degrees in Communication Systems Engineering from Ben-Gurion University of the Negev, Israel. He was a Technical Leader at Cisco Systems for more than 10 years. Currently, he is a postdoctoral fellow at the Computer Science department, Waterloo University, Canada. His research focuses on development of efficient algorithms for routers and switches in various aspects of buffer management, quality of service, scheduling, access to resources and packet classification.

PLACE PHOTO HERE

Gabriel Scalosub received his B.Sc. in Mathematics and Philosophy from the Hebrew University of Jerusalem, Israel, in 1996. He received his M.Sc. and Ph.D. in Computer Science from the Technion - Israel Institute of Technology, Haifa, Israel, in 2002 and 2007, respectively. In 2008 he was a postdoctoral fellow at Tel-Aviv University, Israel, and in 2009 he was a postdoctoral fellow at the University of Toronto, Canada. In October 2009 Gabriel Scalosub joined the Department of Communication Systems Engineering at Ben Gurion University of the Negev, Israel. He served on various technical program committees, including Infocom, IWQoS, IFIP-Networking, ICCCN, and WCNC. His research focuses on theoretical algorithmic issues arising in various networking environments, including buffer management, scheduling, and wireless networks. He is also interested in broader aspects of combinatorial optimization, online algorithms, approximation algorithms, and algorithmic game theory.

PLACE PHOTO HERE

Michael Segal (M'04, SM'08) finished B.Sc., M.Sc. and Ph.D. degrees in Computer Science from Ben-Gurion University of the Negev in 1994, 1997, and 1999, respectively. During a period of 1999-2000 Prof. Michael Segal held a MITACS National Centre of Excellence Postdoctoral Fellow position in University of British Columbia, Canada. Prof. Segal joined the Department of Communication Systems Engineering, Ben-Gurion University, Israel in 2000 where he served as department's Chairman between 2005-2010. His primary research includes algorithms (sequential and distributed), data structures with applications to optimization problems, mobile wireless networks, communications and security.