

Providing Performance Guarantees in Multipass Network Processors

Isaac Keslassy
Department of Electrical Engineering
Technion
Haifa 32000, Israel
isaac@ee.technion.ac.il

Kirill Kogan[†]
Cisco Systems
South Netanya 42504, Israel
kkogan@cisco.com

Gabriel Scalosub Michael Segal
Department of Communication Systems Engineering
Ben-Gurion University of the Negev
Beer-Sheva 84105, Israel
{sgabriel, segal}@bgu.ac.il

Abstract—Current network processors (NPs) increasingly deal with packets with heterogeneous processing times. As a consequence, packets that require many processing cycles can significantly 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 that 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, DPI and intrusion detection. They are often implemented using many processing cores. These cores are either arranged as a pool of identical cores (e.g., the Cavium CN68XX [1]), as a long pipeline of cores (e.g., the Xelerated X11 [2]) or as a combination of both (e.g., the EZChip NP-4 [3]).

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 VPN encryptions, hierarchical packet classification, and compression/decompression.

These features are increasingly challenging for traditional architectures, raising 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 [2]) or longer processing times in run-for-completion cores. Second, a few packets with

many features can delay, and even temporarily starve or cause to drop, the later packets.

In view of the increasing impact of 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.

A major benefit of the multipass architecture is that it is more flexible than the run-for-completion approach. Also, it does not require the NP designer to define a large pipeline length in advance. This is especially useful for NPs with different possible markets. In addition, 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 implemented in the recent Cisco QuantumFlow NP [4]. Also, although not strictly multipass NP architectures, several NP architectures in the literature already allow for recycling of complex packets, such as IP control packets [5].

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 [6]). This is the main motivation guiding our algorithmic design, and methods of analysis.

B. Our Contributions

The goal of this paper is to offer designs with proven performance guarantees for the multipass-NP scheduler. In this paper, we analyze the performance of scheduling and buffer

[†]This work was done while the author was with the Department of Communication Systems Engineering at the Ben-Gurion University of the Negev.

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.

We consider settings where each arriving packet requires some number of processing passes, and study the interplay of three factors: the scheduling policy, the buffer management policy, and the copying cost of packets into the buffer. 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 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 Priority-based environments (Section III). For the case where the system incurs a strictly positive copying cost, we devise competitive buffer management algorithms for Priority-based environments, and provide an elaborate analysis of their performance guarantees (Section IV). Due to space constraints, some of the proofs are omitted, and can be found in [7].

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 (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 [5]. However, to our knowledge, there is no previous work in the literature that discusses the scheduling and buffer management policies in multipass NPs. Moreover, no paper analyzes the impact of the packet admission control policy on the worst-case NP performance.

There is a long history of OS scheduling for multithreaded processors. A comprehensive overview of competitive online scheduling for server systems is provided in [8]. 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 to be optimal for mean response [9]. 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 due to its inherently-limited buffer size.

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-

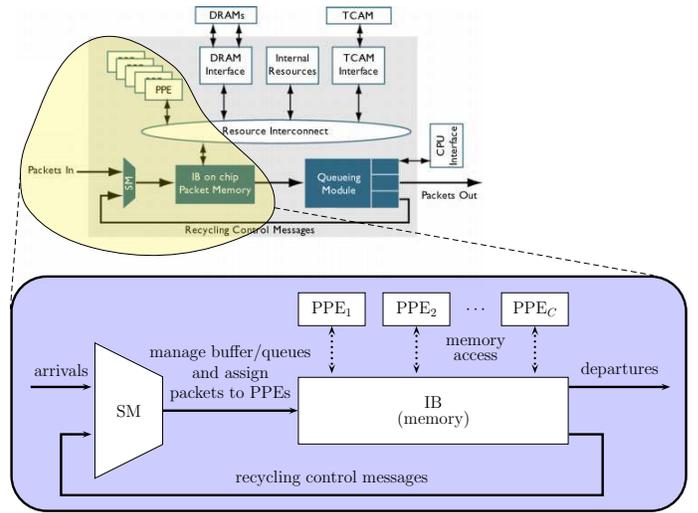


Fig. 1. An outline of the architecture model, as an abstraction of a standard Multipass NP Architecture (see, e.g. [4]).

Queued (OQ) switches (e.g., [10], [11]) and shared memory switches with OQs (e.g., [12], [13]).

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 [4]. 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, while packets with remaining passes recycle control messages, (ii) *arrival*, in which the SM performs its buffer management task considering newly arrived packets and recycled control messages (observe that recycled control messages 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.

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 [14], 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. Finally, we assume that all packets in a given sub-flow require the same number of passes, and therefore there are no reordering issues.

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. Therefore, they 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 messages 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* of $\alpha > 0$ for copying it to *IB*. Clearly, since packets have unit value, we need only consider costs $\alpha \in [0, 1]$.

Finally, we measure the final *overall value* as the total throughput value minus the total copying cost. Therefore, for the case where $\alpha = 0$, the overall value is equal to the system throughput. For the case where $\alpha > 0$, the overall value equals the throughput minus the overall copying cost incurred by admitting packets to *IB*.

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

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

Our goal is to provide performance guarantees for various scheduling and buffer management algorithms. We use competitive analysis [15], [16] 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.

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

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

```

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 time-slot t , we let IB_t^{ALG} denote the set of packets stored in *ALG*'s *IB* 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 need 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 $r_t(p)$, is defined as 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 DECIDEIFPREEMPT subroutine determining whether or not preemption takes place will depend on the algorithm. We will focus on two natural BMs:

- 1) *FIFO*: where 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)*: where packets with less residual passes have higher priority and are served first, 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.

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* (recall that omitted proofs can be found in [7]).

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

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

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

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

B. Preemptive Policies

For the case where $\alpha = 0$, we consider the intuitive preemption rule in which a newly arrived packet p should preempt a buffered packet q at time t iff it has a lower number of residual passes, i.e. $r_t(p) < r_t(q)$. This rule is formalized in Algorithm 2, which gives a formal definition of the DECIDEIFPREEMPT procedure of Algorithm 1. 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:* Consider Algorithm 2 with a BM implementing PQ. We refer to this algorithm as PQ_1 .¹ The following theorem characterizes its performance.

Theorem 3. *PQ_1 is optimal in a $(B, PQ, C, 0)$ -system.*

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

However, 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 higher-priority packets. Therefore, it is of interest to study BMs that ensure such scenarios do not occur. One such predominant BM uses FIFO queueing, which is discussed in the following section.

2) *Preemptive FIFO:* Consider Algorithm 2 with a BM implementing 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 by providing the counterpart to Theorem 3. It shows that the performance of $FIFO_1$ can be rather far from optimal, as opposed to priority queueing.

Theorem 4. *$FIFO_1$ has competitive ratio $\Omega(\frac{\log k}{C})$ in a $(B, FIFO, C, 0)$ -system.*

We now turn to providing an upper bound on the performance of $FIFO_1$, as given by the following theorem.

Theorem 5. *$FIFO_1$ is $2k$ -competitive in a $(B, FIFO, C, 0)$ -system.*

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:                                      $\triangleright$  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

```

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 a *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. Recall that the overall performance of an algorithm in this model is defined as the algorithm throughput, from which we subtract the overall copying cost incurred by 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 any (B, PQ, C, α) -architecture.*

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 optimum scheduling policy.

B. Optimizing Priority Queueing

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 β fewer residual passes than q 's residual passes. Note that the special case of PQ_β with $\beta = 1$ coincides with algorithm PQ_1 described in section III-B (hence the subscript 1).

We now turn to analyzing the performance of the algorithm with PQ_β -preemption. We first prove an upper bound on the performance of the algorithm, for any value of β . We can then

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

Algorithm 3 DECIDEIFPREEMPT(PQ_β, p)

- 1: $p_B \leftarrow$ last packet in $IB_t^{PQ_\beta}$
 - 2: \triangleright 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**
-

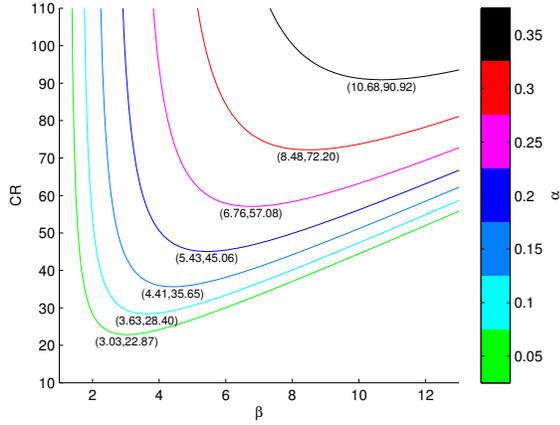


Fig. 2. The competitive ratio guarantee (y -axis) as a function of β (x -axis), for various values of α (color-coded). For each value of α considered, the plot specifies the optimal value of β , and the resulting competitive ratio guarantee. These guarantees are for $k = 100$.

optimize the value of $\beta = \beta(\alpha)$ so as to yield the best possible upper bound.

Theorem 7. PQ_β has a competitive ratio of

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

in a $(B, PQ, 1, \alpha)$ -system, 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.

C. Proof Intuition for Theorem 7

In the remainder of this section, we focus on the proof of Theorem 7. Being rather technical, we provide here only a high-level description of the proof, whereas the full proof can be found in [7]. 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

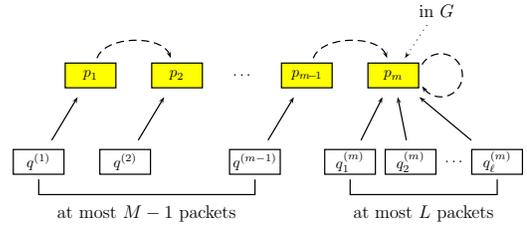


Fig. 3. Mapping χ outline: packet p_1 is admitted to the buffer upon arrival without preempting any packet, and henceforth packet p_{i+1} preempts packet p_i . The mapping ψ along the preemption sequence is depicted by dashed arrows. Such a sequence ends at a packet p_m which is successfully delivered 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 an 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 .

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 $p \in A_2$, $|\phi^{-1}(p)| \leq 1$, and for every $p \in G$, $|\phi^{-1}(p)| \leq L$, for some value of L to be determined later (Lemmas 10 and 11). We then define a mapping $\psi : A_2 \cup G \mapsto G$ such that for every $p \in G$, $|\psi^{-1}(p)| \leq M$, for some value of M to be determined later (Lemma 12). By composing these two mappings we obtain a mapping $\chi : O \setminus G \mapsto G$ such that for every $p \in G$, $|\chi^{-1}(p)| \leq 2(M-1) + L$, 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 $p \in A_2$, $|\phi^{-1}(p)| \leq 1$, and for every $p \in G$, $|\phi^{-1}(p)| \leq L$, for some value of L to be determined later. 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, we will sometimes abuse notation and omit the superscript t , when it is clear from the context. We will further define the *load* of p_i at t by $n_t(p_i) = |\phi^{-1}(p_i)|$, 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. When more than one packet arrive in a time slot, these notations should be considered for every packet

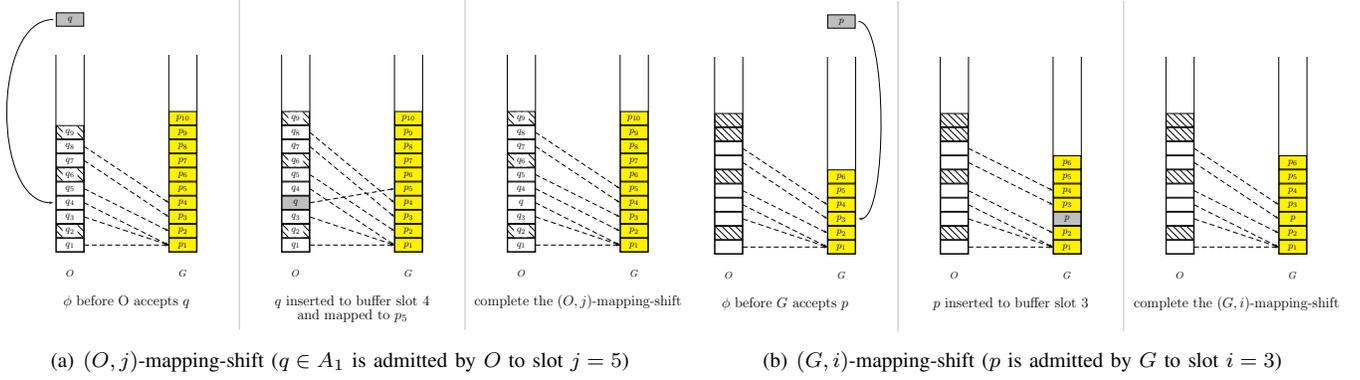


Fig. 4. *Mapping-shifts* outline: the new packet (marked gray) is inserted into the corresponding buffer slot, and the mapping is shifted accordingly. Packets marked with diagonal lines are either packets in $A_2 \cup (O \cap G)$, or packets in A_1 that are no longer alive. White packets are live A_1 packets, which might be affected by changes in the mappings. In both examples $L = 3$.

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 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.

In order to finalize the description of ϕ , it remains to explain the notion of mapping-shifts, and mapping-collapse. As illustrated in Figure 4(a), 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 any packet in G) such that the resulting mapping is sequential.

As shown in Figure 4(b), 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 $j = i + 1$.

We now turn to describing the effect of a *mapping-collapse*, as illustrated in Figure 5. Upon the successful delivery of the HoL packet in G , the packet that was just in the second position in G 's buffer, becomes the HoL. 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 (our analysis will later show that this is indeed feasible). 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 $i \in \{1, \dots, B\}$.

We say that a mapping satisfies the *HoL-before-OPT* (HOBO) property w.r.t. L , iff at any time t , whenever 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 satisfies the HOBO

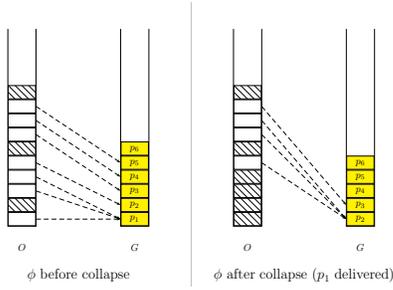


Fig. 5. *Mapping-collapse* outline: upon the delivery of the HoL packet in G , p_1 , the largest set of live A_1 packets closest to the head of queue in G (but no more than L) are mapped to the new HoL packet, p_2 , and remaining packets of A_1 in O 's buffer are shifted downwards appropriately. In this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

property, then except for the HoL packet in the buffer of G , any other packet in the buffer necessarily has load 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 's buffer, except for the first packet, has at most one mapping. The following lemma ensures that upon any event that 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 $i \in \{1, \dots, B\}$, 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 $r_t^O(q) \geq \frac{1}{\beta} r_t^G(p_i)$.*

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.*

In order to complete our analysis of ϕ , it suffices to find a value L that satisfies the HOBO property. Combined with Lemma 8, this would provide a bound on the number of packets mapped to packets in G .

Lemma 11. *Choosing $L = 2 + \log_{\frac{\beta}{\beta-1}}(k/2)$ satisfies the HOBO property. Hence, for this value of L , every $p \in G$ satisfies $\phi^{-1}(p) \leq L$ 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 $p \in G$, $|\psi^{-1}(p)| \leq \log_{\beta} k$, 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 $\psi(p) = \psi(q)$. Since each such preemption witnesses a reduction in the number of residual recycles by a factor of β , we obtain a logarithmic bound on the length of such a preemption sequence, which implies the following:

Lemma 12. *For every $p \in G$, $|\psi^{-1}(p)| \leq \log_{\beta} k$.*

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 11.

By composing the mappings ϕ and ψ we obtain a mapping $\chi : A_1 \cup A_2 \mapsto G$ such that for every $p \in G$, $|\chi^{-1}(p)| \leq 2(\log_{\beta} k - 1) + L = L - 2 + 2\log_{\beta} k$. This follows from the fact that every packet along the preemption sequence, except for the last one, piggybacks at most 1 packet by ϕ (Lemma 10), and the last packet in the preemption sequence piggybacks at most L packets by ϕ (Lemma 11). 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 χ .

All that remains is to bound the value obtained by the optimal solution, compared to the value obtained 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 which accounts for packets in $O \cap G$). ■

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

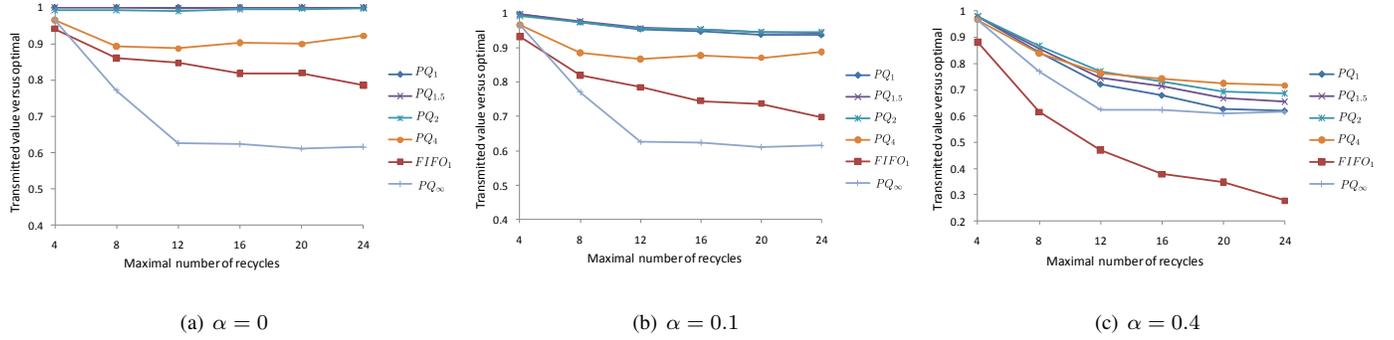


Fig. 6. Performance ratio of online algorithms versus optimal for different values of α , as a function of the maximum number of passes k required by a packet k . The results presented are for a single core (i.e., $C = 1$). The average arrival rate of the simulated traffic for each value of k is fixed to 0.3 (packets per time slot).

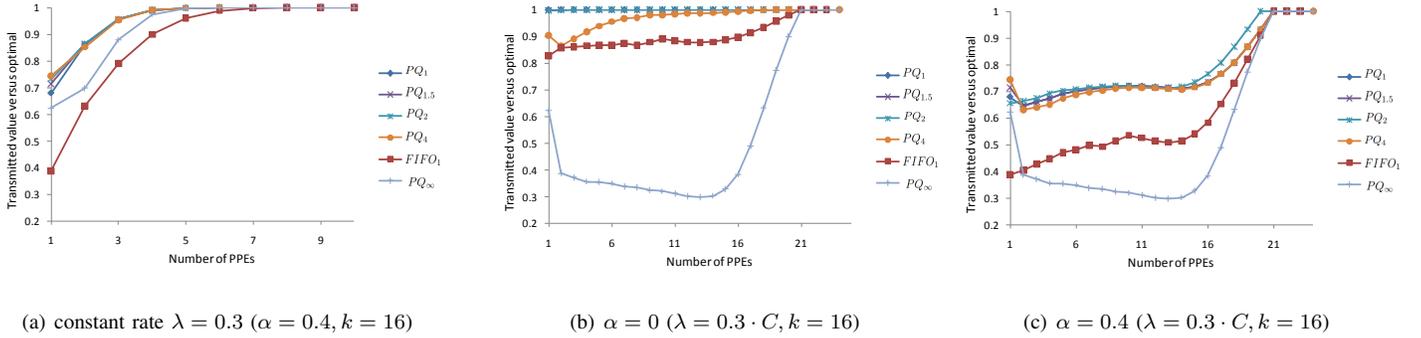


Fig. 7. Performance ratio of online algorithms versus optimal for different values of α , as a function of the number of cores C . In Figure 7(a) the arrival rate is kept constant at $\lambda = 0.3$, regardless of the number of cores C . In all other figures the average arrival rate of the simulated traffic for each value of C is proportional to the number of cores (set to $\lambda = 0.3 \cdot C$).

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 6 and 7 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 3), 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 conduct 20 rounds of simulation, where each round consists of simulating the arrival of 1000 packets. Throughout our simulations we use a buffer of size $B = 20$, and restrict our attention to the single-core case, i.e., $C = 1$.

For $\alpha = 0$, Figure 6(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 6(b)-6(c), and most notably in Figure 6(c), it is interesting to note that the performance of all algorithms (especially $FIFO_1$) degrades substantially, except for 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 to designing algorithms that work well also *on average*. This can be seen especially by comparing Figures 6(b) and 6(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

B. Variable Number of Cores

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

Figure 7(a) presents the results for a constant traffic arrival rate of $\lambda = 0.3$. 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 numbers of cores, while simultaneously increasing the average arrival rate (set to $\lambda = 0.3 \cdot C$, for each value of C), such that the ratio between the speedup and the arrival rate remains constant. The results of this set of simulations are presented in Figures 7(b) and 7(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.

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 admission control mechanisms that aim 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

The authors would like to thank Isask'har (Zigi) Walter for his comments.

This research work was partly supported at the Ben-Gurion University by the US Air Force European Office of Aerospace Research and Development grant FA8655-09-1-3016, by Deutsche Telecom, by the EC FLAVIA project, and by the Israeli MOITAL CORNET consortium; and at the Technion by the European Research Council Starting Grant n° 210389, by an Intel research grant, by the Hasso Platner Center for Scalable Computing, and by the Israeli MOST CMP Research Center.

REFERENCES

- [1] Cavium, OCTEON II CN68XX Multi-Core MIPS64 Processors, Product Brief, 2010. [Online] http://www.caviumnetworks.com/OCTEON-II_CN68XX.html.
- [2] Xelerated, X11 Family of Network Processors, Product Brief, 2010. [Online] <http://www.xelerated.com/Uploads/Files/67.pdf>.
- [3] EZChip, nP-4 Network Processor, Product Brief, 2010. [Online] http://www.ezchip.com/p_np4.htm.
- [4] Cisco, the Cisco QuantumFlow Processor, Product Brief, 2010. [Online] http://www.cisco.com/en/US/prod/collateral/routers/ps9343/solution_overview_c22-448936.html.
- [5] C. Wiseman et al., "Remotely accessible network processor-based router for network experimentation," in *ANCS*, 2008, pp. 20–29.
- [6] T. Sherwood, G. Varghese, and B. Calder, "A pipelined memory architecture for high throughput network processors," in *ISCA*, 2003, pp. 288–299.
- [7] I. Keslassy, K. Kogan, G. Scalosub, and M. Segal, "Providing performance guarantees in multipass network processors," Technical Report TR10-02, Comnet, Technion, Israel, 2010. [Online] http://www.ee.technion.ac.il/~isaac/p/tr10-02_multipass.pdf.
- [8] K. Pruhs, "Competitive online scheduling for server systems," *PER*, vol. 34, no. 3, pp. 52–58, 2007.
- [9] L. Schrage, "A proof of the optimality of the shortest remaining processing time discipline," *OR*, vol. 16, no. 3, pp. 687–690, 1968.
- [10] A. Kesselman, Z. Lotker, B. Patt-Shamir, Y. Mansour, B. Schieber, and M. Sviridenko, "Buffer overflow management in QoS switches," *SICOMP*, vol. 33, no. 3, pp. 563–583, 2004.
- [11] G. Scalosub, J. Liebeherr, and P. Marbach, "Buffer management for aggregated streaming data with packet dependencies," in *Infocom*, 2010.
- [12] W. Aiello, A. Kesselman, and Y. Mansour, "Competitive buffer management for shared-memory switches," *TALG*, vol. 5, no. 1, 2008.
- [13] A. Kesselman and Y. Mansour, "Harmonic buffer management policy for shared memory switches," *TCS*, vol. 324, no. 2–3, pp. 161–182, 2004.
- [14] T. Wolf, P. Pappu, and M. A. Franklin, "Predictive scheduling of network processors," *Computer Networks*, vol. 41, no. 5, pp. 601–621, 2003.
- [15] D. Sleator and R. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, vol. 28, no. 2, pp. 202–208, 1985.
- [16] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.