Providing Performance Guarantees in Multipass Network Processors

Isaac Keslassy
Department of Electrical engineering
Technion – Israel Institute of Technology
Haifa 32000, Israel
isaac@ee.techion.ac.il

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

***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, DPI, intrusion detection, SSL, NAT, firewalling, and traffic engineering. They are often implemented using many processing cores. These cores are either arranged as a pool of identical cores (e.g., the Cavium CN68XX [23]), as a long pipeline of cores (e.g., the Xelerated X11 [24]), or as a combination of both (e.g., the EZChip NP-4 [25]).

These 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 encryption (like IPsec-VPN and SSL-VPN), LZS decompression, VoIP SBC, video CAC, per-subscriber queueing, and hierarchical classification for QoS [15], [23], [26], [27].

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 [24]) or longer processing times in run-for-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 [15]. 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 [15], [18]. 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.

In practice, another appeal of the multipass architecture is that 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 [26]. 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 [19] for IP control packets.

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 [18]). But little is known regarding the worst-case throughput of the various possible multipass NP schedulers.

The goal of this paper is to offer designs with proven performance guarantees for the multipass-NP scheduler. 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, including for traffic with complex processing needs.

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.

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 the buffer which reflects the impact multiple accesses to the buffer have system's throughput.

We design and analyze algorithms which aim at maximizing the overall value obtained by the system, which is affected by both the packet-level throughput (considered as benefit) an 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 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.

We note that due to space constraints, some of the proofs throughout the paper are omitted, and can be found in [9].

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 [19]. However, to our knowledge, there is no previous work in the literature that discusses the scheduling and buffer management policies in multipass NPs. Several other related topics have been studied in the context of NPs, namely, task mapping and load-balancing [8], [20]. 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 multi-threaded processors. A comprehensive overview of competitive online scheduling for server systems is provided in [17]. 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 [16]. Additional objectives, models, and algorithms have been studied extensively in this context (e.g., [5], [13], [14], [16], to name but a few). 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], [12]), shared memory switches with OQs (e.g., [7], [11]), and merging buffers (e.g., [10]). Some works also provide experimental studies of these algorithms and further validate their performance [2].

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

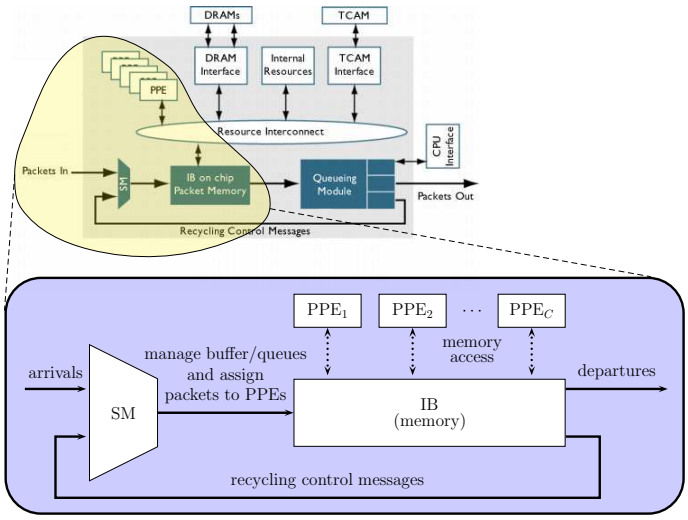


Fig. 1. An outline of the architecture model, as an abstraction of a standard Multipass NP Architecture (see, e.g. [26]).

QuantumFlow NP architecture [26]. 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.

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 [21], 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."

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.

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 [4], [22] 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 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 $\{1, \ldots, k\}$. The value of k will play a fundamental role in our analysis. We note, however, that none of our algorithms need 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 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 note that we will distinguish between the various embodiments of our algorithms also depending on the BM they employ in the IB. More specifically, we will focus our attention on two natural BMs:

1) FIFO: In this policy packets are serviced 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 serviced 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 which packet to discard in case of overflow.

Algorithm 1 ALG: Buffer Management Policy

```
upon the arrival of packet p:
if the buffer is not full then
    accept packet
else
    DecideIfPreempt(ALG,p)
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* (we remind that omitted proofs can be found in [9]).

Theorem 1. *The competitive ratio of any non-preemptive greedy buffer management policy $(B, FIFO, C, 0)$-system is at least $\frac{kB}{C}$, 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 $(B, PQ, C, 0)$-system is at least $\frac{kB}{C}$, 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 does have 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 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)

```
i ← first packet in IB_t^ALG s.t. r_t(p_i) = max_{i'} {r_t(p_{i'})}
                ▷ first in the order implied by the BM
if r(p) < r_t(p_i) then
    drop p_i and accept p
else
    reject p
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.[1] We refer to this algorithm as PQ_1.[2] The following theorem provides some guarantee as to its performance.

Theorem 3. *PQ_1 is optimal.*

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 3 which shows that as opposed to

[1] Packet p has a higher priority than packet q at time t if $r_t(p) < r_t(q)$.
[2] The reason for choosing the subscript 1 would become clear in section IV.

priority queueing, the performance of $FIFO_1$ can be rather far from optimal.

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

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 0 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 which 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 we are guaranteed to have this packet 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 which 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

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

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

We now turn to provide 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.*

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. For 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 overall 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 that would know in advance what packets are winners and would only accept those would be optimal. More formally, combining PQ_1 with a buffer admission control policy that would only accept the packets that ultimately depart using a given optimum scheduling policy can reach optimality.

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 required passes than q 's originally required passes. We note that that somewhat better performance can be obtained if one considers the *residual passes* instead of the original passes, however considering the original passes makes the analysis simpler. We further note that had we used residual passes for the preemption rule, the special case where $\beta = 1$ would have coincided with algorithm PQ_1 described in section III-B (hence the subscript 1).

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.

Algorithm 3 DECIDEIFPREEMPT(PQ_β,p)

$i \leftarrow$ first packet in $IB_t^{PQ_\beta}$ s.t. $r(p_i) = \max_{i'} \{r(p_{i'})\}$
if $r(p) < \frac{r(p_i)}{\beta}$ **then**
    drop p_i and accept p
else
    reject p
end if

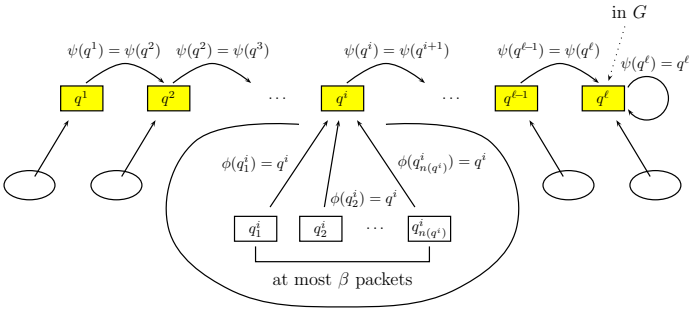


Fig. 2. Outline of mapping χ. Packet q^1 is admitted to the buffer upon arrival without preempting any packet, and henceforth packet q^i preempts packet q^{i-1}. Mapping ϕ maps $n(q^i) \leq \beta$ packets to q^i, and mapping ψ maps all $\ell \leq \log_\beta k$ packets, $q^1, \ldots, q^\ell$, to q^ℓ which is successfully delivered by G. This gives an overall of $O(\beta \log_\beta k)$ packets in $O \setminus G$ mapped to any single packet in G.

Theorem 7. *PQ_β is $(\frac{(\beta+1)(\log_\beta k - 1)(1-\alpha)}{1-\alpha \log_\beta k} + 1)$-competitive for any (B, PQ, C, α)-system, where $\alpha < \min\left\{1, \frac{1}{\log_\beta k}\right\}$.*

In the remainder of this section, we will 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 [9]. 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 $p \in A_2 \cup G$, $\left|\phi^{-1}(p)\right| \leq \beta$, i.e., there are at most β packets from A_1 mapped to any single packet in $p \in A_2 \cup G$ by ϕ. We then define a mapping $\psi : A_2 \cup G \mapsto G$ such that for every $p \in G$, $\left|\psi^{-1}(p)\right| \leq \log_\beta k$, i.e., there are at most $\log_\beta k$ packets from $A_2 \cup G$ mapped to any single packet in $p \in G$ by ψ. By composing these two mappings we obtain a mapping $\chi : O \setminus G \mapsto G$ such that for every $p \in G$, $\left|\chi^{-1}(p)\right| \leq \beta \log_\beta k$, i.e., there are at most $\beta \log_\beta k$ packets from $O \setminus G$ mapped to any single packet in $p \in G$ by χ. Figure 2 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.

C. 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 $p \in A_2 \cup G$, $\left|\phi^{-1}(p)\right| \leq \beta$, i.e., there are at most β packets from A_1 mapped to any single packet in $p \in A_2 \cup G$ by ϕ. 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 $n_t(p_i) = \left|\phi^{-1}(p_i)\right|$, 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 time slot).

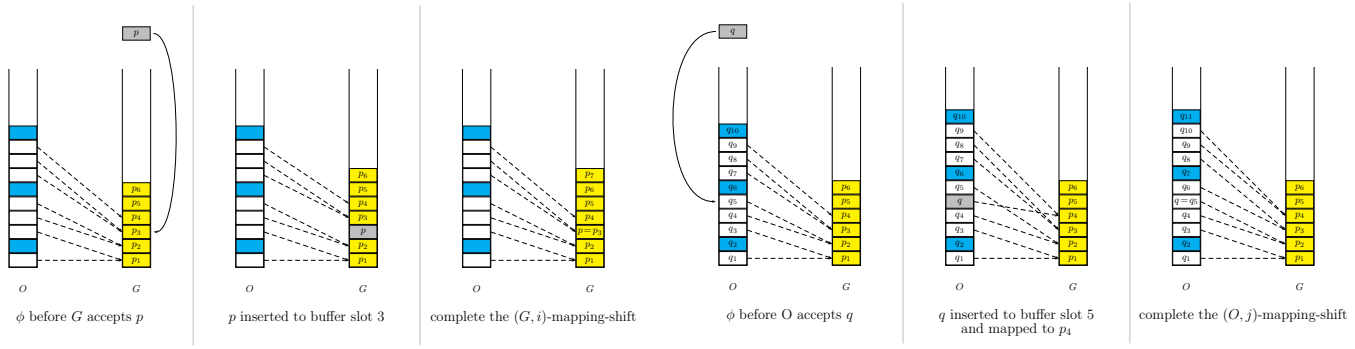
The mapping will be dynamically updated at each packet arrival as follows: Assume packet p arrives at time t. We distinguish between 3 cases:

1) If $p \in A_2$, or if it is not in both O and G (i.e., neither PQ_β nor OPT delivers them 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 $p \in G$, 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).

In order to finalize the description of ϕ, it remains to explain the notion of mapping-shifts. An (O, j)-mapping-shift begins by finding the minimal index i of a packet in the buffer of G with load at most β (i.e., $n_t(p_i) < \beta$) and defines $\phi(p) = p_i$ (we will show in Lemma 8 that this mapping is well defined and there always exists such an i). The next step of the (O, j)-mapping-shift is to adjust the mapping such that the set of packets mapped to any p_i is a subset of a contiguous block of slots in the buffer of O, and the size of any such set is at the maximal possible size subject to being at most β. See Figure 3(b) 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 3(a) gives an example of a (G, i)-mapping-shift.

The following lemma provides the essential properties of mapping ϕ.

Lemma 8. *Mapping ϕ is well defined, and at any time t and for every $p_i^t \in IB_t^{PQ_\beta}$, $\left|\phi^{-1}(p_i^t)\right| \leq \beta$.*



(a) (G,i)-mapping-shift (p is admitted by G to slot $i=3$)

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

Fig. 3. 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 in $A_2 \cup (O \cap G)$, and mapped white packets are in A_1. In both examples $\beta = 2$.

D. 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 essential property of ψ is given in the following lemma (proof omitted due to space constraints).

Lemma 9. *For every $p \in G$, $\left|\psi^{-1}(p)\right| \leq \log_\beta k$.*

E. Putting it All Together

By composing the mappings ϕ and ψ we obtain a mapping $\chi : A_1 \cap A_2 \mapsto G$ such that for every $p \in G$, $\left|\chi^{-1}(p)\right| \leq \beta \log_\beta k + (\log_\beta k - 1) = (\beta+1)\log_\beta k - 1$. This follows from the fact that every packet along the preemption sequence piggybacks at most β packets by ϕ, and 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 2 for an illustration of the mapping χ.

We can now turn to finalize the proof of Theorem 7. 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 therefore follows that

$$\begin{aligned}\frac{w(O)}{w(G)} &\leq \frac{w(O \setminus G)}{w(G)} + \frac{w(G)}{w(G)} \\ &\leq \frac{\max_{p \in G}\left|\chi^{-1}(p)\right||G|(1-\alpha)}{|G|} \cdot (1 - \alpha \log_\beta k) + 1 \\ &\leq \frac{(\beta+1)(\log_\beta k - 1)(1-\alpha)}{1 - \alpha \log_\beta k} + 1,\end{aligned}$$

which completes the proof of of the theorem.

Before we turn to describe our simulation setting and results, it would be instructive to discuss some of the consequences of Theorem 7. By optimizing the value of β one can obtain the minimum value for the competitive ratio (depending on the value of α). Table I gives an illustration of the optimal values of β and the competitive ratio they imply for $k = 10$.

α	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4
β	3.89	4.23	4.62	5.05	5.54	6.08	6.70	7.40
CR	7.96	8.75	9.63	10.62	11.75	13.01	14.44	16.05

TABLE I
OPTIMAL VALUES OF β, AND THE IMPLIED COMPETITIVE RATIO (CR), AS GIVEN BY THEOREM 7, FOR $k = 10$.

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 4 and 5 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 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 4(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 4(a)-4(c), and most notably in Figure 4(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.

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 4(b) and 4(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., the optimal values of β appearing in Table I for $k = 10$).

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.

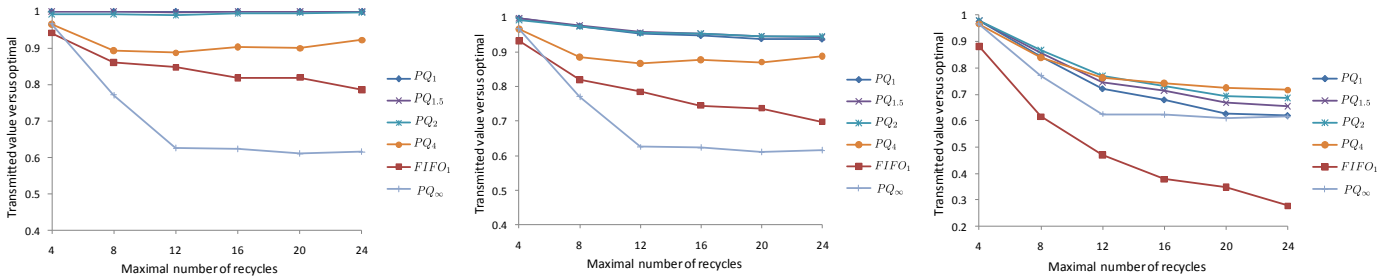
Figure 5(a) presents the results for a constant traffic arrival rate of $\lambda = 3$. Not surprisingly, the performance of all algorithms improves drastically as the number of cores increases. The increase in the number of cores essentially provides the network processor with a speedup proportional 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 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 5(b) and and 5(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

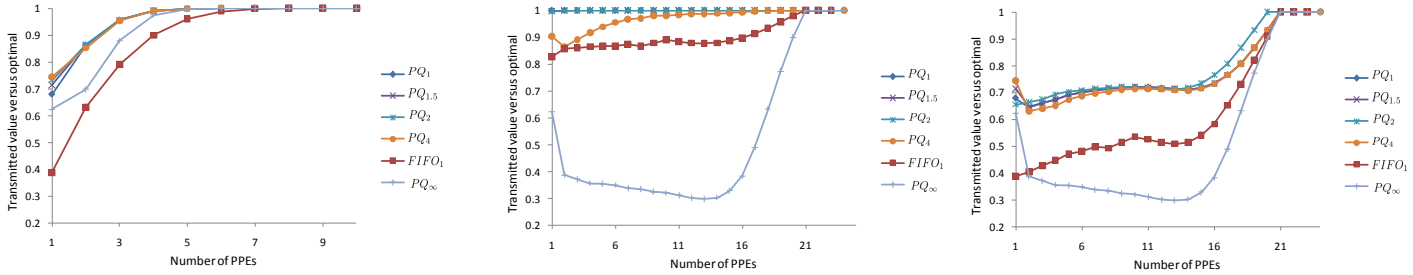
The increasingly-heterogeneous packet-processing needs of NP traffic are posing design challenges to NP architects. In this paper we provide performance guarantees for various algorithms within the multipass NP architecture, and further validate these results by simulations.

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 an admission control mechanism 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 one of the PPEs during the first pass. This is especially interesting because the first pass often corresponds



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

Fig. 4. 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).



(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. 5. Performance ratio of online algorithms versus optimal for different values of α, as a function of the number of cores C. In Figure 5(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 $\lambda = 0.3 \cdot C$).

to processing features that lead to the early dropping of packets, such as ACL.

REFERENCES

[1] W. Aiello, R. Ostrovsky, E. Kushilevitz and A. Rosen. Dynamic routing on networks with fixed-size buffers. SODA, pp. 771–780, 2003.

[2] S. Albers and T. Jacobs. An experimental study of new and known online packet buffering algorithms. ESA, pp. 754–765, 2007.

[3] N. Bansal, H. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. SODA, pp. 693-701, 2009.

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

[5] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs. On-line weighted flow time and deadline scheduling. J. Discrete Algorithms, 4(3):339–352, 2006.

[6] C. Chan and N. Bambos. Throughput loss in task scheduling due to server state uncertainty. *International Conference On Performance Evaluation Methodologies And Tools*, 2009.

[7] E. L. Hahne, A. Kesselman, and Y. Mansour. Competitive buffer management for shared-memory switches. *SPAA*, pp. 53–58, 2001.

[8] X. Huang and T. Wolf, Evaluating dynamic task mapping in network processor runtime systems. *TPDS*, 19(8):1086–1098, 2008.

[9] I. Keslassy, K. Kogan, G. Scalosub, M. Segal, Providing Performance Guarantees in Multipass Network Processors. Technical Report TR10-02, Comnet, Technion, Israel. [Online] http://www.ee.technion.ac.il/~isaac/p/tr10-02_multipass.pdf

[10] A. Kesselman, Z. Lotker, Y. Mansour, and B. Patt-Shamir. Buffer overflows of merging streams. *ESA*, pp. 349–360, 2003.

[11] A. Kesselman and Y. Mansour. Harmonic buffer management policy for shared memory switches. *TCS*, 324(2-3):161–182, 2004.

[12] A. Kesselman, Z. Lotker, B. Patt-Shamir. Y. Mansour, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. *SIAM Journal on Computing*, 33(3):563–583, 2004.

[13] S. Leonardi , D. Raz. Approximating total flow time on parallel machines. *STOC*, pp.110-119, 1997.

[14] R. Motwani , S. Phillips , E. Torng. Nonclairvoyant scheduling. TCS, 130(1):17-47, 1994.

[15] J. Mudigonda, H.M. Vin, R. Yavatkar. A case for data caching in network processors. *Unpublished manuscript*.

[16] S. Muthukrishnan , R. Rajaraman , A. Shaheen , J. Gehrke. Online Scheduling to Minimize Average Stretch. *FOCS*, pp. 433–442, 1999.

[17] K. Pruhs. Competitive online scheduling for server systems. *SIGMETRICS*, 34(4):52–58, 2007.

[18] T. Sherwood, G. Varghese, and B. Calder, A pipelined memory architecture for high throughput network processors. *ISCA*, pp. 288–299, 2003.

[19] C. Wiseman, *et al.* Remotely Accessible Network Processor-Based Router for Network Experimentation. *ANCS*, pp. 20–29, 2008.

[20] N. Weng and T. Wolf, Analytic modeling of network processors for parallel workload mapping. TECS, 8(3):1–29, 2009.

[21] T. Wolf, P. Pappu, and M. A. Franklin, Predictive scheduling of network processors. *Computer Networks*, 41(5):601–621, 2003.

[22] D. Sleator and R. Tarjan, "Amortized efficiency of list update and paging rules," *Commun. ACM*, 28(2):202–208, 1985.

[23] Cavium, OCTEON II CN68XX Multi-Core MIPS64 Processors, Product Brief, 2010. [Online] http://www.caviumnetworks.com/OCTEON-II_CN68XX.html

[24] Xelerated, X11 Family of Network Processors, Product Brief, 2010. [Online] http://www.xelerated.com/Uploads/Files/67.pdf

[25] EZChip, NP-4 Network Processor, Product Brief, 2010. [Online] http://www.ezchip.com/p_np4.htm

[26] Cisco, The Cisco QuantumFlow Processor, Product Brief, 2010. [Online] http://www.cisco.com/en/US/prod/collateral/routers/ps9343/solution_overview_c22-448936.html

[27] Juniper, Junos Trio, White Paper, 2009. [Online] http://www.juniper.net/us/en/local/pdf/whitepapers/2000331-en.pdf