Heterogeneous Packet Processing in Shared Memory Buffers

Patrick Eugster, Kirill Kogan, Sergey I. Nikolenko, Alexander V. Sirotkin

***Abstract*—Packet processing increasingly involves heterogeneous requirements. We consider the well-known model of a shared memory switch with bounded-size buffer and generalize it in two directions. First, we consider unit-sized packets labeled with an output port and a processing requirement (i.e., packets with heterogeneous processing), maximizing the number of transmitted packets. We analyze the performance of buffer management policies under various characteristics via competitive analysis that provides uniform guarantees across traffic patterns [10]. We propose the Longest-Work-Drop policy and show that it is at most 2-competitive and at least $\sqrt{2}$-competitive. Second, we consider another generalization, posed as an open problem in [19], where each unit-sized packet is labeled with an output port and intrinsic value, and the goal is to maximize the total value of transmitted packets. We show first results in this direction and define a scheduling policy that, as we conjecture, may achieve constant competitive ratio. We also present a comprehensive simulation study that validates our results.**

I. Introduction

The modern network edge is required to perform tasks with heterogeneous complexity, including, to list just a few, advanced VPN services, deep packet inspection, firewalling, and intrusion detection. This trend is further exacerbated by the advent of new network management models like Software-Defined Networking, which create new opportunities for advanced services. Each of these services may require to consider various characteristics (e.g., amount of processing or packet values) to achieve efficient implementation at the network processor and may present new challenges for traditional architectures, resulting in performance and implementation issues. Hence, the way how packets are processed may significantly affect throughput. For example, increasing per-packet processing requirements for some flows can increase congestion even for traffic with relatively modest burstiness characteristics.

A. Shared Memory Switch

The main functionalities of a network element include receiving packets on ingress ports, applying specific policies to them, identifying their destination ports, and sending them out through egress ports. When application-induced traffic bursts create an imbalance between incoming and outgoing packet rates to a given port, packets must be queued in the switch packet buffer. The available queue size on a port determines the port's ability to hold the packet until the egress port can emit it. When buffer queue entries are exhausted, packets are dropped, resulting in poor performance. The allocation and availability of the switch's buffer resources to its ports — determined not only by the buffer's size but also by the buffering architecture of the network element.

Overprovisioning in terms of buffer capacity at each network node to absorb bursty behavior is not viable, as networks do not have unlimited resources; quite conversely, cloud data centers can only scale out as fast as the effective per-port cost and power consumption. These factors, in turn, are driven by the chosen buffering architecture. The shared memory switch allows to absorb traffic bursts in the best way since the whole buffer can be utilized by a same output port if needed. Being an actual choice in practice [14] we focus our efforts in this paper on this type of buffer architecture.

The paradigm of shared memory switch in general is very flexible since it can support both complete sharing, where the same buffer serves all output ports, and complete partition, where each output port gets a *de facto* dedicated buffer. Complete sharing utilizes the entire buffer space but can hamper fairness: a single output port may monopolize shared memory and drop packets dedicated to other output ports. On the other hand, complete partitioning ensures fairness but may lead to significantly underutilized buffer space, losing more packets in case of congestion. But how to get the best of both worlds?

B. Throughput Optimization

There are two major tasks performed by network elements: (1) changing the properties of single packets (e.g., recoloring, encapsulation) and (2) changing the properties of packet streams (rate-limiting, shaping); the latter may require buffers. Devising a buffering architecture and its management is a fundamental problem in network switch design. To accommodate for network size, number of transport requests, and efficient reuse of network infrastructure, the Internet currently implements "best-effort" servicing that during congestion prioritizes some types of traffic over others to optimize desired objectives.

This work is an extended version of [16]. This version contains rewritten and extended Sections I, II, and V, new Figure 4, Tables I and II, new examples that illustrate the proposed algorithms.

P. Eugster is with Departments of Computer Science of Purdue University and TU Darmstadt; e-mail: peugster@cs.purdue.edu. His work was partially supported by Cisco Systems through grant "A Fog Computing Architecture", the German Research Foundation through project "Multi-Mechanism Adaptation for Future Internet" and the European Research Council through project "Lightweight Verification of (Distributed) Software".

K. Kogan is with IMDEA Networks Institute; e-mail: kirill.kogan@imdea.org.

S.I. Nikolenko is with the National Research University Higher School of Economics, St. Petersburg, and with Steklov Mathematical Institute at St. Petersburg; e-mail: sergey@logic.pdmi.ras.ru. His work was partially supported by the Government of the Russian Federation grant 14.Z50.31.0030.

A.V. Sirotkin is with National Research University Higher School of Economics, St. Petersburg; e-mail: avsirotkin@hse.ru.

Efficient methods for buffer management beyond *fairness* objective functions lead to new challenges in performance and implementation for traditional switch architectures. Inherited from the Internet, fairness is in fact a design choice which can conflict with other objectives in various economic models (e.g., utilization of network infrastructure, or profit [8], [21], [22]). Furthermore, utilization of network infrastructure and profit can be formulated as a weighted throughput optimization.

C. Admission Control - beyond Buffer Occupancy

The modern network edge is required to perform tasks with heterogeneous complexity, including features such as advanced VPN services, deep packet inspection, firewalling, and intrusion detection (to list just a few). This trend is further exacerbated by the advent of new paradigms like Software-Defined Networking [35], Fog Computing [9] which create new opportunities for advanced services.

Implementing such services, which give rise to different *traffic types*, requires considering various parameters of packets (different amounts of processing or intrinsic values) to achieve efficient implementation at the network processor. This leads to new challenges in performance and implementation for traditional architectures.

Most admission control policies are based on a simple characteristic as buffer occupancy, whereas traffic has additional important characteristics such as processing requirements and values that are not explicitly taken into account.

Because of the store-and-forward nature of packet processing — in contrast to OS scheduling — packets are processed in a "run-for-completion" manner where no two different cores can access the same packet during its processing to avoid expensive rescheduling [11], [13][1]. In this setting, an efficient input buffer architecture and management become crucial. The best way to accommodate traffic burstiness is to adopt a single queue architecture, where the whole buffer is shared among all types of traffic and every core is able to process any type of traffic (see the top of Fig 1). In this architecture an online greedy policy that implements priority queuing (PQ), where packets are ordered in non-decreasing order of their processing requirement and can be pushed out upon arrival of new packets, has optimal throughput [23].

However this simple approach has important drawbacks. First, PQ order is required: with a simpler to implement FIFO processing order the competitive ratio degrades to $\Omega(\log k)$ with respect to the optimal clairvoyant algorithm [31], where k is the maximal possible processing per packet. Second, it can cause starvation of packets with higher processing requirements, which implicitly leads to setting priorities among traffic types and thus services so that they are rigged to the inverse of the processing requirements. Second, handling several traffic types on individual cores leads to increased complexity of the programs loaded onto these cores, as these must implement several services in a combined manner.

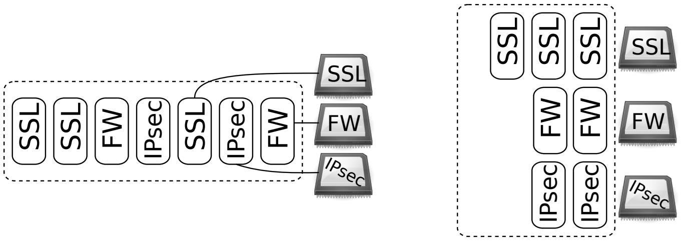


Fig. 1: Two different buffer management models. Left: each core can process any traffic type, IPsec, SSL, or firewall (FW); right: each core processes a specific traffic type from its own queue, but all queues share the same buffer.

We thus consider a shared memory switch where the buffer is shared among all types of traffic but in contrast to the previous single queue model each core processes only one type of traffic. The single queue buffer architecture is in fact a special case of a shared memory switch (see the bottom of Fig 1). Since all arriving packets with the same output port label have the same traffic type, the implementation of advanced processing order is not required (regular FIFO suffices). Furthermore, there is no starvation of any specific type of traffic.

D. Our Contributions

To study performance, we employ *competitive analysis* [10], [40]. An algorithm ALG is said to be *α-competitive* for some $\alpha \geq 1$ if for any arrival sequence, the number of packets transmitted by ALG is at least $1/\alpha$ times the number of packets transmitted by an optimal offline clairvoyant algorithm OPT.

In this paper we consider the crucial problem of throughput maximization. We generalize the previous shared memory switch model with uniform values and required processing [2] and consider two different packet characteristics that define a traffic type together with output port labels: (1) required processing per packet and (2) packet value. Our main contribution is to identify properties of an "ideal" online policy that maximizes throughput and achieve a constant competitive ratio [10]. We argue that required processing and packet values have different natures. In particular, we show that to achieve a constant competitive ratio in the case of required processing, it is enough to consider the total work per queue but the normalized work per queue cannot be used to achieve a constant competitiveness. In the value case however (where throughput is measured by total transmitted value) the total value per queue constitutes a poor choice but normalized value can potentially achieve constant competitiveness. Most considered algorithms are greedy (accept all arrivals if there is enough buffer space) and hence allow for simple implementations.

In the first part of this work (1) we consider *heterogeneous processing requirements*, as opposed to shared memory switches with a single processing cycle per packet [2], [28]. In Section III, we consider a shared memory architecture where an arriving packet has an output port label and amount of required processing in cycles, with the constraint that

[1] The admission control and processing are two separate logical entities in network processor architectures (e.g., [11], [13]), where only during processing multiple threads can access a shared memory in parallel. Since processing in these network processors is implemented in run-for-completion manner there is no real contention during processing. From the other side the admission control in these architectures are implemented as a centralized unit with a desired value of speedup.

packets accepted to a same queue have the same processing requirements (two different queues still can accept packets with the same required processing, see Fig. 2). Our main result is to exhibit a novel Longest-Work-Drop (LWD) policy that is at most 2-competitive. When processing requirements are uniform, LWD emulates the well-known Longest-Queue-Drop (LQD) policy [2]. As a result, LWD is at least $\sqrt{2}$-competitive. In the special case when all queues accept packets with different processing requirements LWD is at least $\left(\frac{4}{3}-\frac{6}{B}\right)$-competitive for $k \geq 6$, where B is the buffer size. We also demonstrate a number of lower bounds for several policies that have good performance characteristics for homogeneous processing requirements. Namely, we show that:

(1) LQD is at least $(\sqrt{k}-o(\sqrt{k}))$-competitive, where k is the maximum processing required for packets;

(2) the Harmonic policy [28], which is at most $O\left(\log n\right)$-competitive in the model with a single processing cycle per packet and n output ports, is at least $\left(\frac{1}{2}\sqrt{k\ln k}-o(\sqrt{k\ln k})\right)$-competitive in our model;

(3) the Biggest-Packet-Drop (BPD) policy that attempts to keep packets with lowest processing requirements is at least $(\ln k+\gamma)$-competitive for $\gamma = 0.5772\ldots$.

In the second part (2) we consider a different yet related model of shared memory switch, where each unit-sized packet is labeled with an output port and an *intrinsic value* (Section IV). The objective is to optimize the total value of transmitted packets. The existence of a scheduling policy achieving a constant competitive ratio here was presented as an open question in SIGACT News [19, p. 22]. To the best of our knowledge, we provide the first results addressing this problem as follows. First we show that the above-mentioned LQD policy is at least $\left(\sqrt[3]{k}-o\left(\sqrt[3]{k}\right)\right)$-competitive. Furthermore, we define a Minimal-Value-Drop (MVD) policy (that equivalent to BPD) that in the case of congestion drops a packet with minimal value. We show that MVD is at least $\frac{m-1}{2}$-competitive for $m = \min\{k, B\}$, where k is the maximal packet value and B is the buffer size. As a result, an ideal policy should address a fundamental tradeoff between a number of "active" ports (like LQD) and maximization of total admitted value (like MVD) to achieve a constant competitiveness (similar to LWD in the first model considered in this paper). We thus define a Maximal-Ratio-Drop (MRD) policy that pushes out a packet from a queue with maximal ratio $\frac{|Q_j|}{a_j}$, where a_j is the average value in the j-th output queue Q_j and $|Q_j|$ is the size of Q_j. The competitiveness of MRD is at least $\sqrt{2}$. We also conduct a comprehensive simulation study validating our theoretical results in Section V.

II. Related Work

Our current work can also be viewed as part of a larger research effort concentrated on studying competitive algorithms for buffer management of bounded buffers. Initiated in [27], [33], this line of research has received tremendous attention over the past decade. Various models have been proposed and studied, including QoS-oriented models where packets have individual weights [3], [15], [27], [33] and models where packets have dependencies [24], [34]. A related field that has recently attracted much attention focuses on various switch architectures and aims to design competitive algorithms for multi-queued scenarios therein (cf. [4], [6], [7], [25], [26]). However, these models do not cover the case of packets with heterogeneous processing requirements.

Pruhs [39] provides a comprehensive overview of competitive online scheduling for server systems. Note that scheduling for server systems usually concentrates on average response time, while we focus mostly on throughput. Furthermore, scheduling of server systems does not allow jobs to be dropped, which is an inherent aspect of our model due to size limitations on buffers.

The efforts most closely related to the present work are those of Aiello et al. [2] and of Kesselman and Mansour [28]. Aiello et al. [2] propose a non-push-out buffer management policy called Harmonic that is at most $O(\log n)$-competitive and establish a lower bound of $\Omega(\frac{\log n}{\log\log n})$ on the performance of any online non-push-out deterministic policy, where n is the number of output ports. Kesselman and Mansour [28] demonstrate that the LQD policy is at most 2- and at least $\sqrt{2}$-competitive. Both these works consider packets with an invariably single required processing cycle and the proposed algorithms have no good competitiveness results in the models with heterogeneous processing requirements and packet values.

Another line of closely related research considers packets with heterogeneous processing requirements. In particular, the works [5], [23], [29], [31] discuss the impact of processing order and admission control policies for a single queue architecture, where packets have heterogeneous processing requirements with uniform values. Kogan et al. [32] studied the model for packets with heterogeneous length and processing requirements. Recently, Chuprikov et al. [12] considered packets with both heterogeneous processing and heterogeneous values, proving a general lower bound on any online algorithm in this setting. Kogan et al. [30] evaluate various buffer management policies for multiple separated queues in various settings. A survey by Nikolenko et al. [36] provides a good overview of this field.

III. Packet Scheduling with Heterogeneous Processing Requirements

A. Model Description

We consider an $l \times n$ shared memory switch with a buffer of size B; l and n represent the number of input and output ports, respectively. We assume that $B \geq n$. Each i^{th} output port manages a single output queue Q_i, $1 \leq i \leq n$; the number of packets in Q_i is denoted by $|Q_i|$. Each Q_i implements first-in-first-out (FIFO) processing order. Each packet $p(d, w)$ arriving at an input port is labeled with the output port number d and its required work w in processing cycles ($1 \leq d \leq n$ and $1 \leq w \leq k$), where k denotes the global upper bound on required work per packet. In what follows, we denote by $\boxed{w}$ a packet with required work w; by $h \times \boxed{w}$, a burst of h packets with required work w each.

Time is slotted; we divide each time slot into two phases (see Fig. 2): During the (1) *arrival phase* a burst of new packets arrives at each input port, and the *(buffer management)*

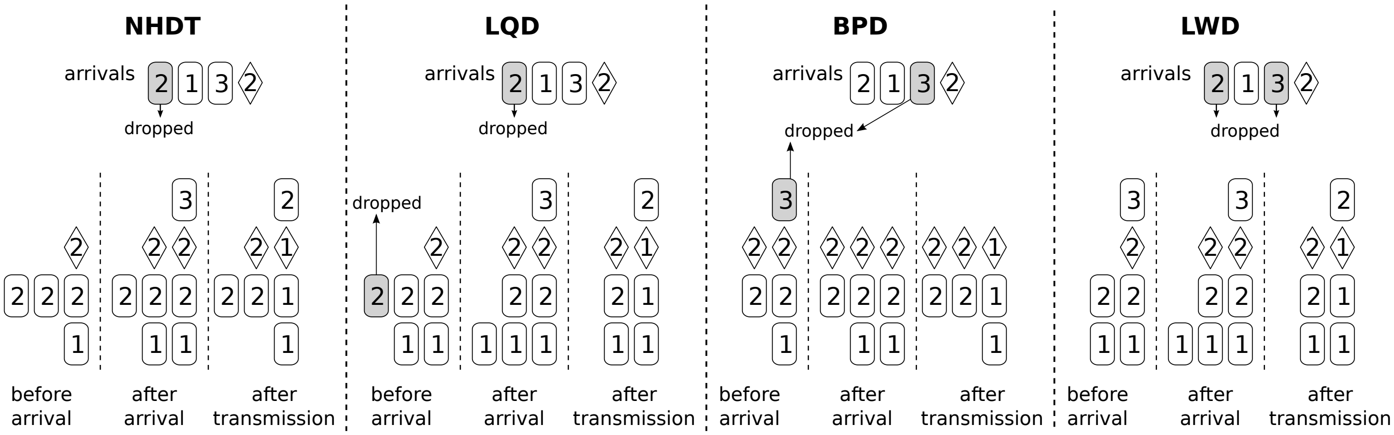


Fig. 2: A sample time slot of Non-Push-Out-Harmonic-Dynamic-Threshold (NHDT), Longest-Queue-Drop (LQD), Biggest-Packet-Drop (BPD), and Longest-Work-Drop (LWD) policies with maximal processing $k = 3$, 4 output ports (there are two different ports with the same processing requirement 2, denoted by rectangular and rhomboidal packets respectively), and a shared buffer of size $B = 8$.

Algorithm	Lower bound	Upper bound
Homogeneous processing		
General	$\frac{4}{3}$	-
General non-push-out	$\Omega(\frac{\log n}{\log\log n})$	-
NHDT	-	$O(\log n)$
LQD	$\sqrt{2}$	2

TABLE I: Results summary: lower and upper bounds for homegeneous processing requirements; n is a number of output ports.

policy decides which ones should be admitted; we assume that during the arrival phase input ports are served in a fixed order from input port 1 to input port l with no restriction on the burst size (at the same time slot more than l arrivals are allowed); the arrivals are adversarial and do not assume any specific traffic distribution except a single constraint: all packets arriving with output port label i have the same required processing w_i, $1 \le w_i \le k$. Note that two different output queues Q_i and Q_j, $1 \le i, j \le n$ can still accept packets with the same processing requirement, $w_i = w_j$. Without loss of generality, we assume that queues are sorted in the order of their processing requirement: if $i < j$ then $w_i \le w_j$. An accepted packet can be later dropped from the buffer when another packet is accepted instead; in this case we say that a packet p is *pushed out* by another packet q, and a policy that allows this is called a *push-out* policy. During the (2) *transmission phase*, required work of the head-of-line packet in FIFO order at each non-empty queue is reduced by one, and every packet with zero residual work is transmitted.

B. Choosing the Right Properties for Buffer Management

Next we attempt to explore properties of a "perfect" policy that provides constant competitiveness — if it exists — in the generalized model with heterogeneous processing requirements and for any switch *configuration*, that is, any assignment of specific processing requirements to cores. We consider, in turn, several algorithms that are either simple, look like natural candidates, or have been proven to be efficient for the case of uniform processing [2], [28] (see Table I).

For each algorithm, we show a fast growing lower bound that indicates that this algorithm does not really extend competitively to our generalized model with heterogeneous processing. Rather than choosing respective corner-case configurations for each considered policy, i.e., assigning required processing values to output ports in a specific maximally adversarial way for every policy, we show all lower bounds in the same configuration of the model above: the case when there are exactly k output ports, and queue Q_i contains packets with required processing i. We call this the *contiguous case* since required processing in different queues is ordered in a contiguous sequence from 1 to k. Even in the contiguous case, we show that established policies do not rise up to the challenges of heterogeneous processing, showing large lower bounds for each of them.

Thus, a good policy has to account for the processing requirements explicitly; still, it remains to learn how to account for it. We first consider the BPD policy that drops packets with largest required processing in case of congestion, but also show a fast growing lower bound for BPD. Then we proceed to the LWD policy for which we prove (in the general case, not only the contiguous case) a constant upper bound, the main result of this paper.

1) Non-Push-Out Policies: In the case of a single queue, non-push-out greedy policies perform poorly (they are k-competitive) on packets with heterogeneous processing regardless of processing order [23], [31]. In our model, we begin with two simple cases of greedy non-push-out buffer management policies with static thresholds. The first policy sets static thresholds on each queue inversely proportional to required processing, while the second simply uses identical thresholds for all queues. Interestingly, the second policy has a better competitive ratio. We denote $Z = \sum_{i=1}^{n} \frac{1}{r_i}$ (sum of inverse values of required processing for each port); also, in what follows we omit $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ for clarity, assuming that B divides all we need it to divide.

Non-Push-Out-Harmonic-Static-Threshold (NHST): during the arrival of a packet p with output port i, if $|Q_i| < \frac{B}{r_i Z}$, accept p into Q_i; else drop p.

Theorem 1. *NHST is $(kZ + o(kZ))$-competitive.*

Proof. For the lower bound, consider a burst of $B \times \boxed{k}$. NHST will only accept $\frac{B}{kZ}$ of them, while OPT is free to accept all. After all packets are processed, the process repeats, getting the kZ lower bound. The matching upper bound in this case follows since all queues are isolated and process packets separately, so worst case analysis is reduced to a single queue architecture. □

Non-Push-Out-Equal-Static-Threshold (NEST): during the arrival of a p with output port i, if $|Q_i| < B/n$ packets then accept p into Q_i, else drop p.

Theorem 2. *NEST is $(n + o(n))$-competitive.*

Proof. Each queue is simply a separate homogeneous (and hence optimal) queue with buffer size $\frac{B}{n}$. □

Next we consider the Non-Push-Out-Harmonic-Dynamic-Threshold (NHDT) policy, previously considered in [28]. In the NHDT policy, thresholds for queues are dynamic and depend on the number of packets in the queues. The idea is that for each $1 \le m \le k$, m queues with most packets in them should contain at most $\frac{B}{H_k}\left(1 + \frac{1}{2} + \ldots + \frac{1}{m}\right)$ packets, where $H_k = 1 + \frac{1}{2} + \ldots + \frac{1}{k}$ is the k-th harmonic number.

Non-Push-Out-Harmonic-Dynamic-Threshold (NHDT): during the arrival of a packet p with output port i, denote by $j_1, j_2, \ldots, j_m = i$ the queues for which $|Q_{j_s}| \ge |Q_i|$; if

$$\sum_{s=1}^{m} |Q_{j_s}| < \frac{B}{H_k}\left(1 + \frac{1}{2} + \ldots + \frac{1}{m}\right)$$

then accept p into Q_i, else drop p.

The NHDT policy is illustrated with an example on Fig. 2; it is the first policy on the left. Initial buffer state is shown on the left: $|Q_3| = 0$, $|Q_2^{\Diamond}| = 1$, $|Q_2^{\Box}| = 3$ and $|Q_1| = 1$ for $B = 8$ and $k = 3$, so there are two free slots in the buffer. Here we have $H_k = 1 + \frac{1}{2} + \frac{1}{3} = \frac{11}{6}$, and $\frac{B}{H_k} = \frac{36}{11}$. There arrive four packets, $\boxed{2}^{\Diamond}$, $\boxed{3}$, $\boxed{1}$, and $\boxed{2}$ (arriving in this order). When $\boxed{2}^{\Diamond}$ arrives, it is accepted because the dynamic threshold for two longest queues is $(1 + \frac{1}{2})\frac{36}{11} = \frac{57}{11} > 5$. When $\boxed{3}$ arrives, it is accepted to the empty queue. The reasoning for $\boxed{1}$ is the same as $\boxed{2}^{\Diamond}$. When $\boxed{2}$ arrives, it is dropped because the dynamic threshold for one longest queue is $\frac{36}{11} < 4$. Thus, after all arrivals we have $|Q_3| = 1$, $|Q_2^{\Diamond}| = 1$, $|Q_2| = 3$, and $|Q_1| = 2$, and processing transmits one $\boxed{1}$ and reduces the processing of the other three head-of-line packets by one.

For packets with homogeneous processing, NHDT was shown to be at most $O(\log n)$-competitive for n output ports [28] (in our model, there are k output ports). It turns out that for heterogeneous processing requirements, NHDT is still lacking: although we have not been able to prove a linear bound similar to Theorem 1 for NHST, the following lower bound still grows too fast.

Theorem 3. *If B is asymptotically greater than k, the NHDT policy is at least $\left(\frac{1}{2}\sqrt{k\ln k} - o(\sqrt{k\ln k})\right)$-competitive in the sequential case.*

Proof. We denote $A = \frac{B}{\ln k}$. To show the lower bound, consider an input burst consisting of $B(m+1)$ packets with processing requirements k, $k-1$, …, $k-m$, and finally 1; here m is a number from 1 to $k-1$ that will be chosen below.

The packets arrive in reverse order: first $B \times \boxed{k}$, then $B \times \boxed{k-1}$, and so on. As a result, NHDT accepts $A \times \boxed{k}$, $\frac{A}{2} \times \boxed{k-1}$, …, $\frac{A}{k-m+1} \times \boxed{1}$. OPT, on the other hand, accepts only one packet for each of the queues from k to $(k-m)$ and fills the rest of its buffer with $\boxed{1}$s, accepting $(B-k+m) \times \boxed{1}$. Then every i^{th} time slot, another $\boxed{i}$ arrives for $k-m \le i \le k$, so OPT's queues are always busy.

In $\frac{A}{k-m+1}$ rounds of processing, NHDT transmits all of its $\boxed{1}$s and then proceeds to process only higher queues. In $B-k+m-\frac{A}{k-m+1}$ more rounds, OPT also runs out of $\boxed{1}$s. By this time ($B-k+m$ time slots in total), OPT has processed $(B-k+m)(1+H_k-H_m)$ packets, while NHDT has processed $(B-k+m)(H_k-H_m)+A/(k-m+1)$ packets; then another large burst arrives, and the process is repeated, getting the competitive ratio

$$\frac{1+H_k-H_m}{H_k-H_m+\frac{A}{(B-k+m)(k-m+1)}} \approx \frac{\ln k-\ln m+1}{\ln k-\ln m+\frac{1}{(k-m)\ln k}}$$

(discarding asymptotically smaller terms and assuming that asymptotically B is greater than k).

It now remains to optimize this ratio by choosing suitable m; setting $m = k - \sqrt{\frac{k}{\ln k}}$, since $-\ln\left(1-\sqrt{\frac{1}{k\ln k}}\right) \approx \sqrt{\frac{1}{k\ln k}}$, we get the ratio of

$$\frac{-\ln\left(1-\sqrt{\frac{1}{k\ln k}}\right)+1}{-\ln\left(1-\sqrt{\frac{1}{k\ln k}}\right)+\sqrt{\frac{1}{k\ln k}}} \approx \frac{1}{2}\sqrt{k\ln k} - o(\sqrt{k\ln k}).$$

□

At this point, it is unclear how to generalize NHDT to heterogeneous processing better; this remains an interesting problem for future research.

2) Push-Out Policies: We now proceed to push-out policies, starting with the well-known Longest-Queue-Drop that overlooks required processing and only considers queue sizes.

Longest-Queue-Drop (LQD): during the arrival of a packet p with output port i, denote by $j^* = \arg\max_j\{|Q_j| + [i = j]\}$ where $[i = j] = 1$ if $i = j$ and 0 otherwise (i.e., Q_{j^*} is the longest queue once we virtually add p to Q_i; we choose one with largest required processing if there are several); then do the following:

(1) if the buffer is not full, accept p into Q_i;
(2) if the buffer is full and $i \ne j^*$, push out last packet from Q_{j^*} and accept p into Q_i; else drop p.

LQD is illustrated on Fig. 2; it is the second policy on the left. The initial buffer state is, again, $|Q_3| = 0$, $|Q_2^{\Diamond}| = 1$, $|Q_2^{\Box}| = 3$ and $|Q_1| = 1$ for $B = 8$ and $k = 3$, There arrive four packets, $\boxed{2}^{\Diamond}$, $\boxed{3}$, $\boxed{1}$, and $\boxed{2}$ (arriving in this order). When $\boxed{2}^{\Diamond}$ and $\boxed{3}$ arrive, they are accepted since the buffer is not full yet. When $\boxed{2}$ arrives, it is dropped because the buffer is full, and Q_2 is already the longest queue. When $\boxed{1}$ arrives, it would make Q_1 as long as Q_2, so considering that it has a smaller processing requirement, it is accepted, and $\boxed{2}$ is dropped from the buffer; thus, at this point we have $|Q_3| = 1$, $|Q_2| = 2$, and $|Q_1| = 3$. When $\boxed{2}$ arrives, it would make Q_2 as long as Q_1, so it is dropped since it has a larger processing requirement. Therefore, after arrival we have $|Q_3| = 1$, $|Q_2^{\Diamond}| = |Q_2^{\Box}| = 2$, $|Q_1| = 3$, and processing transmits one $\boxed{1}$ and reduces the processing of the other three head-of-line packets by one.

In case of homogeneous processing, LQD is at least $\sqrt{2}$- and at most 2-competitive [2]. For heterogeneous required processing, the situation is worse.

Theorem 4. *For sufficiently large B, LQD is at least $(\sqrt{k} - o(\sqrt{k}))$-competitive in the sequential case.*

Proof. We introduce a parameter m to be defined later. Over the first burst, there arrive B packets of each of the following kinds:

$$\boxed{1}, \boxed{k}, \boxed{k-1}, \ldots, \boxed{k-m+1}.$$

LQD evenly distributes the packets among queues and has B/m packets in each of its nonempty queues (throughout the proof we assume that B is large and is divisible by everything we need it to be). OPT accepts $(B - m + 1) \times \boxed{1}$ and one each in the remaining queues. Packets with required processing from $k - m + 1$ to k keep coming so OPT always has packets in these queues to work on, but there are no more $\boxed{1}$s. Thus, after B/m time slots LQD runs out of $\boxed{1}$'s; by this time, both LQD and OPT have processed $\frac{B}{m} + \frac{B}{m}\beta_{k,m}$ packets, where $\beta_{k,m} = \frac{1}{k} + \frac{1}{k-1} + \ldots + \frac{1}{k-m+1}$, and OPT still has $\left(B\frac{m-1}{m} - m\right) \times \boxed{1}$ in the first queue. Therefore, over the next $B\frac{m-1}{m}$ steps LQD processes $B\frac{m-1}{m}\beta_{k,m}$ packets while OPT processes $B\frac{m-1}{m}(1 + \beta_{k,m}) - m$ packets. The total ratio is

$$\frac{\frac{B}{m} + \frac{B}{m}\beta_{k,m} + B\frac{m-1}{m}(1 + \beta_{k,m}) - m}{\frac{B}{m} + \frac{B}{m}\beta_{k,m} + B\frac{m-1}{m}\beta_{k,m}} = 1 + \frac{\frac{m-1}{m} - \frac{m}{B}}{\frac{1}{m} + \left(1 - \frac{m}{B}\right)\beta_{k,m}} - O(1/Bm).$$

To optimize the bound with respect to m, we choose $m = k^{\alpha}$, approximate the bound for large B as $1 + \frac{k^{\alpha}-1}{1+k^{\alpha}\beta_{k,m}}$, and recall that

$$\beta_{k,m} = \frac{1}{k} + \ldots + \frac{1}{k-m+1} = \ln\frac{k}{k-m} + o(\frac{1}{m}) = \ln\frac{k}{k-k^{\alpha}} + o(k^{-\alpha}).$$

For $\alpha < 1$ and large k,

$$\ln\frac{k}{k-k^{\alpha}} = -\ln(1 - k^{\alpha-1}) = k^{\alpha-1} + o(k^{\alpha-1}),$$

which yields the bound as

$$1 + \frac{k^{\alpha}}{1 + k^{\alpha}k^{\alpha-1}} + o(k^{-\alpha}) = 1 + \frac{k^{\alpha}}{1 + k^{2\alpha-1}} + o(k^{-\alpha}),$$

and this fraction reaches its maximum of $\sqrt{k} - o(\sqrt{k})$ for $\alpha = 1/2$. □

Next we propose a policy that aims to minimize total required processing in case of congestion by pushing out packets with maximal processing requirements.

Biggest-Packet-Drop (BPD): during the arrival of a packet p with output port i, denote by Q_j the nonempty queue with largest processing requirement; then do the following:

(1) if the buffer is not full, accept p into Q_i;
(2) if the buffer is full and $i \leq j$, push out last packet from Q_j and accept p into Q_i;
(3) if the buffer is full and $i > j$, drop p.

Figure 2 shows a sample time slot of the BPD policy; it is the second policy on the right. The initial buffer state is now $|Q_3| = 1$, $|Q_2^{\Diamond}| = 2$, $|Q_2^{\Box}| = 2$, and $|Q_1| = 1$ for $B = 8$ and $k = 3$, There arrive four packets, $\boxed{2}^{\Diamond}$, $\boxed{3}$, $\boxed{1}$, and $\boxed{2}$ (arriving in this order). When $\boxed{2}^{\Diamond}$ and $\boxed{3}$ arrive, they are accepted since the buffer is not full yet. When $\boxed{1}$ arrives, it pushes out $\boxed{3}$ as the biggest packet, and we have $|Q_3| = 1$, $|Q_1| = 3$. When $\boxed{2}$ arrives, it pushes out another $\boxed{3}$, so there are none of them left. Therefore, after arrival we have $|Q_3| = 0$, $|Q_2^{\Diamond}| = |Q_2^{\Box}| = 3$, and $|Q_1| = 1$, and processing transmits one $\boxed{1}$ and reduces the processing of two packets with required processing 2 by one.

Theorem 5. *For $B \geq \frac{k(k+1)}{2}$, BPD is at least $(\ln k + \gamma)$-competitive in the sequential case, where $\gamma = 0.5772\ldots$ is the Euler–Mascheroni constant.*

Proof. The counterexample is the following: every time slot, there arrive $B \times \boxed{1}$, $B \times \boxed{2}$, …, $B \times \boxed{k}$ (a full set of packets); BPD accepts only $B \times \boxed{1}$ and keeps processing only 1 packet per time slot, i.e., $k!$ packets per $k!$ time slots, while OPT is free to accept the packets evenly and get $k! + k!/2 + \ldots + k!/k$ packets per $k!$ time slots, getting the bound as

$$1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} = H_k \geq \ln k + \gamma.$$

□

As an improvement over LQD, we propose a policy that pushes packets out of the queue with most required work; we denote the total work in queue Q_i (sum of remaining work for packets in Q_i) by W_i.

Longest-Work-Drop (LWD): during the arrival of a packet p with output port i, let $j^* = \arg\max_j\{W_j + [i = j]r_i\}$ (i.e., W_{j^*} is maximal once we virtually add p to Q_i; we choose maximal among those queues if there are several); then do the following:

(1) if the buffer is not full, accept p into Q_i;
(2) if the buffer is full and $i \neq j^*$, push out last packet from Q_{j^*} and accept p into Q_i; else drop p.

LWD is the rightmost policy illustrated on Fig. 2. The initial buffer state is now $|Q_3| = 1$, $|Q_2^{\Diamond}| = 1$, $|Q_2^{\Box}| = 2$, and $|Q_1| = 2$ for $B = 8$ and $k = 3$, There arrive four packets, $\boxed{2}^{\Diamond}$, $\boxed{3}$, $\boxed{1}$, and $\boxed{2}$ (arriving in this order). When $\boxed{2}^{\Diamond}$ and $\boxed{3}$ arrive, they are accepted since the buffer is not full yet. When $\boxed{1}$ arrives, it is accepted and $\boxed{3}$ is dropped because $W_3 = 6$ is the largest work (we could have pushed out $\boxed{2}$ instead, the queues are identical). When $\boxed{2}$ arrives, it is dropped because with is, we would have $W_2 = 6$ to be the largest work. Therefore, after arrival we have $|Q_3| = 1$, $|Q_2^{\Diamond}| = 1$, and $|Q_2| = |Q_1| = 3$, and processing transmits one $\boxed{1}$ and reduces the processing of three other head-of-line packets by one.

Theorem 6. *If $k \geq 6$, LWD is at least $\left(\frac{4}{3} - \frac{6}{B}\right)$-competitive in the sequential case.*

Proof. Over the first burst, there arrive $B \times \boxed{1}$, $\frac{B}{4} \times \boxed{2}$, $\frac{B}{6} \times \boxed{3}$, and $\frac{B}{12} \times \boxed{6}$; LWD accepts $\frac{B}{2} \times \boxed{1}$ and all the other packets, while OPT accepts one of each larger packets and, correspondingly, $(B-3) \times \boxed{1}$. Then packets with required processing 2, 3, and 6 keep arriving as needed to keep OPT's queues busy. Thus, over the first $\frac{B}{2}$ time slots both OPT and LWD process $\frac{B}{2}\left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{6}\right) = B$ packets, but LWD has now run out of $\boxed{1}$ packets while OPT still has $\frac{B}{2} - 3$ more, so over the next $\frac{B}{2} - 3$ time slots OPT processes $\frac{B}{2} - 3 + \left\lfloor \frac{B/2-3}{2} \right\rfloor + \left\lfloor \frac{B/2-3}{3} \right\rfloor + \left\lfloor \frac{B/2-3}{6} \right\rfloor \geq B - 9$ packets while LWD can only process $\left\lfloor \frac{B/2-3}{2} \right\rfloor + \left\lfloor \frac{B/2-3}{3} \right\rfloor + \left\lfloor \frac{B/2-3}{6} \right\rfloor \leq \frac{B}{2}$ packets. After that, the first burst arrives again, and the sequence repeats itself, getting the bound as $\frac{B+B-9}{B+B/2} = \frac{4}{3} - \frac{6}{B}$. □

Since LWD is equivalent to LQD when packets in all queues have the same processing requirement, LWD is at least $\sqrt{2}$-competitive in this setting [2]. Note that our lower bound for LWD in the contiguous case is less than the $\sqrt{2}$ lower bound of LQD in the model with uniform processing requirements. LWD optimizes a "local" state by dropping packets from queues with maximal latency. On the other hand, OPT can leave these packets and keep these ports active when OPT knows that there will be no later arrivals to high-latency queues, whereas the corresponding ports of LWD will be idle; a similar phenomenon explains the non-optimality of LQD in the model with uniform processing. But in the contiguous case of the generalized model it is significantly harder for OPT to win much in this way since LWD will drop less packets from queues with the highest latencies (proportional to the required processing).

C. Upper Bound on the Competitiveness of LWD

Judging from the lower bounds presented in the previous section, LWD is the most promising policy among all we have considered. We now present the main result of this work: a constant upper bound on its competitive ratio. Note that although this bound is equal to the upper bound on LQD in the homogeneous processing model [2], the mapping routine is actually very different.

Theorem 7. *LWD is at most 2-competitive.*

Proof. The *latency* $\mathrm{lat}_t^{ALG}(p)$ of a packet p in a buffer is the number of time slots needed before p is transmitted (assuming it will not be pushed out). We define the latency of an already transmitted packet as -1 and the latency of a packet that has not yet arrived as ∞. A port of ALG's buffer is called *active* at time slot t if it transmits during t; otherwise, it is called *idle*. Since OPT is an offline optimal algorithm, we can assume that it never pushes out packets. To prove that LWD is 2-competitive, we will map each packet transmitted by OPT to a packet transmitted by LWD in such a way that at most two OPT packets correspond to each LWD packet. A packet p in OPT buffer mapped to an already transmitted LWD packet is called *ineligible* for mapping; otherwise p is called *eligible*.

To avoid ambiguity during the arrival phase, a reference time t should be interpreted as the arrival of a single packet. If several packets arrive at the same time slot, we consider them independently, in the sequence in which they arrive. If several output ports are active during the same transmission phase, t should be interpreted as processing and (if needed) transmission by a single output port in the following order: first, output ports with non-empty queues in LWD buffer, from 1 to n; then, remaining output ports with non-empty queues in OPT buffer, from 1 to n. A well-defined processing order of output ports during the transmission phase is necessary to process eligible OPT packets when mapping changes. The mapping routine is shown on Fig. 3, and Fig. 4 contains illustrative examples for each of the steps.

Lemma 8. *Let p be an eligible packet at position i (not counting ineligible packets) of Q_j^{OPT} at time t. If Q_j^{LWD} contains a packet q at the same position at time t, p is mapped to q by step A0, and $\mathrm{lat}_t^{\mathrm{OPT}}(p) \geq \mathrm{lat}_t^{\mathrm{LWD}}(q)$. Otherwise, p is mapped to an LWD packet q' by step A1, and $\mathrm{lat}_t^{\mathrm{OPT}}(p) \geq \mathrm{lat}_t^{\mathrm{LWD}}(q')$. Moreover, at most one OPT packet is mapped to an LWD packet by each of the steps A0 and A1, and all OPT packets are mapped.*

Proof. We prove the lemma by induction on the number of mapping changes. Since all ports process packets at the same rate (one processing cycle per time slot for each head-of-line packet), the mapping may change only when algorithms accept or transmit packets. For the base case, consider the first packet p accepted by OPT, say to Q_j^{OPT}. If $|Q_j^{\mathrm{LWD}}| > 0$, there is a head-of-line packet $q \in Q_j^{\mathrm{LWD}}$ (maybe $q = p$) that could be accepted before p, and therefore may be partially processed, so p is mapped to q by step A0 and the lemma holds. If $|Q_j^{\mathrm{LWD}}| = 0$, p is not accepted by LWD, so LWD's buffer is full, and LWD's buffer must contain a packet q that satisfies the latency constraint; thus, we map p to q by step A1, and the lemma holds again.

Assume by induction that the lemma holds for any $t' < t$. We are to show that it holds after the t^{th} mapping change. Let t^- be the time just before arrival or processing at timeslot t. Since packets cannot be accepted or transmitted during $(t-1, t^-]$, the induction hypothesis holds at time t^-. The following two cases are possible for a packet transmitted at time t by OPT or LWD from Q_j.

(1) Neither Q_j^{LWD} nor Q_j^{OPT} is empty at time t^-. Let p be the first eligible packet in Q_j^{OPT} (if it exists) and q be the first packet in Q_j^{LWD}. If p does not exist then during t the latency of a packet in Q_j^{LWD} may only decrease, and the induction hypothesis holds at the end of t. Otherwise, since the induction hypothesis holds at time t^-, p is mapped to q

Consider the time t^* after transmission of a head-of-line packet p from Q_j by OPT or LWD, just before the mapping is updated.

- **T0:** If LWD transmits p from Q_j then its image in the OPT buffer becomes ineligible (we are to show that OPT never transmits eligible for mapping packet from Q_j if LWD does not transmit from Q_j at time t).

Consider the time t^* after acceptance of an i-packet p by OPT or LWD, just before the mapping is updated (if necessary).

- **A0 (same queue):** if p is accepted by OPT to the l^{th} position (all ineligible packets are not accounted) of Q_i^{OPT} and there is a packet q (can be p) at the same position of Q_i^{LWD}, map p to q.
- **A1 (other queue):** if a packet p accepted by OPT is not mapped by A0, find any other packet q in LWD's memory that has no assignment of any OPT packet by step A1 and $\mathrm{lat}_{t^*}^{\mathrm{OPT}}(p) \ge \mathrm{lat}_{t^*}^{\mathrm{LWD}}(q)$; map p to q.
- **A2 (push out):** if LWD's packet p' of Q_j is pushed out by p, clear all mappings to p' and for each OPT packet q that was assigned to p' (at most one by step A0 and at most one by step A1) find a packet p'' in LWD's buffer that has no assignment by step A1 and map q to p''.
- **A3 (release A1):** if p is accepted by LWD and p is mapped to an OPT packet q by step A0 at time t and q was previously mapped by step A1, clear A1 mapping from q and its LWD pre-image.

Fig. 3: Mapping Routine for LWD policy.

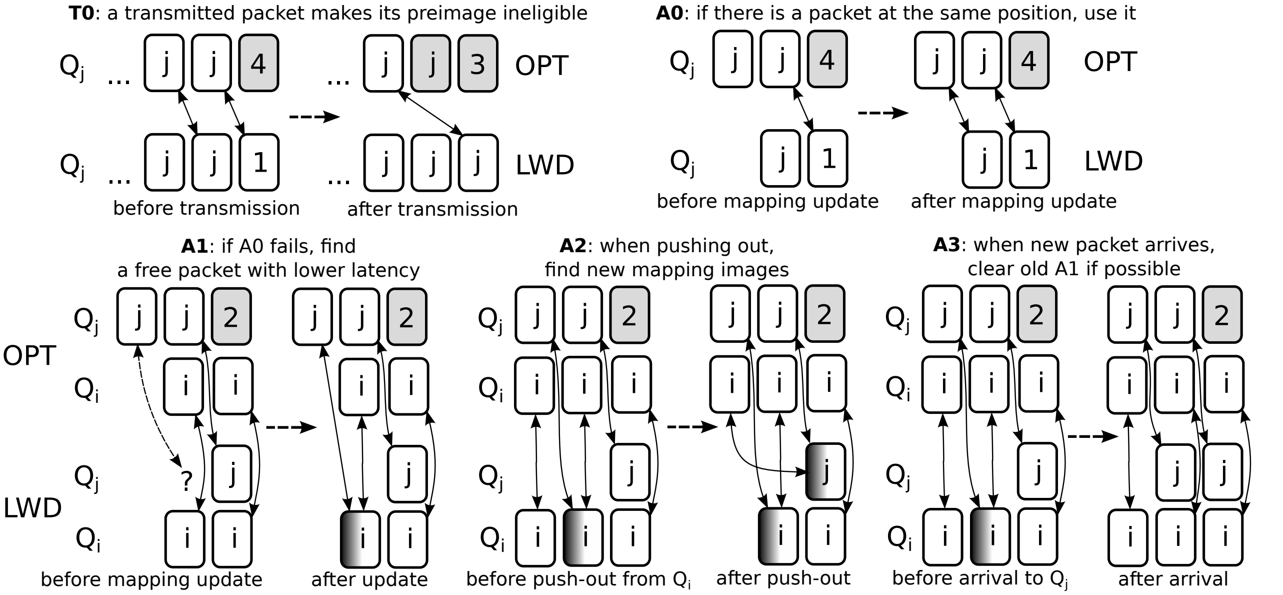


Fig. 4: The mapping routine. Shaded rectangles denote ineligible packets; rectangles with gradient fill, packets with two preimages (A0 and A1).

at time t^- by step A0 and $\mathrm{lat}_{t^-}^{\mathrm{OPT}}(p) \ge \mathrm{lat}_{t^-}^{\mathrm{LWD}}(q)$, so either both algorithms or LWD only may transmit at time t. In both cases, mapping for eligible packets does not change at the end of t. Since output ports are served in the same order, and each non-empty output port reduces required processing by a single cycle, after the t^{th} mapping change the latency constraint holds for all packets mapped with A0 and A1. Moreover, since there were no new arrivals all OPT packets are mapped at the end of t.

(2) OPT transmits an eligible packet p from Q_j^{OPT} but Q_j^{LWD} is empty at time t^-. Since at time t^- all OPT packets are mapped and Q_j^{LWD} is empty, it must be the case that p is mapped to an LWD packet q by step A1. Since output ports are processed in the same order, p cannot be eligible at time t^-, so this case is impossible.

The following two cases are possible when a packet is accepted at time t by OPT or LWD to Q_j.

(3) A packet is accepted at time t by OPT to the l^{th} position (ineligible packets are not counted) of Q_j^{OPT}, and there are at most $|Q_j^{\mathrm{LWD}}|$ eligible packets in Q_j^{OPT}. So the l^{th} LWD packet q in Q_j^{LWD} is available for mapping by step A0. Moreover, since the latency constraint holds for each of the first $i-1$ eligible OPT packets and all packets admitted to Q_j require the same processing, $\mathrm{lat}_{t}^{\mathrm{OPT}}(p) \ge \mathrm{lat}_{t}^{\mathrm{LWD}}(q)$. If p is accepted by LWD and pushes out another packet p' from Q_n^{LWD} then we need to find LWD's packets available for mapping by step A1 for the pre-image of p' in OPT buffer. Since the induction hypothesis holds at time t^-, at most one packet is mapped to p' by each of the steps A0 and A1. Also, since p' is pushed out by LWD, its latency is highest in LWD's buffer, so any LWD packet (including p) that has no assignment by step A1 is available for mapping. Since LWD pushes out at time t, LWD's buffer was full at time t^-, so (since buffer sizes are equal) there are enough packets available to map the pre-image of p' by step A1.

(4) A packet q is accepted at time t by LWD to the l^{th} position, and there are more than $|Q_j^{\mathrm{LWD}}|$ eligible packets in Q_j^{OPT}. Since latency constraints hold for the first $l-1$ eligible OPT packets mapped by step A0 and only j-packets are admitted to Q_j, the latency constraint holds for the l^{th} eligible OPT packet and the l^{th} LWD packet in Q_j. By step A3, the A1 mapping is cleared from the image of the l^{th} eligible OPT packet. Therefore, if a packet q is accepted by OPT and even if LWD's buffer is not full q can be mapped to p'' by step

A1. If q is accepted by LWD and pushes out another packet p', LWD's buffer is already full at time t^-; moreover, p' has the highest latency in LWD's buffer at time t^-. Similar to (3), LWD has enough packets to map the image of p' by step A1.

Thus, the induction hypothesis holds at the end of t. □

By Lemma 8, each packet transmitted by LWD is the image of at most two OPT packets (one by step A0 and another by step A1), and at any time all OPT packets are mapped. Theorem 7 immediately follows. □

IV. Packet Scheduling with Heterogeneous Values

We now proceed to the packet scheduling problem with heterogeneous values in a shared memory switch architecture. In this model, each unit-sized packet with required work 1 has two parameters: an output port label and an intrinsic value; the objective is to maximize the total transmitted value. It has been formulated as an open problem in SIGACT News [19, p. 22]: "What happens in the shared memory, multiple output queue model with general-valued packets? Is constant competitiveness achievable?" In this section we provide the first results in this direction. Note that here we consider a more general model and allow packets with heterogeneous values to sojourn in the same queue. As result, in all policies below we use the most favourable processing order in each queue corresponding to an output port, assuming they are priority queues (PQ) where most valuable packets are processed first. This can only be an improvement over FIFO processing or any other order since priority queue is optimal in the case of a single queue.

A. Model Description

We consider an $l \times n$ switch with shared memory of size B, with l input ports and n output ports. Each output port i has one corresponding output queue Q_i; we assume that $B \geq n$. Packets arrive at input ports, each labeled with a destination output port and an intrinsic value of at most k, $B \geq k \geq 1$. Note that unlike the previous model, each packet requires only a single processing cycle. In what follows, we denote by $\boxed{v}$ a packet with value v; by $x \times \boxed{v}$, a burst of x packets with value v each. All admitted packets in an output queue are ordered in non-increasing order of values; the number of packets in Q_i is denoted by $|Q_i|$. Time is slotted; we divide each time slot into two phases (see Fig. 5). During the (1) *arrival phase* a burst of new packets arrives at each input port, and the *buffer management policy* decides which packets to admit; we assume that during the arrival phase input ports are served in a fixed order from input port 1 to input port n, with no restriction on the burst size. During the (2) *transmission phase*, the head-of-line packet in each non-empty output queue is transmitted. An accepted packet can again later be dropped from the buffer when another packet is accepted instead (push-out policy). Note that for unit values this reduces to the model of [2], so the $4/3$ lower bound on any online policy shown there holds in our case too.

B. Choosing the Right Properties for Buffer Management

Even if processing order is fixed, it is unclear which traffic should be admitted; there may be a tradeoff between traffic that opens new active ports and traffic that increases the total admitted value. We do not consider non-push-out policies since it is straightforward that a greedy non-push-out policy that accepts a packet if there is available space is at least k-competitive (fill the buffer with $\boxed{1}$s, then send in the $\boxed{k}$s). We define several push-out scheduling policies that greedily optimize the number of active ports, total admitted value, or a combination of both. The algorithms are illustrated on Fig. 5.

First, we consider the Longest-Queue-Drop (LQD) heuristic again; in this model, LQD drops the last (lowest value) packet from the longest queue when a buffer is congested, balancing the queue sizes.

It is illustrated on the left of Fig. 5. Here and in all other examples, $B = 8$, and the queues start from 0 packets in Q_1, one $\boxed{1}$ in Q_2, $\boxed{1}$, $\boxed{2}$, and $\boxed{3}$ in Q_3, and $\boxed{1}$ and $\boxed{4}$ in Q_4; on this sample step, there arrive one $\boxed{3}$ to queue 1, one $\boxed{1}$ to queue 2, and $2 \times \boxed{3}$ to queue 3. In this example, we need to drop two packets, and queue 3 is by far the longest, so LQD drops the two lowest value packets from queue 3 (like all other queues, it processes arrving packets in PQ order). Then we process four highest-value packets, one from each queue.

However, in the value-based model LQD again fails to reach constant competitiveness. Note that constructions in Theorems 9, 10, and 11 below all fall into an interesting special case of the general model, when a packet's value is uniquely defined by its output port label. This special case makes practical sense: often a different subset of cores is assigned to process specific types of traffic, and these types can be defined either by required processing, as in Section III, or by value, as in this special case. Obviously, the same bounds apply in the general case as well.

Theorem 9. *The competitive ratio of LQD is at least* $\left(\sqrt[3]{k} - o\left(\sqrt[3]{k}\right)\right)$.

Proof. In this construction, a packet's value is equal to its output port label. Fix a, $1 \leq a \leq k$. On the first time slot, there arrive B packets of every value from 1 to a and B more packets of value k. LQD balances the queues, leaving $\frac{B}{a}$ in each of them (throughout the paper, we assume that B divides everything we need it to divide). At the same time, OPT takes in B packets of value k. Then, on each time slot packets with values from 1 to a arrive but packets of value k do not. Thus, over the next B time slots OPT transmits $B(a+1)$ packets with total value $B(\frac{1}{2}a(a-1)+k)$ while LQD transmits $B(a+\frac{1}{a})$ packets with total value $B(\frac{1}{2}a(a-1)+k/a)$. Then the initial burst arrives again, and the construction repeats itself, getting a competitive ratio of $\frac{\frac{1}{2}a(a-1)+k}{\frac{1}{2}a(a-1)+k/a}$. This expression is maximized for $a \approx \sqrt[3]{k}$, with the resulting ratio of $\sqrt[3]{k} - o(\sqrt[3]{k})$. □

The next policy greedily maximizes total admitted value by pushing out packets with minimal current value.

Minimal-Value-Drop (MVD): during the arrival of an $\boxed{m}$-packet p that is destined to the i-th port, denote by Q_j the nonempty queue that contains a packet with minimal value (if

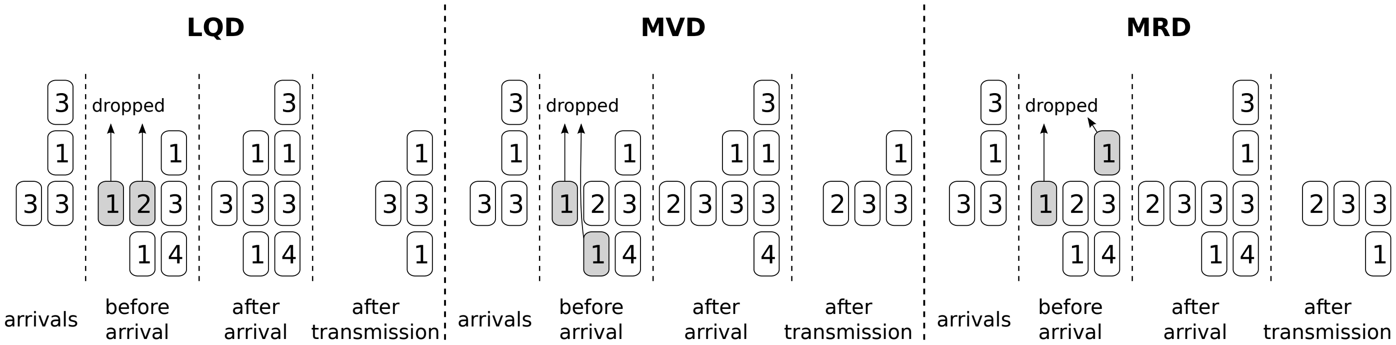


Fig. 5: A sample time slot of Longest-Queue-Drop (LQD), Minimal-Value-Drop (MVD), and Maximal-Ratio-Drop (MRD) with maximal value $k = 4$, 4 output ports, and a shared buffer of size $B = 8$.

there are several such queues, choose the longest among them); then do the following: (1) if the buffer is not full, accept p into Q_i; (2) if the buffer is full and the minimal admitted value is less than $\boxed{m}$, push out last packet from Q_j and accept p into Q_i; (3) if the buffer is full and the minimal admitted value is at most $\boxed{m}$, drop p.

MVD is the middle policy shown on Fig. 5. Again, $B = 8$, the queues start from nothing in Q_1, one $\boxed{1}$ in Q_2, $\boxed{1}$, $\boxed{2}$, and $\boxed{3}$ in Q_3, and $\boxed{1}$ and $\boxed{4}$ in Q_4; there arrive one $\boxed{3}$ to Q_1, one $\boxed{1}$ to Q_2, and $2 \times \boxed{3}$ to Q_3. Again, we need to drop two packets, and the MVD algorithm drops two packets with lowest values, in this case two $\boxed{1}$s out of three; to break ties, it makes sense to drop packets from the longest queues, so we drop $\boxed{1}$s from Q_3 and Q_4. Then we process four highest-value packets, one from each queue.

The MVD algorithm prioritizes packets of maximal value. Unfortunately, MVD has a linear lower bound on competitiveness in the worst case.

Theorem 10. MVD *is at least* $\frac{m-1}{2}$*-competitive for* $m = \min\{k, B\}$.

Proof. Again, each packet's value equals its output port label. On each time slot, there arrive B packets of every value from 1 to m. As a result, MVD only processes packets of value m while the optimal algorithm processes each value from 1 to m, getting a competitive ratio of $\frac{m(m-1)/2}{m}$. □

Note that a corresponding policy to MVD in the model with heterogeneous processing requirements is BPD that minimizes a total required processing. But in difference from MVD, BPD can achieve a relatively good competitive ratio that demonstrate a significant difference between required processing requirements and packet's values characteristic.

As a policy that we conjecture may reach constant competitive ratio, we propose a combination of these two characteristics, an idea similar to the LWD policy in the previous model that we have shown to be 2-competitive in Section III-C. A natural idea would be to try to combine these two heuristics, prioritizing packets with high value only if they come from shorter queues. One implementation of such an idea is given in the MAXIMAL RATIO DROP algorithm.

Maximal-Ratio-Drop (MRD): during the arrival of an $\boxed{m}$-packet p that is destined to the i-th port, denote by Q_j the nonempty queue with a maximal value of $\frac{|Q_j|}{a_j}$, where a_j is an average value in Q_j (if there are several such queues, choose as Q_j a queue that contains a packet with a smaller value) then do the following:

(1) if the buffer is not full, accept p into Q_i;
(2) if the buffer is full and has admitted values smaller than $\boxed{m}$, push out last packet from Q_j and accept p to Q_i;
(3) if the buffer is full and minimal admitted value in the buffer is bigger than $\boxed{m}$, drop p.

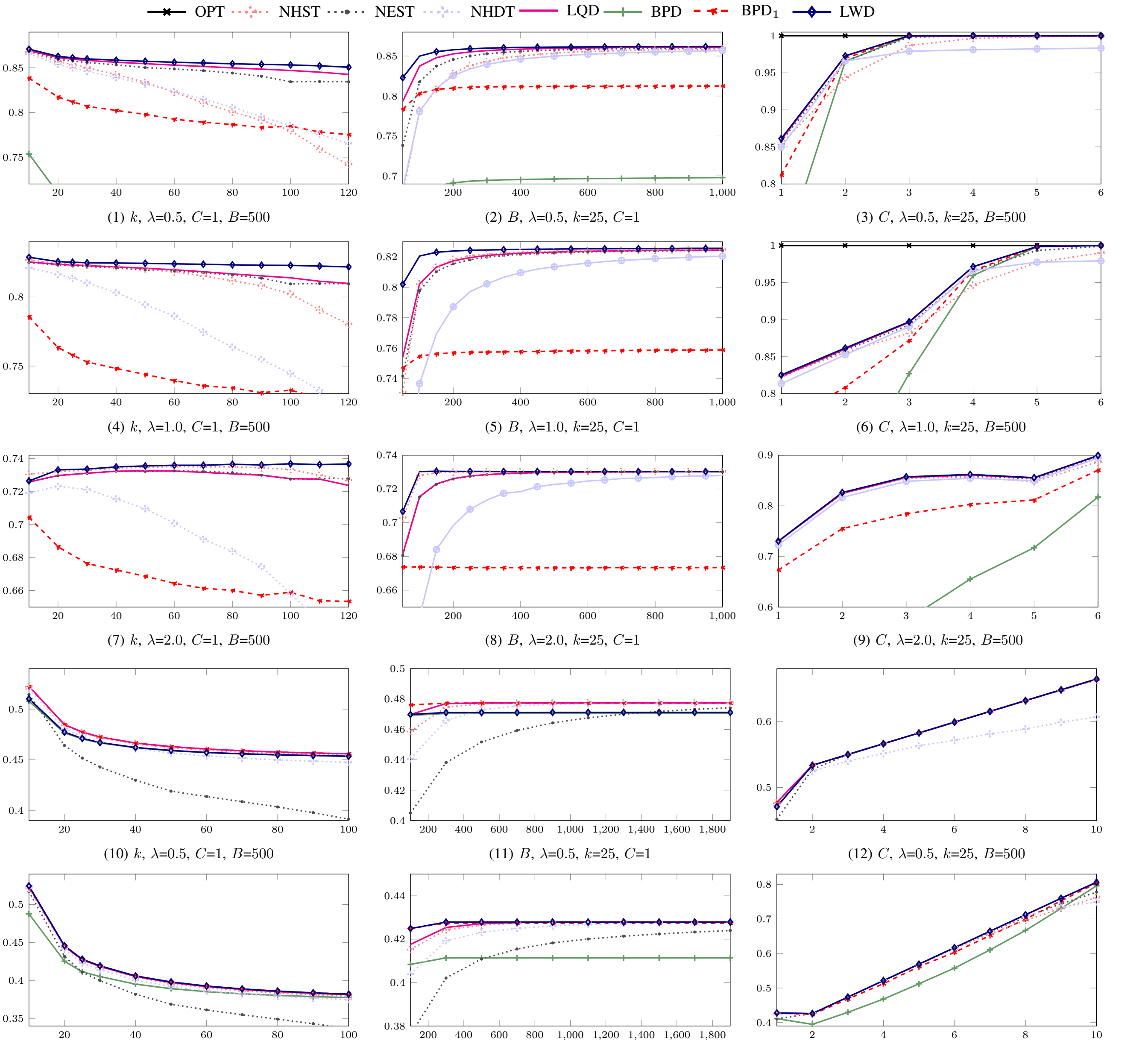
A sample MRD time slot shown on the right of Fig. 5, Again, $B = 8$, the queues start from nothing in Q_1, one $\boxed{1}$ in Q_2, $\boxed{1}$, $\boxed{2}$, and $\boxed{3}$ in Q_3, and $\boxed{1}$ and $\boxed{4}$ in Q_4; there arrive one $\boxed{3}$ to Q_1, one $\boxed{1}$ to Q_2, and $2 \times \boxed{3}$ to Q_3. To choose the two extra packets to drop, MRD computes the ratio of queue size to average packet value for queues with new packets added to them: $r_1 = \frac{1}{3}$, $r_2 = \frac{2}{1} = 2$, $r_3 = \frac{5}{12/5} = \frac{25}{12}$, $r_4 = \frac{2}{5/2} = \frac{4}{5}$. The maximal ratio is r_3, so we drop the lowest valued packet $\boxed{1}$ from Q_3. Then r_3 changes to $r_3 = \frac{4}{11/4} = \frac{16}{11}$, Q_2 becomes the maximal ratio queue, and we drop $\boxed{1}$ from there. Then MRD processes four highest-value packets, one from each queue.

Note that MRD emulates LQD in case all packets have unit values, and the lower bound $\sqrt{2}$ shown in [2] applies. We can also show a constant lower bound for MRD in the special case when each packet's value is equal to its output port label.

Theorem 11. MRD *is at least* $\frac{4}{3}$*-competitive in case each packet's value is equal to its output port label.*

Proof. Consider the first burst with B packets of value 1, 2, 3, and 6 each. Balancing the size-value ratio, MRD will accept $\frac{B}{2} \times \boxed{6}$, $\frac{B}{4} \times \boxed{3}$, $\frac{B}{6} \times \boxed{2}$, and $\frac{B}{12} \times \boxed{1}$, while OPT accepts $(B-3) \times \boxed{6}$ and one packet of each other value. Then packets of value 1, 2, and 3 keep coming so that OPT always has something to do but packets of value 6 do not arrive any more. Thus, in $B-3$ steps OPT will have transmitted for a total value $12(B-3)$, while MRD will have total value $12\frac{B}{2} + 6(\frac{B}{2} - 3) = 9B - 18$, getting the bound. □

It remains an interesting open problem to show whether MRD has a constant competitive ratio in the worst case.



(1) k, λ=0.5, C=1, B=500

(2) B, λ=0.5, k=25, C=1

(3) C, λ=0.5, k=25, B=500

(4) k, λ=1.0, C=1, B=500

(5) B, λ=1.0, k=25, C=1

(6) C, λ=1.0, k=25, B=500

(7) k, λ=2.0, C=1, B=500

(8) B, λ=2.0, k=25, C=1

(9) C, λ=2.0, k=25, B=500

(10) k, λ=0.5, C=1, B=500

(11) B, λ=0.5, k=25, C=1

(12) C, λ=0.5, k=25, B=500

(13) k, λ=0.5, C=1, B=500

(14) B, λ=0.5, k=25, C=1

(15) C, λ=0.5, k=25, B=500

Fig. 6: Simulation results for the heterogeneous required processing model. Graphs show the competitive ratio as a function of: left column – k for $B = 500$, $C = 1$; middle column – B for $k = 25$, $C = 1$; right column – C for $k = 25$, $B = 500$; rows have different burst intensity λ: first, fourth, and fifth row – $\lambda = 0.5$, second row – $\lambda = 1.0$, third row – $\lambda = 2.0$. The fourth row shows the results with two possible processing values; the fifth row, with three.

V. Simulations

In this section, we present a comprehensive simulation study that complements our theoretical results. While we have shown that, e.g., LWD is provably better in the worst case than other policies, this result would have much less relevance if in practice LWD performed significantly worse on average. Hence, we have experimented with the proposed policies to study their performance in a practical context.

A. Experimental Setting

To see how the proposed policies compare in actual applications, we have conducted an extensive simulation study. It would be valuable to test the proposed policies on real life traces, but, unfortunately, available datasets such as CAIDA [18], first, do not contain traces with heterogeneous processing requirements, and second, do not contain enough information about the hardware and network processor configuration to determine the time scale. Naturally, our model and policies crucially depend on how long a timeslot should last: it determines how much congestion there will be in the buffer. Thus, we have conducted three series of experiments on synthetic traces, studying how the competitive ratio depends on maximal packet size (i.e., the number of queues) k, the buffer size B, and the number of processing cores for each queue C (performance speedup).

Note that the actual optimal algorithm in our model has prohibitive computational cost. Thus, to provide a gold standard against which to compare our policies we have used an algorithm which is actually better than optimal: a single priority queue that processes smallest packets first (resp., packets with largest value) and has kC cores. This algorithm has been proven optimal in the single queue model, so in our model, where some queues may be congested and others idle, it performs even better than optimal.

As for traffic generation, it is well known that network traffic has a long-tail distribution due to its bursty nature and long-range dependencies exhibited by network traffic [41] that, unlike many other kinds of data streams, does not conform to Poisson models [38]. Many mathematical models have been put forward to simulate long-tail traffic, including Markov-modulated Poisson processes [17], [20] and fractional Brownian motion [37]. In this work, we are using the probably most successful traffic model to date, the Poisson Pareto burst process (PPBP) [1], [42]. In PPBP, the traffic is modeled with multiple overlapping bursts whose lengths conform to a Pareto (long-tail) distribution. We use PPBP to model the stream of incoming packets.

As for required processing, output ports, and values, lacking real life data on these secondary parameters we have decided to test the following plausible variations: (1) uniform distribution used in most of our experiments; (2) distribution of required processing where only values 1 and k are possible (row 4 on Fig. 6); (3) distribution where only values 1, $k/2$, and k are possible (row 5 on Fig. 6).

B. Heterogeneous Processing Experiments

Figure 6 shows simulation results presented in terms of competitive ratio: each graph shows the competitive ratio (ratio of packets processed) w.r.t. to the "better than optimal" reference algorithm (in black; always equals 1.0). The figure shows the results of experiments with unit packet value and variable processing requirements.

Across all simulations, it is clear that LWD is the best algorithm on average, with LQD a close second. BPD in this setting is abysmally bad, clearly the worst of all algorithms; this is due to the fact that BPD pushes out large packets independently of queue size, so it consistently underuses available cores. Since it does not make sense to push out the last packet remaining in a queue, thus artificially reducing the number of active ports, we also introduce the BPD$_1$ algorithm that works similar to BPD but does not push out the last packet in a queue; BPD$_1$ does much better than BPD but still loses to other algorithms.

In the first set of simulations (the first column of Fig. 6, graphs (1), (4), (7), (10), (13)), we study competitive ratio as a function of the maximal packet size k. Naturally, the competitive ratio of all algorithms drops as k grows (the better-than-optimal OPT is by default better at using available resources), but non-preemptive algorithms clearly deteriorate faster.

In the second set (second column of Fig. 6, graphs (2), (5), (8), (11), (14)), we study competitive ratio as a function of the buffer size B, covering various values from small to very large buffer sizes; in this set of simulations, we attempted to cover the transition from small buffers with large dropout rates into a situation with nearly no congestion. Here, all algorithms improve similar to each other, and the only algorithm that fails to take advantage of a larger buffer and starts losing to non-preemtive algorithms is BPD (including BPD$_1$). Other preemptive algorithms do better than non-preemptive ones, and LWD remains the best option throughout.

In the third set of experiments, we look at competitive ratio as a function of the speedup C imposed on each core. We see that preemptive algorithms can make better use of this advantage and catch up with the optimal algorithm faster than non-preemptive ones. Again, LWD is the best algorithm in this case.

Experiments with alternative distributions of required processing (graphs (10)-(15)) show virtually the same picture as experiments with the uniform distribution; keep in mind that the same values of k now correspond to much more congested traffic since there are no intermediate values of required processing.

C. Heterogeneous Value Experiments

Figure 7 shows the results of experiments in the model with variable packet values and unit required processing. Note that in the case of unit processing a much larger incoming intensity and smaller buffers are required to achieve the same kind of congestion, so intensities in these experiments are increased by a factor of 10 (we use $\lambda = 5.0$, 10.0, and 20.0) while the default value of buffer size is now $B = 20$. Again, we ran three series of experiments, varying n, B, and C for different intensities λ.

The first set of experiments (left column on Fig. 7, graphs 1, 4, and 7) contains experiments with varying number of output ports n. An interesting pattern emerges: as n increases, the competitive ratios of MRD and MVD first deteriorate but then begin to improve. This is due to the following effect: for high λ and small n, it is advantageous for OPT to have larger n since our overly optimistic version of OPT does not care about output ports; but then (especially for $n > B$) it actually becomes easier for other algorithms as the probability that they can just put in one packet per output port and send them all out immediately increases. LQD here lags behind MRD and MVD because it does not distinguish between packets in a specific queue, and this is exactly what is needed when there is room for one packet per queue only. MRD is the clear winner in most settings.

The second set (middle column, graphs (2), (5), and (8) on Fig. 7) varies the buffer size B. Again, first OPT makes better use of increased buffer size but then larger buffers basically begin to equate the differences between algorithms. Here again MRD wins over both LQD and MVD.

Finally, the third set (right column, graphs (3), (6), and (9) on Fig. 7) increases C, the number of processing cores at

Algorithm	Lower bound	Upper bound
Heterogeneous processing		
NHST	$kZ + o(kZ)$	$kZ + o(kZ)$
NEST	$n + o(n)$	$n + o(n)$
NHDT	$\frac{1}{2}\sqrt{k \ln k} - o(\sqrt{k \ln k})$	-
LQD	$\sqrt{k} - o(\sqrt{k}))$	-
BPD	$\ln k + \gamma$	-
LWD	$4/3 - 6/B$	2
Heterogeneous values		
LQD	$\sqrt[3]{k} - o\left(\sqrt[3]{k}\right)$	-
MVD	$(m-1)/2$	-
MRD	$4/3$	-

TABLE II: Results summary: lower and upper bounds.

work. For the case of heterogeneous values and unit processing, it is obvious that very soon we achieve the situation where all packets residing in the buffer are processed at every time slot, and the remaining differences between algorithms are constant and are entirely due to their admission policies (as they still cannot accept more than B packets at a time). In this specific setting, MVD is naturally the best algorithm because it optimizes precisely what is needed to optimize in this case; note that it keeps improving for longer than MRD and LQD due to longer individual queues allowed by MVD.

Generally speaking, we can conclude that in most situations of congestion MRD is the better policy of the three. MVD, however, wins in a special case of both very small buffers and high computational power, when all or almost all of the packets in the buffer can be consistently processed on every time slot but the buffer does not have room to store all incoming packets. In this case, output ports are no longer relevant and the problem basically degenerates to a priority queue, so MVD is clearly the right choice.

VI. Conclusion

Cloud applications bring new challenges to the design of network elements, in particular how to accommodate the burstiness of traffic workloads. Over the recent years, there has been a growing interest in understanding the impact of buffer architecture on network performance. Since shared memory switches represent the best candidate architecture to exploit buffer capacity, in this paper we analyze the performance of this architecture. Our goal is to explore the impact of additional traffic characteristics such as varying processing requirements.

In this work, we study the tradeoffs inevitable on the path to a "perfect" policy in a shared memory switch, both analytically and with simulations. Recent research advocates the use of smaller buffers in routers, aiming to reduce queueing delay in the presence of (mostly) TCP traffic; however, it sidesteps the issue that as buffers get smaller, the effect of processing delay becomes much more pronounced. The majority of currently deployed admission control policies do not take into account (at least explicitly) the importance of heterogeneous packet processing. Our theoretical results are summarized in Table II. In the first part of this work, we study the impact of heterogeneous processing on throughput in the shared memory switch architecture. We demonstrate that policies such as LQD or NHDT — very attractive under uniform processing requirements — perform poorly in the worst case. We believe that this observation will provide new insights to the practice of admission control policies.

In addition, we have considered an open problem that optimizes transmission of unit-sized packets with heterogeneous values in a shared memory switch architecture. We provide the first results in this direction and define the most promising directions for further studies. We have also conducted a comprehensive simulation study that validates our intuition obtained in the worst case. In particular, LQD indeed does turn out to be worse than the policies we have proposed for this setting, namely LWD in case of heterogeneous processing requirements and MRD in case of heterogeneous packet values.

References

[1] Ronald G. Addie, Timothy D. Neame, and Moshe Zukerman. Performance evaluation of a queue fed by a poisson pareto burst process. *Computer Networks*, 40(3):377 – 397, 2002.

[2] William Aiello, Alexander Kesselman, and Yishay Mansour. Competitive buffer management for shared-memory switches. *ACM Transactions on Algorithms*, 5(1), 2008.

[3] William Aiello, Yishay Mansour, S. Rajagopolan, and Adi Rosén. Competitive queue policies for differentiated services. *J. Algorithms*, 55(2):113–141, 2005.

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

[5] Yossi Azar and Oren Gilon. Buffer management for packets with processing times. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 47–58, 2015.

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

[7] Yossi Azar and Yossi Richter. An improved algorithm for CIOQ switches. *ACM Transactions on algorithms*, 2(2):282–295, 2006.

[8] BBC News. US Watchdog to Propose New Net Neutrality Rules, 2014. http://www.bbc.com/news/technology-27141121.

[9] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *MCC*, pages 13–16, 2012.

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

[11] Cavium. OCTEON II CN68XX multi-core MIPS64 processors, product brief, 2010. http://www.caviumnetworks.com/OCTEON-II_CN68XX.html.

[12] Pavel Chuprikov, Sergey I. Nikolenko, and Kirill Kogan. Priority queueing with multiple packet characteristics. In *INFOCOM*, pages 1–9, 2015.

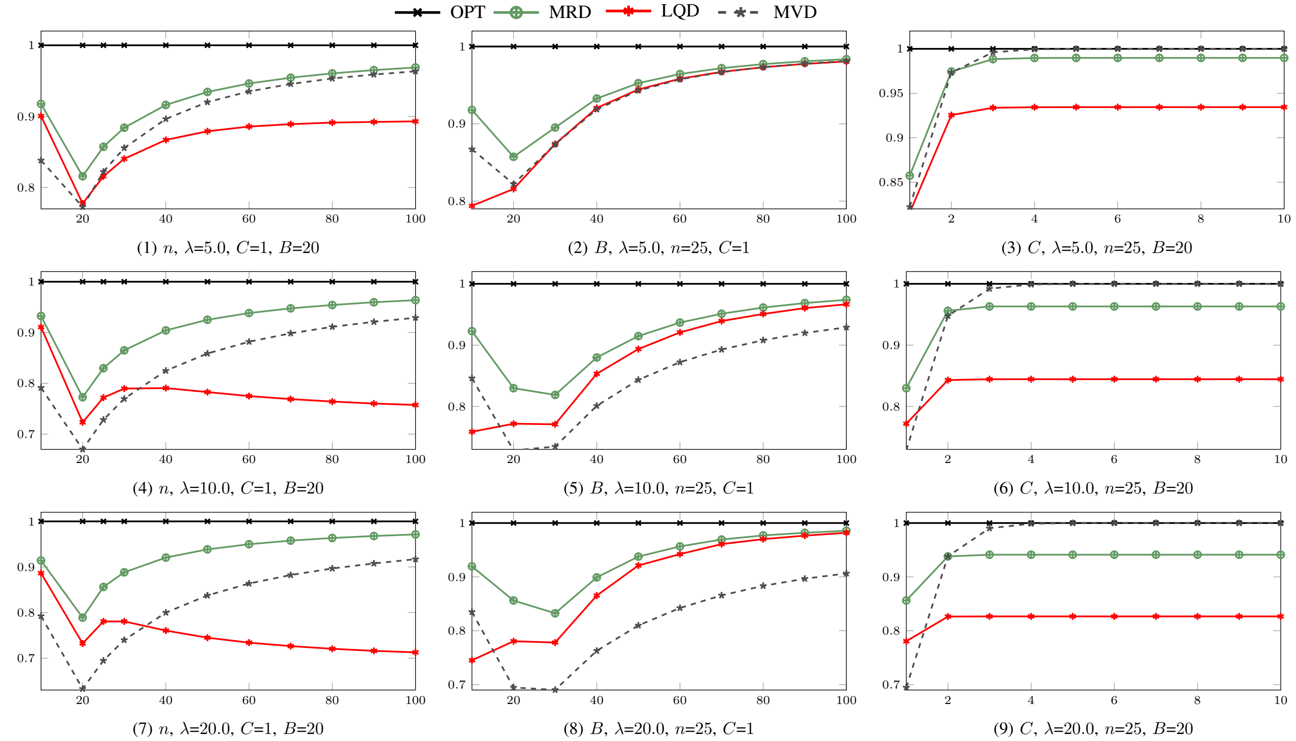
[13] Cisco. The Cisco QuantumFlow processor, product brief, 2010. [Online] http://www.cisco.com/en/US/prod/collateral/routers/ps9343/solution_overview_c22-448936.html.

[14] Sujal Das and Rochan Sankar. Broadcom smart-buffer technology in data center switches for cost-effective performance scaling of cloud applications, 2012. https://www.broadcom.com/collateral/etp/SBT-ETP100.pdf.

[15] Matthias Englert and Matthias Westermann. Lower and upper bounds on FIFO buffer management in QoS switches. *Algorithmica*, 53(4):523–548, 2009.

[16] Patrick Th. Eugster, Kirill Kogan, Sergey I. Nikolenko, and Alexander Sirotkin. Shared memory buffer management for heterogeneous packet processing. In *IEEE 34th International Conference on Distributed Computing Systems, ICDCS 2014, Madrid, Spain, June 30 - July 3, 2014*, pages 471–480, 2014.

[17] Wolfgang Fischer and Kathleen Meier-Hellstern. The markov-modulated poisson process (mmpp) cookbook. *Performance evaluation*, 18(2):149–171, 1993.



(1) n, λ=5.0, C=1, B=20

(2) B, λ=5.0, n=25, C=1

(3) C, λ=5.0, n=25, B=20

(4) n, λ=10.0, C=1, B=20

(5) B, λ=10.0, n=25, C=1

(6) C, λ=10.0, n=25, B=20

(7) n, λ=20.0, C=1, B=20

(8) B, λ=20.0, n=25, C=1

(9) C, λ=20.0, n=25, B=20

Fig. 7: Simulation results for the heterogeneous value model. Graphs show the competitive ratio as a function of: left column – k for $B = 20$, $C = 1$; middle column – B for $n = 25$, $C = 1$; right column – C for $n = 25$, $B = 20$; rows have different burst intensity λ: top row – $\lambda = 5.0$, middle row – $\lambda = 10.0$, bottom row – $\lambda = 20.0$.

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

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

[20] H. Heffes and D. Lucantoni. A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance. *Selected Areas in Communications, IEEE Journal on*, 4(6):856–868, Sep 1986.

[21] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 15–26, 2013.

[22] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: experience with a globally-deployed software defined wan. In *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 3–14, 2013.

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

[24] Alex Kesselman, Boaz Patt-Shamir, and Gabriel Scalosub. Competitive buffer management with packet dependencies. In *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2009.

[25] Alexander Kesselman, Kirill Kogan, and Michael Segal. Packet mode and QoS algorithms for buffered crossbar switches with FIFO queuing. *Distributed Computing*, 23(3):163–175, 2010.

[26] Alexander Kesselman, Kirill Kogan, and Michael Segal. Improved competitive performance bounds for cioq switches. *Algorithmica*, 63(1-2):411–424, 2012.

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

[28] Alexander Kesselman and Yishay Mansour. Harmonic buffer management policy for shared memory switches. *Theor. Comput. Sci.*, 324(2-3):161–182, 2004.

[29] Kirill Kogan, Alejandro López-Ortiz, Sergey I. Nikolenko, and Alexander Sirotkin. A taxonomy of semi-fifo policies. In *IPCCC*, pages 295–304, 2012.

[30] Kirill Kogan, Alejandro López-Ortiz, Sergey I. Nikolenko, and Alexander Sirotkin. Multi-queued network processors for packets with heterogeneous processing requirements. In *COMSNETS*, pages 1–10, 2013.

[31] Kirill Kogan, Alejandro López-Ortiz, Sergey I. Nikolenko, Alexander Sirotkin, and Denis Tugaryov. Fifo queueing policies for packets with heterogeneous processing. In *MedAlg*, pages 248–260, 2012.

[32] Kirill Kogan, Sergey Nikolenko, Alejandro López-Ortiz, Gabriel Scalosub, and Michael Segal. Balancing work and size with bounded buffers. In *COMSNETS*, pages 1–8, 2014.

[33] Yishay Mansour, Boaz Patt-Shamir, and Ofer Lapid. Optimal smoothing schedules for real-time streams. *Distributed Computing*, 17(1):77–89, 2004.

[34] Yishay Mansour, Boaz Patt-Shamir, and Dror Rawitz. Overflow management with multipart packets. In *INFOCOM*, pages 2606–2614, 2011.

[35] Nick McKeown, Guru Parulkar, Scott Shenker, Tom Anderson, Larry Peterson, Jonathan Turner, Hari Balakrishnan, and Jennifer Rexford. OpenFlow switch specification, 2011. http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[36] Sergey I. Nikolenko and Kirill Kogan. Single and multiple buffer processing. In *Encyclopedia of Algorithms*. Springer, 2015.

[37] I. Norros. On the use of fractional brownian motion in the theory of connectionless networks. *Selected Areas in Communications, IEEE Journal on*, 13(6):953–962, Aug 1995.

[38] Vern Paxson and Sally Floyd. Wide area traffic: The failure of poisson modeling. *IEEE/ACM Trans. Netw.*, 3(3):226–244, June 1995.

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

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

[41] Xiaoyun Zhu, Jie Yu, and John Doyle. Heavy tails, generalized coding, and optimal web layout. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1617–1626 vol.3, 2001.

[42] M. Zukerman, T.D. Neame, and R.G. Addie. Internet traffic modeling and future technology implications. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 1, pages 587–596 vol.1, March 2003.