Large Profits or Fast Gains: A Dilemma in Maximizing Throughput with Applications to Network Processors

Kirill Kogan and Alejandro López-Ortiz
School of Computer Science
University of Waterloo
{kkogan,alopez-o}@uwaterloo.ca

Gabriel Scalosub and Michael Segal*
Dept. of Comm. Syst. Eng.
Ben-Gurion University of the Negev
{sgabriel,segal}@cse.bgu.ac.il

Abstract

We consider the fundamental problem of managing a bounded size queue buffer where traffic consists of packets of varying size, where each packet requires several rounds of processing before it can be transmitted from the queue buffer. The goal in such an environment is to maximize the overall size of packets that are successfully transmitted. This model is motivated by the ever-growing ubiquity of network processors architectures, which must deal with heterogeneously-sized traffic, with heterogeneous processing requirements. Our work addresses the tension between two conflicting algorithmic approaches in such settings: the tendency to favor packets with fewer processing requirements, thus leading to *fast* contributions to the accumulated throughput, as opposed to preferring packets of larger size, which imply a *large* increase in throughput at each step. We present a model for studying such systems, and present competitive algorithms whose performance depend on the maximum size a packet may have, and maximum amount of processing a packet may require. We further provide lower bounds on algorithms performance in such settings.

Keywords: Scheduling, Buffer Management, Priority Queueing, Switches, Online Algorithms, Competitive Analysis.

*The work by Michael Segal has been partly supported by France Telecom, General Motors, European project FLAVIA and Israeli Ministry of Industry, Trade and Labor (consortium CORNET).

1 Introduction

Modern day computing faces increasingly heterogeneous tasks, varying in, e.g., importance, size, and processing requirements. Such scenarios are encountered, for example, in OS caching environments, job-shop scheduling, and most notably, network processors dealing with packet-switched traffic. Dealing with such heterogeneity is usually done via prioritizing the tasks, both at the ingress of the queue where discard decisions are taken (if such decisions are allowed), as well as at the egress of the queue where decisions are being made as to which task should be scheduled for processing. The decisions made by the prioritization module affect various Quality-of-Service (QoS) metrics, such as delay, throughput, etc. Furthermore, in many of these environments the queue might be constrained to use a limited size buffer, which further restricts the ability to provide adequate performance to the underlying computing tasks. Such scenarios occur, e.g., when using a highly limited CPU cache in high-performance computing, in job-shop scheduling, and in network processor architectures which must perform multiple tasks on incoming packets.

In what follows, we adopt the terminology used to describe queue management within a router in a packet-switched network. We focus our attention on a general model for the problem where we are required to manage the admission control and scheduling modules of a single bounded size queue, where arriving traffic consists of *packets*, such that each packet has some *size* (e.g., in bytes), and a *processing requirement* (in processor cycles). A packet is successfully *transmitted* once the scheduling module has scheduled the packet for processing for at least its required number of cycles, while the packet resides in the buffer. If a packet is dropped from the buffer, either upon arrival due to admission control policies, or after being admitted and possibly partially, but not fully, processed (e.g., in scenarios where push-out is allowed), then the packet is irrevocably lost. We focus our attention on maximizing the *throughput* of the queue, measured by the overall number of bytes of packets that are successfully transmitted by the queue.

The above simple model highlights a natural tension between two (potentially) conflicting goals: on the one hand it seems beneficial to focus on packets which require few processing cycles, since these can contribute to the throughput relatively fast. On the other hand, one is also enticed to favor large packets since these imply a significant boost to the throughput once they are completed. Our work addresses this dilemma, and presents algorithms with provable performance guarantees in such settings.

The main motivation for our model follows from managing queues in Network Processor (NPs) architectures. Such NPs are responsible for performing complex packet processing tasks in modern high-speed routers, including forwarding, classification, protocol conversion and intrusion detection, to name but a few. Common NPs usually rely on multi-core architectures, where multiple processors (also called *PPEs* – packet processing elements) perform the various processing tasks required by the arriving traffic. Such architectures are based on a pool of identical cores [4, 9], pipeline of cores [30], hybrid pool-pipeline [12], and multipass [10], and are very efficient for traffic that requires homogeneous processing time per packet. However, following operator demands, packet processing needs are becoming more involved and as a consequence processing time is increasingly heterogeneous, as NPs need to cope with more complex tasks (e.g., VPN encryption, hierarchical classification for QoS, etc.). The main concern in such settings is maximizing the throughput attainable by the NP, measured by the overall number of bytes corresponding to packets successfully processed by the system. It should be noted that modern architectures (e.g., OpenFlow [22]) support the availability of full information on the underlying traffic, such as the amount of processing required by each packet of a flow.

In addition to NP architectures, we believe our model and results may also be applicable to other queueing environments exhibiting a conflict between processing and size in settings where buffers have a bounded size.

1.1 Our Contributions

In this paper we provide a formal model for studying the problems of online buffer management and online scheduling in settings where packets have both size and processing requirements, and one has a limited size buffer to store arriving packets. Our model allows for studying the interplay between the (potentially) conflicting approaches of favoring large packets first, as opposed to favoring packets with fewer processing requirements, where the goal is to maximize the throughput of the queue. It should be noted that the offline version of the problem is NP-hard, as it encompasses Knapsack as a special case. For the more natural online settings, we provide both algorithms with provable performance guarantees as well as lower bounds on the performance of such algorithms.

We focus our attention on priority-based buffer management and scheduling, in both *push-out* settings, where admitted packets are allowed to be pushed out of the queue prior to having its processing completed (in which case the packet does not contribute to the system's throughput), as well as *non-push-out*, in which case buffer management decisions are limited to admission control. Specifically, we consider the following priority queueing regimes: (i) Shortest Remaining Processing Time (SRPT) first, common in job scheduling environments, and (ii) Longest-Packet (LP) first. We present competitive buffer management algorithms for these schemes, and further present lower bounds on the performance of algorithms for such priority queues. We show that the competitive ratio obtained by our algorithms depends on two fundamental parameters of the problem, namely, (i) the maximum size of a packet, and (ii) the maximum number of processing cycles required by a packet . We note that none of our algorithms requires knowledge of the maximum number of processing cycles in advance (although some of our algorithms need prior knowledge of the maximum allowed packet size, e.g., specified by a bound on the network MTU). We further show that this dependence is necessary by proving lower bounds on the performance of buffer management algorithm for the problem.

1.2 Related Work

Keslassy et al. [14] where the first to consider buffer management and scheduling in the context of network processors, where arriving traffic has heterogeneous processing requirements. They study both FIFO and SRPT priority schedulers, in both push-out and non-push-out buffer management regimes. They focused on the case where packets are of unit size, and showed competitive algorithms, as well as lower bounds, for such settings. They further introduced the notion of push-out costs which serves to balance the aggressiveness demonstrated by the the buffer management module. We believe the assumption made in [14] that packets are of unit size is very restrictive, since in real life NPs have to deal with packets of varying size, and it is unclear how should one design algorithms that ensure good throughput guarantees in such highly heterogeneous scenarios. The work of Keslassy et al. [14], as well as our current work, can be viewed as a part of the larger research effort of studying competitive algorithms for buffer management and scheduling, and specifically the study of such algorithms in bounded-buffers settings (see, e.g., a recent survey by Goldwasser [13] which provides an excellent overview of this field). This line of research, initiated in [18, 20], has received tremendous attention in the past decade, where various models where studied including QoS-oriented models where packets have weights (e.g., [1, 11, 18, 20]), models where packets have dependencies (e.g., [15, 21]), among others. It should be noted that most of these models focused on FIFO scheduling disciplines, which seem ill-suited for dealing with traffic where packets have heterogeneous processing requirements (as suggested by the results of [14]). A related field that has received much attention in recent years focuses on various switch architectures and aims at designing competitive algorithms for such multi-queue scenarios (e.g., [3, 5, 6, 16, 17]). Some works also provide experimental studies of these algorithms and further validate their performance [2].

There is a long history of OS scheduling for multithreaded processors which is germane to our research. For instance, the SRPT algorithm has been studied extensively in such systems, and it is well known to be optimal for mean response [27]. Additional objectives, models, and algorithms have been studied extensively in this context (e.g., [19, 23, 24]). For a comprehensive overview of competitive online scheduling

for server systems see the survey of Pruhs [25]. When comparing this body of research with our proposed framework one should note that OS scheduling is mostly concerned with average response time and average slowdown, while we focus on providing (worst-case) guarantees on the throughput. Furthermore, OS scheduling does not allow dropping of jobs, which is an inherent aspect of our proposed model, as implied the the fact we have a limited-size buffer, and overflowing packets must be dropped.

The model considered in our work is also closely related to Job-shop scheduling problems [8], and most notably to hybrid flow-shop scheduling [26], in scenarios where machines have bounded buffers. These models apply to a multitude of production-floor shop problems, where a pipeline of machines should perform a set of operations on a sequence of tasks, where each machine can only accumulate a bounded number of tasks awaiting execution at the current machine. Upon terminating the processing of a task at a given machine, it can be forwarded to the next machine only if the buffer at this subsequent machine is not full. Similarly to the case of OS scheduling, the main concern in these problems is designing algorithms whose main objective is optimizing aspects of delay (either minimizing average delay, minimizing tardiness in cases where tasks have deadlines, etc.). This line of research differs significantly from our work in that our main focus is maximizing throughput, and our model allows for dropping of packets (or tasks, if one uses the machine scheduling terminology), which is prohibited in job-shop problems.

1.3 Model Description and Algorithmic Framework

We consider a buffer with a bounded capacity of B bytes, handling the arrival of a sequence of packets. Each arriving packet p has some size $\ell(p) \in \{1, \ldots, L\}$ (in bytes), and some number of required processing cycles $r(p) \in \{1, \ldots, k\}$. Both of these quantities are known for every arriving packet.[1] The values of k and L will play a fundamental role in our analysis. We note, however, that none of our algorithms needs to know k in advance. The queue performs two main tasks, namely, *buffer management*, which handles admission control of newly arrived packets and push-out of currently stored packets, and *scheduling*, which decides which of the currently stored packets will be scheduled for processing. The scheduler will be determined by the *priority policy* employed by the queue. Our framework assumes a multi-core environment, where we have C processors, and at most C packets may be assigned for processing in any given time. However, for simplicity, in the remainder of this paper we assume the system may assign a single packet for processing at any given time (i.e., $C = 1$). This simple setting suffices to show both the intrinsic difficulties of the model, as well as our algorithmic scheme. We assume slotted time, where each time slot t consists of 3 phases: (i) *transmission*: in which packets with zero remaining required processing leave the queue, (ii) *arrival*: in which new packets arrive, and the buffer management module performs both admission-control and possibly push-out, and (iii) *assignment and processing*: in which a single packet is assigned for processing by the scheduling module. Figure 1 provides a graphic depiction of our general model (disregarding at this point the specific priority scheme employed by the system).

If a packet is dropped prior to being *transmitted* (i.e., while it still has a positive number of required processing cycles), then it is lost. Note that dropping a packet may occur either upon its arrival, or due to a push-out decision while it is stored in the buffer. A packet contributes its size to the objective function only upon being successfully transmitted. The goal is to devise buffer management algorithms for each of the considered priority regimes that aim at maximizing the overall throughput, i.e., the overall number of bytes transmitted from the queue.

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

We say that an arriving packet p *preempts* a packet q that has already been accepted into the buffer iff q is dropped in order to accommodate buffer space for p and p is admitted to the buffer instead. A

[1] The availability of this information is motivated by [29]. We note that assuming the size may be as small as 1 Byte is merely for simplicity, and can be viewed as merely a scaling assumption.

Algorithm 1 PO(p): Buffer Management Policy

1: accept p
2: **while** the last packet q in IB starts above position $B - 2L + 1$ **do**
3: drop q
4: **end while**

buffer management policy is called a *push-out* policy whenever it allows packets to preempt currently stored packets. For any algorithm ALG and any time-slot t, we define IB_t^{ALG} as the set of packets stored in the buffer of algorithm ALG at time t.

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

We consider both push-out and non-push-out policies. which are identified by the subscripts PO and NPO respectively. Furthermore, we will focus our attention on *priority-queueing* disciplines, which determine both the scheduling as well as the buffer management behaviour of the queue. Specifically, we will focus our attention on the following priorities, which differ by the parameter determining the priority. The parameters we consider are: (i) *processing*: in which the packet with the least amount of residual cycles has the top priority (referred to as SRPT), and (ii) *length*: in which the largest packet receives the top priority (referred to as LP).

Our goal is to provide performance guarantees for various buffer management algorithms working in various priority queueing disciplines. We use competitive analysis [7, 28] when evaluating the performance guarantees provided by our online algorithms. An algorithm ALG is said to be α-*competitive* (for some $\alpha \geq 1$) if for any arrival sequence σ, the overall length of packets successfully transmitted by ALG is at least $1/\alpha$ times the overall length of packets successfully delivered by an optimal solution (denoted OPT), obtained by a possibly offline clairvoyant algorithm.

Next we define both non-push-out (non-preemptive) and push-out (preemptive) algorithms that are used for all types of characteristics. The type of characteristic will be clear from the context of description. The Non-Push-Out Algorithm (NPO) is a simple greedy work-conserving policy that accepts a packet if there is enough available buffer space.

In the push-out case the generic algorithmic setting for the buffer management policy is defined in Algorithm 1. Note that algorithm PO is somewhat conservative in its use of the buffer, as can be seen from line 2. The reason for this will be clear from our results presented in Sections 3.2, 4.1.

We will sometimes use the term *value* to denote the overall length of a set of packets, and our analysis will be based on comparing the mapping value obtained by an optimal solution OPT, to that obtained by our algorithm. Specifically, we will make use of mappings between packets transmitted by OPT and those transmitted by our algorithm, such that their respective values differ by a mere multiplicative factor, which in turn will serve as the bound on the competitive ratio of our algorithm.

2 Useful Properties of Ordered Multiset

To facilitate our proofs, we will make use of properties of ordered (multi-)sets. These notions, as well as properties we show they satisfy, will enable us to compare the performance of our proposed algorithms with the optimal policy possible, for various priority disciplines. In the following, we consider multi-sets of real numbers, where we assume each multi-set is ordered in non-decreasing order. We will refer to such multi-sets as *ordered sets*. For every $1 \leq i \leq |A|$, we will further refer to element $a_i \in A$ or to $A[i]$ as the i-th element in the set A, as induced by the order. Given two ordered sets A, B, we say $A \geq B$, if for every i for which both a_i and b_i exist, $a_i \geq b_i$.

The following lemma, and its corollary, will be a fundamental tool used throughout our analysis. The proofs can be found in the appendix.

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

The following corollary shows that the same result holds if we add an item to only one of the sets.

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

3 Buffer Management with SRPT-based Priorities

In this section we address the problem of buffer management when the queueing discipline gives higher priority to packets with fewer required processing cycles.

3.1 Non-preemptive Policy

In this part we consider the performance of NPO greedy algorithm. The following theorem provides a lower bound on its performance and its proof can be found in Appendix.

Theorem 3. NPO *has competitive ratio at least kL for SRPT-based priorities.*

Next, we demonstrate an upper bound of the competitiveness for the NPO policy (the proof in Appendix).

Theorem 4. NPO *is at most $kL\frac{B}{B-L}$-competitive for SRPT-based priorities.*

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

3.2 Preemptive Policy

In this part we consider an upper bound of PO Algorithm 1. We first turn to provide a lower bound on the performance of PO for required-processing-based priorities (the proof can be found in Appendix).

Theorem 5. PO *has competitive ratio at least L for SRPT-based priorities, for $B \geq 2L$.*

3.2.1 Upper Bound of PO when $B > 2L$

In this section we provide a first upper bound of PO. Specifically, we prove the following

Theorem 6. *If $B > 2L$, then* PO *is at most $\frac{4L-2}{L_a}$-competitive for SRPT-based priorities, where L_a is an average length of packets transmitted by* PO.

In a later section we show how our bounds can be refined to show a better upper bound on the competitive ratio for the case of sufficiently large buffers.

In what follows we assume that OPT never preempts packets. Surely, such an optimal solution exists since one can consider the whole input being available to OPT a priori. Thus, any packets accepted by OPT are transmitted. Our analysis will be based on describing a mapping of packets in OPT's buffer to packets transmitted by PO, such that every packet q transmitted by PO has at most $4L - 2$ bytes of OPT associated with it. To facilitate the exposition we describe packet processing as if packets arrive individually and sequentially one at a time, though more than one packet might arrive at a single time step t. The mapping will be dynamically updated for each packet arrival and for each packet transmission, in both OPT and PO.

Mapping routine. During the transmission phase we distinguish between the three following cases:

T0 If both OPT and PO do not transmit then the mapping remains unchanged.

T1 If PO transmits a packet q then we remove its mapped image in OPT's buffer from future consideration in the mapping. The subset of these OPT packets or bytes that stays in OPT buffer at the end of transmission phase are called of type 1.

T2 If OPT transmits a packet p but its mapped packet q in PO is not transmitted then p is termed a packet of type 2. (We will show next that this case never occurs).

At time t, denote by M_t^O the ordered set of residual pass values for all non-type 1 OPT packets. All M_t^O values are grouped into blocks in the following way. A block is a minimal subset of consecutive M_t^O values starting from the lowest position that is not covered by any previous block, such that the overall length of the packets associated with the block is at least L. A minimal value in each block is called a block *representative*. Denote by R_t an ordered set of representatives at time t. In addition we denote by M_t^P an ordered set of processing cycles values of packets in PO's buffer at time t.

After the arrival at time t of a packet p we distinguish between the following cases:

A0 If p is not accepted by both OPT and PO, then the mapping remains unchanged.

A1 If after acceptance of p some PO packets were dropped then clear the mappings by step A1 between these PO packets and its mapped OPT mates. If p remains in PO's buffer and p is an i-th packet in it perform a (P, i)-*mapping-shift* (see Figure 3(a)): for each non-empty j-th block b and j-th PO packet q, with $j \geq i$ clear the mapping to q by step A1 and map all packets of block b to q. If p is accepted by OPT to the j-th block, perform an (O, j)-*mapping-shift* (see Figure 3(b)): clear all mappings by step A1 between packets of the old l-th block and l-th PO packet (if both exist), $l \geq j$, recompute blocks starting from the j-th and map packets of l-th block to l-th PO packet if both exist, $l \geq j$.

A2 Clear all mappings assigned by step A2. Map packets of all unmapped blocks to the HOL PO packet.

Claim 7. *The mapping is feasible.*

Proof. By definition PO accepts the arriving packet and all the packets with packet start above $B - 2L + 1$ are dropped. Hence, if after the application of step A1 of the mapping routine there are still unmapped OPT packets then it must be the case that PO buffer contains at least one packet. Therefore, all unmapped by Step A1 OPT packets are mapped by step A2. □

Claim 8. *The overall length of packets of the same block is at most* $2L - 1$.

Proof. Essentially, in the worst case an overall length of all packets in the block except the last one is $L - 1$ and the last packet of the same block has L length. Henceforth, the claim follows. □

Lemma 9. *After the arrival of the* t*-th packet, if an* OPT *packet* p *is mapped to a (possibly transmitted)* PO *packet* q *then* $r_t(p) \geq r_t(q)$. *Moreover, all* OPT *packets are mapped and at most* $2L - 1$ *bytes are mapped to each* PO *packet by step A1, and possibly at most an additional* $2L - 1$ *bytes are mapped to the HOL packet by step A2, at any time* t.

Proof. We prove the lemma by induction on the number of arrived packets. For the base case, consider the first arriving packet. By definition PO always accepts an arriving packet p. If p is dropped by OPT then the claim trivially holds. If p is accepted by OPT, it creates a new block where p becomes its representative. Clearly, $r_1(p) \geq r_1(p)$, all OPT packets are mapped, and at most L bytes are mapped to the PO packet p. Thus, the base case holds.

Assume by induction that for any time $t' < t$, after the arrival of the t'-th packet it holds that for any OPT packet p that is mapped to a (possibly transmitted) PO packet q, $r_{t'}(p) \geq r_{t'}(q)$. Moreover, all OPT

packets are mapped and at most $2L-1$ OPT bytes are mapped to each PO packet by step A1. In addition at most $2L-1$ OPT bytes are mapped to the HOL packet in PO buffer at time t' by step A2.

Clearly, if for a representative of a block p' that is mapped to a PO packet q by step A1 it holds that $r_{t'}(p') \geq r_{t'}(q)$ at time t', then for any packet p'' of the same block $r_{t'}(p'') \geq r_{t'}(q)$. In order to show this inequality after the arrival of the t-th packet, it suffices to focus our attention on the ordered set of representatives R_{t-1} and its update after the arrival of the t-th packet. Specifically, it suffices to show that $R_t \geq M_t^P$. By the induction hypothesis the remaining number of processing cycles of any OPT packet is at least the number of processing cycles of its PO mapped mate, i.e., $R_{t-1} \geq M_{t-1}^P$, and there are no OPT packets of type 2 formed during acceptance of the first $t-1$ packets. Denote by $R_{t'}^1$ a set of representatives whose blocks are mapped by step A1. Since all packets of the same block are mapped to the same PO packet then $\left|R_{t-1}^1\right| \leq \left|M_{t-1}^P\right|$.

Denote by $t-$ the time that is just before the arrival of t-th packet. We first consider the possibility that a transmission occurs prior to the arrival of the t-th packet. Assume that between t-th and $t-1$-th packet arrivals at least one packet is transmitted by OPT or PO. By the induction hypothesis, it is impossible for OPT to transmit a packet corresponding to the first value in M_{t-1}^P before the packet whose value is the first in M_{t-1}^P. Note that this will hold for any sequence of transmission occurring prior to the t-th arrival. Therefore, if PO transmits between $t-1$-th and t-th packets arrival, then $\left|M_{t-}^P\right|$ is reduced by one. On the other hand, $\left|M_{t-}^O\right|$ is reduced by the number of packets in the first block (if existed) that is mapped by step A1 to the packet sent by PO. Hence, $\left|R_{t-}^1\right| \leq \left|M_{t-}^P\right|$. In addition if some value is mapped by step A2 to a packet transmitted by PO then this value is removed from M_{t-}^O upon this transmission (by the definition of M_t^O which consists solely of non-type 1 packets). Thus, in this case $R_{t-} = R_{t-}^1$ and $|R_{t-}| \leq \left|M_{t-}^P\right|$. It follows that the claim is satisfied at time $t-$, in particular $R_{t-} \geq M_{t-}^P$.

Consider now the arrival of the t-th packet p. We distinguish between several cases.

Case 0 OPT does not accept p and PO accepts and immediately drops p. We are done.

Case 1 PO does not drop p and OPT does not accept p. Note that in this case $R_t = R_{t-}$ and it therefore suffices to show that $R_{t-} \geq M_t^P$.

Case 1.1 Assume first that $|R_{t-}| \geq \left|M_{t-}^P\right|$. Since $\left|R_{t-}^1\right| \leq \left|M_{t-}^P\right|$, some OPT packets that are represented in R_{t-} are mapped by step A2. In this case the last packet in the buffer of PO occupies the $(B-2L+1)$-th byte (since each block is of length at least L, each block is mapped to a single PO packet and all bytes in PO buffer are available for mapping). Since PO does not drop p, the value of $r_t(p)$ is at most the last value in M_t^P. Since in this case OPT does not accept p, by Corollary 2 case (i), $R_{t-} \geq M_{t-}^P \cup \{r_t(p)\} = M_t^P$.

Case 1.2 Consider next the case where $\left|R_{t-}^1\right| \leq \left|M_{t-}^P\right|$. Again, since in this case OPT does not accept p, by Corollary 2 case (ii), $R_{t-} \geq M_{t-}^P \cup \{r_t(p)\} = M_t^P$.

Case 2 OPT accepts p and PO drops p. In this case $M_t^P = M_{t-}^P$ and $r_t(p)$ is larger than any value in M_t^P. Let l be the position of p in the buffer of OPT. Thus, for any $m \geq l$ the m-th OPT packet has a number of residual processing cycles larger than any value in M_t^P. Therefore, for any OPT packet p' that is mapped to PO packet q by step A1, $r_t(p') \geq r_t(q)$, and we have $R_t \geq M_t^P$.

Case 3 OPT accepts p and PO does not drop p. If $|R_{t-}| \leq \left|M_{t-}^P\right|$ or $|R_{t-}| \geq \left|M_{t-}^P\right|$, then similarly to the Cases 1.1 and 1.2, by Lemma 1 we have that

$$R' = R_{t-} \cup \{r_t(p)\} \geq M_{t-}^P \cup \{r_t(p)\} = M_t^P.$$

Therefore, in this case it suffices to show that $R_t \geq R'$, which in turn implies $R_t \geq M_t^P$. Let j denote the index of the block where p is inserted in OPT. We have to consider two possibilities for the position of p's number of processing cycles in R', which could be either the j-th or the $(j+1)$-th position.

Case 3.1 $R'[j] = r_t(p)$. In this case p now serves as the representative of block j, i.e., $R_t[j] = r_t(p)$. If $\ell(p) = L$ then p forms a full block and $R_t[m] = R^1_{t-}[m-1]$ for all $m > j$. Therefore, $R_t \geq R'$ (as a matter of fact, in this case we have strict equality). Otherwise we have $\ell(p) < L$. It follows that a number of residual processing cycles of at least the $(l+1)$-th element will join the j-th block after recomputation (since such a block must add to overall length at least L). We therefore have $R_t[m] \geq R^1_{t-}[m]$ for all $m > j$. Since $R'[m] = R_{t-}[m-1]$, $m > j$, this case follows.

Case 3.2 $R'[j+1] = r_t(p)$. In this case the representative of block j remains unchanged, i.e., $R'[j] = R_t[j] = R_{t-}[j]$ and $R'[m] = R_{t-}[m-1]$, $m > j+1$. Since p belongs to the j-th block after acceptance and $r_t(p)$ is not a representative of the block, then $R_t[j+1] \geq r_t(p)$. Since after recomputation representatives will move up for no more than one block in R_t compared to R_{t-1}, $R_t[m] \geq R_{t-}[m-1]$, $m > j+1$. Therefore, $R_t \geq R'$ and this case follows.

Now let us to show that there are sufficiently many PO packets in order to map all of OPT packets such that at most $2L-1$ bytes are assigned by step A1 to each transmitted packet of PO and additionally at most $2L-1$ bytes are assigned to the HOL packet of PO by step A2. Recall that the claim holds for time $t-$. Consider the arrival of the t-th packet p. If PO accepts p, then the claim holds, since this new packet can support the block changes (and possible addition) that may potentially occur if OPT also accepts p. If PO does not accept p then by the definition of PO this can only happen if the buffer occupancy of PO is at least $B - 2L + 1$. Clearly, the overall length of a block mapped by step A1 to any PO packet is at most $2L-1$ (by definition). Furthermore, since we have shown that $R_t \geq M^P_t$, and by definition the blocks are of overall length at least L, it must follow that the overall length of packets in PO covers at least this amount of overall length of packets in OPT mapped to PO by step A1. It follows that the remaining overall length of packets in the buffer of OPT that are not mapped by step A1 can be at most $2L-1$ (the possibly unused space in PO). It follows that the overall length of packets mapped to the HOL packet of PO by step A2 is at most $2L-1$, as required. □

The proof of Theorem 6 now follows immediately from Lemma 9.

3.2.2 Upper Bound of PO for Sufficiently Large Buffers

In this section we will generalize the previous mapping and show how to improve the upper bound of PO for sufficiently large buffers.

Theorem 10. PO *is at most* PO *is at most* $\frac{(2L-1)(N+1)}{NL_a}$*-competitive for SRPT-based priorities, where* L_a *is an average length between all transmitted packets by* PO *and* $N = \lceil \frac{B-2L+1}{2L-1} \rceil$.

The idea is to redistribute mapped bytes by step A2 between different PO packets. Let $N = \lceil \frac{B-2L+1}{2L-1} \rceil$. We consider an updated version of step A2 and now to each PO packet can be mapped at most $\frac{2L-1}{N}$ value by step A2.

The mapping routine is unchanged during the transmission phase and now it operates on M^O_t in the following way. Denote by M^O_t at time t an ordered set of values of processing cycles of no-type 1 OPT packets that are not mapped by the step A2 as defined below. Observe that now we exclude from the future consideration by step A1 all OPT packets that are mapped by step A2 even before its PO mates are transmitted. The definition of block, representative, R_t and M^P_t remain unchanged.

During the arrival of a packet p at time t, steps A0 and A1 remain unchanged. Next, we define the changed or new steps.

A2 If prior to time t there are no bytes mapped by step A2 and after the application of t-th step A1 there are still Y unmapped OPT bytes then each j-th portion of $\frac{Y}{N}$ unmapped bytes by step A1 map to PO

packet whose mapped block contains $(j-1)(2L-1)+1$-th byte x, $1 \leq j \leq N$. We say that such byte x "defines" a mapping of this portion of still unmapped bytes. Let Y be the overall length mapped by step A2 prior to time t and still there are Y_0 unmapped bytes after applying step A1 during time t. Let the mapping of the Y-th byte that is assigned by step A2 be the l-th byte in the OPT buffer. Map each j-th portion of $\frac{Y_0}{N}$ still unmapped by step A1 byte to PO packet whose mapped block contains $j(2L-1)+l+1$-th byte, $1 \leq j \leq N$. Observe that both these bytes can be remapped to the other PO packet during the (O,j)-*mapping-shift* .

A3 Any unmapped value by Steps A1 and A2 are assigned to the HOL PO packet. We are to show that step A3 is never applied and is required only for completeness.

The mapping is feasible since during arrivals the PO buffer contains at least one packet and any value that is unmapped by Steps A1 and A2 is assigned by step A3 to the HOL PO packet. The Claim 8 remains the same.

The next lemma is very similar to Lemma 9. Namely, if an OPT packet p is mapped by step A1 to a (possibly transmitted) PO packet q then $r_t(p) \geq r_t(q)$. The fact that the overall assigned value to each PO packet is at most $\frac{(2L-1)(N+1)}{N}$ follows from the fact that for each OPT packet p that is mapped by step A1 to a PO packet q at any time t, $r_t(p) \geq r_t(q)$, the maximal block size is $2L-1$ bytes.

Lemma 11. *After arrival of the t-th packet, if an OPT packet p is mapped to a (possibly transmitted)* PO *packet q then $r_t(p) \geq r_t(q)$. Moreover, all OPT packets are mapped and at most $\frac{(2L-1)(N+1)}{N}$ value is mapped to each* PO *packet at time t, where $N = \lceil \frac{B-2L+1}{2L-1} \rceil$.*

The proof of Theorem 10 follows immediately from Lemma 11.

Corollary 12. *If $B > 4L^2 - 2L$, then* PO *is at most $\frac{2L}{L_a}$-competitive for SRPT-based priorities, where L_a is an average length of packets transmitted by* PO.

4 Buffer Management with LP-based Priorities

In this section we concentrate our attention to the preemptive case since non-preemptive results are very similar to Section 3.

4.1 Preemptive Policy

Now we will show the lower bound of the PO algorithm (the proof in Appendix).

Theorem 13. PO *is more than k-competitive for LP-based priorities on a sufficiently long sequence.*

Next we consider an upper bound of PO and show the following result.

Theorem 14. PO *is at most $k+3$-competitive for LP-based priorities on sufficiently big buffers.*

The mapping routine is unchanged during the transmission phase as in Section 3. At time t, denote by M_t^O a set of non-type 1 packets sojourns in OPT buffer and not mapped by step A2. In addition M_t^O is ordered in non increasing order of packet length. All M_t^O packets are grouped into blocks in the following way. Let q be an i-th packet in PO buffer at time t. An i-th block is defined in the following way. Consider a minimal set B_0 of packets starting from the lowest position that are represented in M_t^O and not covered by any other block whose overall required work is at least $r_t(q)$. If the overall length of all packets in B_0 is at least $\ell(q)$ then B_0 forms a block. Otherwise, add to B_0 a minimal set of packets B_1 starting from the first packet that is represented in M_t^O and not covered by B_0 such that the overall length of packets in $B_0 \cup B_1$ will be at least $\ell(q)$. In this case a set of packets that is covered by $B_0 \cup B_1$ defines a block. Denote by

$\ell(X)$ the overall length of packets and by $r_t(X)$ the overall required work in a set of packets X at time t. A block b that is mapped to a PO packet q is called *fully mapped* to a packet q at time t if $\ell(b) \geq \ell(q)$ and $r_t(b) \geq r_t(q)$. Observe that it is possible that $\ell(B_0)$ will be less than its mapped PO mate. In this case OPT may later accept packets that will not be accepted by PO.

During the arrival of a packet p at time t Steps A0, A2 and A3 are the same as in Subsection 3.2.2. Next we define a new Step A1 where the blocks are recomputed after (P,i)-*mapping-shift* .

A1 If after acceptance of p some PO packets were dropped then clear the mappings by step A1 between these PO packets and its mapped OPT mates. If p remains in PO buffer and p is an i-th packet in the PO buffer perform a (P,i)-*mapping-shift* : clear all mappings by step A1 between packets of the old l-th block in OPT buffer and l-th packet in PO buffer, $l \geq j$, recompute blocks from j-th and map packets of l-th block to the l-th PO packet if both exist, $l \geq j$. If p is accepted by OPT to j-th block, perform an (O,j)-*mapping-shift*: clear all mappings by step A1 between packets in OPT buffer of the old l-th block and l-th packet in PO buffer (if both exist), $l \geq j$, recompute blocks from j-th and map packets of l-th block to l-th PO packet if both exist, $l \geq j$.

Next, we will explain the the intuition of the proof of Theorem 14. Clearly, the mapping is feasible since if OPT accepts some packet that is not accepted by PO, PO buffer contains at least one packet in its buffer. Since affected blocks are recomputed after each (P,i)-*mapping-shift* and (O,j)- *mapping-shift* and by definition of a block b that is mapped to a PO packet q at time t, $\ell(b) \geq \ell(q)$ and $r_t(b) \geq r_t(q)$. Thus, we will consider sufficiently big buffers where $\frac{2L-1}{B}$ tends to zero and because of the above properties of the block step A2 will introduce at most additional ϵ value for each PO packet. Hence, for each transmitted by PO packet q, OPT transmits at most $(k+1)l(q)+\epsilon$ by Steps A1 and A2. Denote by T the overall number of transmitted bytes by PO and by P the total number of transmitted bytes by OPT during processing of preempted PO packets. Thus, the competitive ratio is at most $\frac{(k+1+\epsilon)T+P}{T}$. Now let us estimate P and substitute to the previous expression. For each preempted by PO packet p denote by $T(p)$ a number of time slots when p was HOL before it was preempted. Clearly, that the process of preemptions of packets that have positive $T(p)$ will be stopped once all packets will have a maximal packet length L or it can continue during each time slot when there is at least one packet in PO buffer of length smaller than L. Moreover, if preemption happens the buffer occupancy is at least $B-2L+1$. Denote by $\mathcal{P}$ a set of PO packets preempted during this interval of time. So for each $B-2L+1$ bytes transmitted by PO, P is bounded by $\sum_{p\in\mathcal{P}} T(p)l(p) \leq \sum_{p\in\mathcal{P}} kl(p)] \leq kL(L+1)/2$. Thus, PO is at most $\frac{2(k+1+\epsilon)B+kL(L+1)}{2(B-2L+1)}$-competitive. For the buffers that are significantly bigger than $kL(L+1)$, PO is at most $k+3$-competitive.

5 Discussion

The increasingly-heterogeneous packet-processing needs of NP traffic pose novel design challenges to NP architects. In this paper we provide performance guarantees for various NP buffer scheduling algorithms for packets with heterogeneous lengths. The objective is to maximize the amount of transmitted bytes under various settings such as push-out and non-push-out buffers and various packet ordering strategies. As future work, it remains to extend these results to a strategy which prioritizes packets by their residual processing requirement to size ratio.

References

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

[2] Susanne Albers and Tobias Jacobs. An experimental study of new and known online packet buffering algorithms. *Algorithmica*, 57(4):725–746, 2010.

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

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

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

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

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

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

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

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

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

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

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

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

[15] 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.

[16] Alexander Kesselman, Kirill Kogan, and Michael Segal. Improved competitive performance bounds for cioq switches. In *ESA*, pages 577–588, 2008.

[17] 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.

[18] 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.

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

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

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

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

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

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

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

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

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

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

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

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

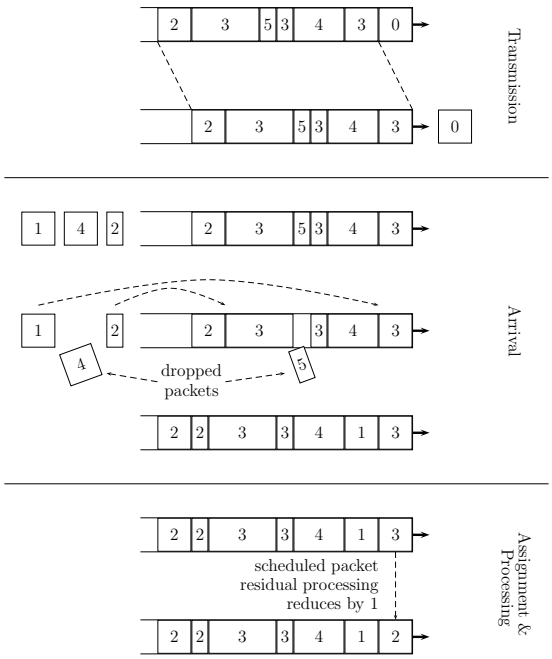
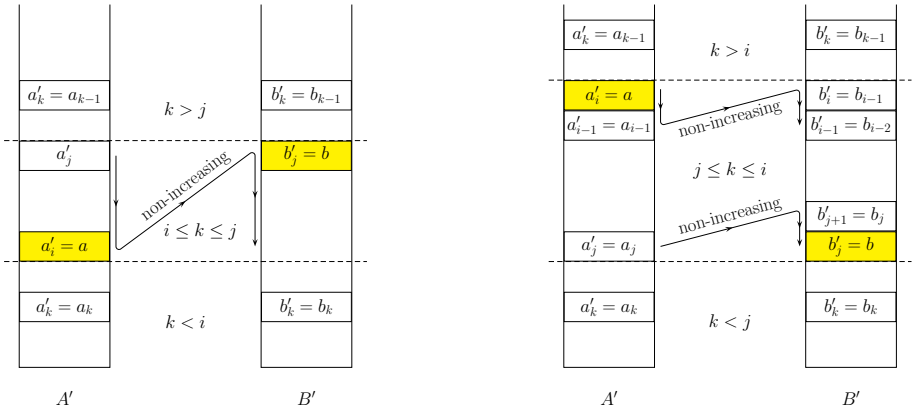


Figure 1: An outline of the model. The top subfigure shows the transmission phase, the middle subfigure shows the arrival phase where packets might be discarded, and the bottom subfigure shows the assignment and processing phase. The length of a packet represents its size, and the number stamped on the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



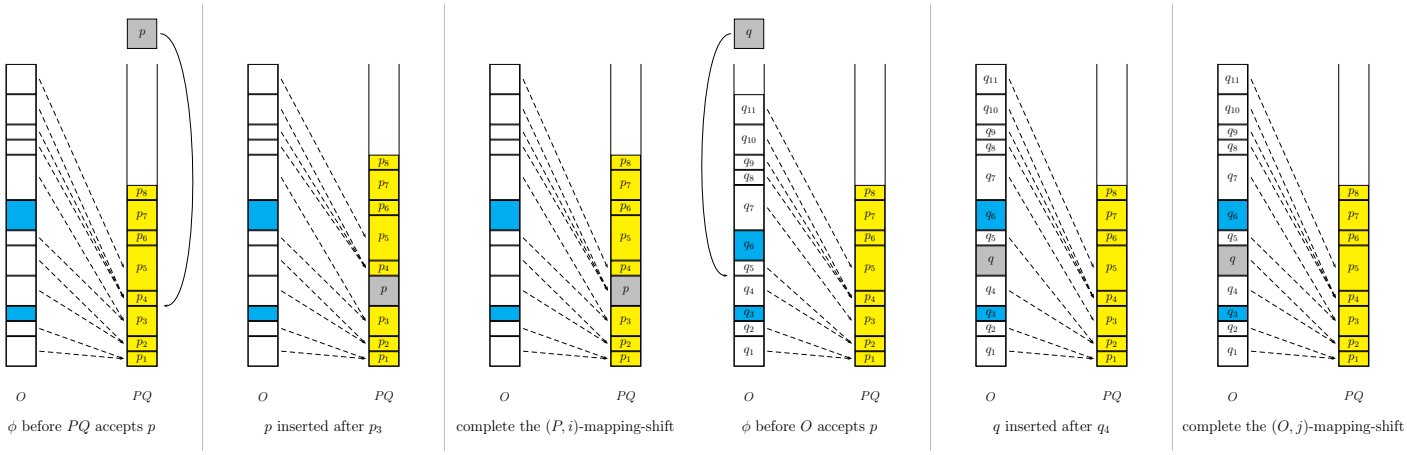
(a) Case $i \le j$ (b) Case $i > j$

Figure 2: Cases of Lemma 1.

A Appendix

Proof of Lemma 1. We will refer to elements in A' and B' as a' and b', respectively. Assume i and j are the positions of $a \in A'$ and $b \in B'$, respectively. I.e., $a'_i = a$ and $b'_j = b$. We need to show that for every k for which both a'_k and b'_k exist, $a'_k \ge b'_k$. We distinguish between 2 cases:

(a) $i \le j$ (see Figure 2(a)): for all $k < i$, $a'_k = a_k$, and $b'_k = b_k$, hence by the assumption that $A \ge B$, $a'_k \ge b'_k$. By the assumption that $a \ge b$, and the fact A' and B' are ordered, for every $p \ge i$ and $q < j$ we have $a'_p \ge a'_i \ge b'_j \ge b'_q$. In particular, for every $i \le k < j$ we have $a'_k \ge b'_k$ (by taking $p = q = k$). For $k = j$, since A' and B' are ordered, and since in the current case $i \le j$, we have $a'_j \ge a'_i = a \ge b = b'_j$. For $k > j$ we have $a'_k = a_{k-1} \ge b_{k-1} = b'_k$, where the inequality follows from the assumption that $A \ge B$.



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

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

Figure 3: Example of the mapping used in the proof of Theorem 6, and the mapping shifts performed by the analysis. The white OPT packets should be mapped and white PO packets are available for mapping. The blue OPT packets are of type 1.

(b) $i > j$ (see Figure 2(b)): for all $k < j$, $a'_k = a_k$, and $b'_k = b_k$, hence by the assumption that $A \geq B$, $a'_k \geq b'_k$. For $k = j$, $b'_k \leq b'_{k+1} = b_k \leq a_k = a'_k$, which follows from the fact that b is inserted in slot $j = k$, B' is ordered, the assumption that $A \geq B$ and $b \leq b_{|B|}$ or $|A| \leq |B|$. For $j < k < i$, $b'_k = b_{k-1} \leq a_{k-1} \leq a_k = a'_k$, which follows from the assumption that $A \geq B$. For $k = i$, $a = a'_i \geq a_{i-1} \geq b_{i-1} = b'_i$. For $k > i$, $a'_k = a_{k-1} \geq b_{k-1} = b'_k$.

We are therefore guaranteed to have $A' \geq B'$, as required. □

Proof of Corollary 2. Assume b is inserted in B' in location j. Consider a virtual item a, such that $a > \max\{a_{|A|}, b\}$. We now virtually consider adding both a and b to sets A and B, respectively. By Lemma 1, it follows that the resulting sets A', B' satisfy $A' \geq B'$. Notice that the first $|A|$ elements of A' is exactly the set A (by the choice of a), implying that we also have $A \geq B'$. □

Proof of Theorem 3. During the first time slot a burst of $B - L + 1$ 1-byte packets with k processing cycles arrives followed by B/L packets of L length with a single residual pass. NPO accepts all 1-byte packets and rejects all L packets. On the other hand OPT drops all 1-byte packets and fills up its buffer with L-bytes packets. At each subsequent time slot one 1-byte packet with k processing cycles and one L-bytes packet with one residual pass arrive. Clearly, no L packet is accepted by NPO. Moreover, after the first time slot OPT buffer is full. Later, while there are non-empty arrivals for each accepted and transmitted by NPO 1-byte packet, OPT accepts and transmits kL bytes. At the last time slot with non-empty arrivals, the NPO buffer contains less than B bytes but the OPT buffer is full. Thus, the competitiveness of NPO is more than kL for a long enough sequence of arrivals. □

Proof of Theorem 4. Observe that NPO must fill up its buffer before it drops any packets. Moreover, so long as the NPO buffer is not empty then after at most k time steps NPO must transmit its HOL packet. This means that NPO is transmitting at a rate of at least 1-byte every k time steps, while OPT in the same time interval transmitted at most k packets each of size L. Hence, the number of transmitted bytes at time t for NPO is at least t/k [2] while OPT transmitted at most tL bytes for a competitive ratio of kL so long as the NPO buffer did not become empty before OPT's did.

If, on the other hand, NPO empties its buffer first, this means there were no packet arrivals since the NPO buffer went below the $B - L + 1$ threshold at a time t. From that moment on NPO empties its buffer transmitting thus at least $B - L$ bytes, while OPT transmitted at most B bytes.

[2] We assume k divides t evenly for simplicity of exposition.

So in total the number of bytes transmitted by NPO is at least $\frac{t}{k}+B-L$ while the total number of bytes transmitted by OPT is $tL+B$ with competitive ratio

$$\frac{tL+B}{t/k+B-L} \leq \frac{tL+kLB}{t/k+B-L} = kL\frac{t+kB}{t+k(B-L)} \leq kL\frac{B}{B-L}$$

. □

Proof of Theorem 6. Assume that B/L is an integer. All packets received will have a single residual pass. Consider the following sequence of arrivals. At the beginning $B-2L+1$ 1-byte packets arrive. PO accepts all of them. OPT drops all of them. Later on during the same time slot B/L packets of length L arrive, each with a single residual pass. PO drops all of them since their value is no better than the value of packets in its buffer, but OPT accepts all of them and thus OPT buffer is full. During each following time slot one 1-byte packet arrives, each requiring a single processing cycle, followed by one packet of size L bytes, requiring a single processing cycle. PO accepts all 1-byte packets but it does not accept any of the L-bytes packets. Thus, for each time slot when there are arrivals, OPT transmits a packet of size L, and at the same time PO transmits a 1-byte packet. At the end, OPT transmits B bytes while PO transmits $B-2L+1$ additional bytes. Therefore, $B+nL$ and $B-2L+1+n$ bytes are transmitted by OPT and PO, respectively, where n is a number of time slots with non-empty arrivals. We obtain that for $n >> B$, PO cannot have a competitive ratio better than L. □

Proof of Theorem 13. Here, we will consider preemptive version of OPT for simplicity of description. Assume $\frac{B}{L}$ be an integer value. Consider a cycle of L iterations of the first type and later sequence of $n>0$ iterations of the second type (defined below). Each iteration of the first type contains $k-1$ time slots. At the beginning of the i-th iteration of the first type $\lceil \frac{B}{i} \rceil$ packets of i bytes with k processing cycles arrive and later during the same time slot $\lceil \frac{B}{i} \rceil$ packets of i bytes with 1 processing cycles arrive. OPT drops the first subsequence and accepts the second. On the other hand PO accepts the first subsequence and drops the second. So during each iteration of the first type OPT transmits $i(k-1)$ bytes but PO transmits zero bytes. At the beginning of the next iteration of the first type both algorithms preempt already admitted packets that still remain in the buffers.

After the L-th iteration both buffers are nearly full with packets of size L, but with k processing cycles in the case of PO and one residual pass in the case of OPT. Now a sequence of the second type starts. After the last transmission by PO, $k+1$ packets arrive in the following order: first one L-byte packet with k passes and thereafter k packets of length L with a single residual pass. The first packet is accepted by PO and dropped by OPT. The latter $k-1$ packets are dropped by PO and accepted by OPT. Each buffer is completely full again. So during each iteration of the second type OPT transmits kL bytes but PO only L bytes. After n iterations of the second type the overall transmission of OPT is $\frac{L(1+L)(k-1)}{2}+knL+B$ while PO transmits $Ln+B$ bytes. Thus, the lower bound on competitive ratio of PO is $\frac{L(1+L)(k-1)+2kLn+2B}{2(Ln+B)}$. □