

# Packet-Mode Policies for Input-Queued Switches

Dan Guez  
Dept. of Computer Science  
Technion  
Haifa, Israel  
guezd@cs.technion.ac.il

Alex Kesselman  
Max Planck Institut für  
Informatik  
Saarbrücken, Germany  
akessel@mpi-sb.mpg.de

Adi Rosén  
Dept. of Computer Science  
Technion  
Haifa, Israel  
adiro@cs.technion.ac.il

## ABSTRACT

This paper considers the problem of packet-mode scheduling of input queued switches. Packets have variable lengths, and are divided into cells of unit length. Each packet arrives to the switch with a given deadline by which it must traverse the switch. A packet successfully passes the switch if the sequence of cells comprising it is *contiguously* transmitted out of the switch before the packet's deadline expires. A packet transmission may be preempted and restarted from the beginning later. The scheduling policy has to decide at each time step which packets to serve. The problem is online in nature, and thus we use competitive analysis to measure the performance of our scheduling policies.

First we consider the case where the goal of the switch policy is to maximize the total number of successfully transmitted packets. We derive two algorithms achieving the competitive ratios of  $(2^{2\sqrt{\log L}} + 1)$  and  $N + 1$ , respectively, where  $L$  is the ratio between the longest and the shortest packet lengths and  $N$  is the number of input/output ports. We also show that any deterministic online algorithm has a competitive ratio of at least  $\min(\lfloor \log L \rfloor + 1, N)$ .

Then we study the general case in which each packet has an intrinsic value representing its priority, and the goal is to maximize the total value of successfully transmitted packets. We derive an algorithm which achieves a competitive ratio of  $2\kappa + 2\sqrt{\kappa + 1/2} + \frac{2\kappa + \sqrt{\kappa + 1/2} + 1}{\sqrt{\kappa + 1/2}} + 3$ , where  $\kappa$  is the ratio between the maximum and the minimum value per cell. We note that [4] gives a lower bound of  $\Omega(\kappa)$  on the performance of any deterministic online algorithm. In particular, our algorithm achieves a competitive ratio of approximately 11.123 for  $\kappa = 1$ , which improves upon the previous best-known upper bound for this problem [17].

We complement our results by studying the offline version of the problem, which is NP-hard. We give a pseudo-polynomial 3-approximation algorithm for the general case and a polynomial 3-approximation algorithm for the case of unit value packets.

## Categories and Subject Descriptors

C.2.1 [Packet-switching networks, Store and forward networks];  
F.2.2 [Routing and layout, Sequencing and scheduling]

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SPAA'04, June 27–30, 2004, Barcelona, Spain.  
Copyright 2004 ACM 1-58113-840-7/04/0006 ...\$5.00.

## General Terms

Algorithms, Performance, Theory

## Keywords

Input-Queued Switches, Packet-Mode Scheduling, Competitive Analysis

## 1. INTRODUCTION

The explosive growth of voice and video traffic in the Internet generates new Quality of Service (QoS) requirements. QoS mechanisms can generally be described as mechanisms for a network to satisfy the varied grade of service, such as maximum delay or minimum bandwidth, required by different applications. In a packet-switched network, multiple traffic streams pass through a sequence of switches. Due to the high variability of the Internet traffic, simple scheduling strategies such as FIFO or Round-Robin might not guarantee the adequate QoS to all applications. The problem is that packets of real-time applications such as voice and video cannot be delayed outside of a small pre-defined time window, i.e., there is a maximum permissible delay of such a packet at a router. In addition, packets may have different priorities. For example, a packet carrying an Independent (I) frame of an MPEG video stream is more important than a packet carrying a Predicted (P) frame, although they have the same delay requirements. In this paper we consider switch policies that take into account the permissible delay and the priority of the various packets.

We consider the input queuing (IQ) switch architecture, where packets arriving from the input lines are queued at the input ports. The packets are then extracted from the input queues to cross the switch fabric and to be forwarded to the output ports. An IQ switch operates on fixed-size cells and the time that elapses between two consecutive switching decisions is called a time slot. Each arriving IP packet is divided into cells at the input, and these cells are scheduled across the switch fabric separately. A major issue in the design of IQ switches is the scheduler that controls the access to the switching fabric in order to avoid contention at the input ports and at the output ports. A large number of cell scheduling algorithms have been proposed in the literature for the IQ switch architecture: these are PIM [2], IRRM [23], iSLIP [21], iOCF [22], RPA [20] and Batch [8], to name just a few. All the above mentioned works consider the scheduling problem on the cell level; the underlying architecture is such that the various cells comprising a given IP packet are buffered at the output port, and when all cells comprising a packet are collected, the packet is re-assembled and transmitted on the output line [9].

Cell-mode scheduling has a number of drawbacks. First, one must keep a packet-reassembly buffer at each output port. Second,

traditional cell-mode schedulers are typically unaware of the existence of packets, and thus different cells of the same packet may experience different delays. This may badly affect the QoS perceived by the user because the actual delay of a packet is the delay of the last cell. As a result, *packet-mode* scheduling of input queued switches, where the whole packet rather than a single cell becomes the switching unit, has received considerable attention. In packet-mode switching the scheduling policy is constrained to schedule all cells of a given packets *contiguously*. The main advantage of this architecture is that the reassembly overhead is saved. In addition, a packet-based scheduler, which is aware of the packet entity, may use this information to provide better performance through scheduling. The works in [19, 24, 9] consider scenarios where packets (with no specific deadlines) arrive to the switch over time, and study the throughput of the switch. Scheduling of packets with individual deadlines in the context of a single buffer is considered in [15].

In the present paper we study the problem of packet-mode scheduling in IQ switches, where each packet has a length, a deadline by which it must be sent out of the switch, and a value. This model is warranted by networks that guarantee end-to-end delay and support different packet priorities (cf. the DiffServ model [5]). A scheduling policy is presented with packet arrivals and has to serve packets online, i.e., without knowledge of future arrivals. We do not make any assumptions about the incoming traffic. To the best of our knowledge, the present work is the first to consider the general setting of this problem, where packets may have arbitrary values, lengths, and deadlines.

A packet is said to successfully complete its transmission if it is contiguously scheduled for a number of time slots equal to its length, and the last cell is scheduled by the packet's deadline. In this case the packet value is accrued by the system. The scheduler may preempt the transmission of a packet and restart it later *from the beginning* (The ability to preempt packets helps in situations in which arriving voice or video packets require immediate service while the switch fabric is busy transferring a large data packet). The aim of the system is that of maximizing the total value of the successfully transmitted packets. For the case of unit-value packets, the goal is that of maximizing the number of successfully transmitted packets.

We use competitive analysis [25, 6] to study the performance of our algorithms. In competitive analysis, an online algorithm *ALG* is compared with an optimal offline algorithm *OPT*, that knows the entire input sequence in advance. The advantage of competitive analysis is that a uniform performance guarantee is provided over all input instances. Denote the benefit accrued by *ALG* and by *OPT* on an input sequence  $\sigma$  by  $V^{ALG}(\sigma)$  and  $V^{OPT}(\sigma)$ , respectively. We say that *ALG* is  $c$ -competitive if for every sequence of packets  $\sigma$ ,  $V^{OPT}(\sigma) \leq c \cdot V^{ALG}(\sigma) + a$ , where  $a$  is a constant independent of  $\sigma$ .

**Our results.** The the sequel we use the following terms. The *slack* of a packet is the number of time slots the scheduler may wait before starting to transmit the packet so as to finish its transmission before its deadline expires. The *value density* of a packet is its value divided by its length (in cells). The *importance ratio*  $\kappa$  is the ratio of the maximal to the minimal value density (the special case of  $\kappa = 1$  is called the uniform value density case). The *length ratio*,  $L$ , is the length (in cells) of the longest packet assuming that the shortest one has just one cell.

First we consider the special case of unit value packets. We derive two algorithms: the first algorithm achieves a competitive ratio of  $(2^{2^{\sqrt{\log L}}} + 1)$  and the second algorithm achieves a competitive ratio of  $N + 1$ , where  $N$  is the number of input/output

ports. We also show a lower bound of  $\min(\lfloor \log L \rfloor + 1, N)$  on the competitive ratio of any deterministic online algorithm. We further give a stronger lower bound that pertains to a class of deterministic algorithms based on maximum length-weighted matching. Then we study the general case of variable value packets. We present an algorithm that achieves a competitive ratio of  $2\kappa + 2\sqrt{\kappa + 1/2} + \frac{2\kappa + \sqrt{\kappa + 1/2} + 1}{\sqrt{\kappa + 1/2}} + 3$ . We note that there is a lower bound of  $(\sqrt{\kappa} + 1)^2$  in a related model [4], which holds in our model as well. For  $\kappa = 1$ , the competitive ratio of our algorithm is approximately 11.123, which improves upon the previous best-known upper bound of 11.656 due to [17]. The work in [17] also gives a lower bound of  $8 - \epsilon$  on the competitive ratio of any online scheduling algorithm for the continuous time model for  $\kappa = 1$ . This lower bound can be also transformed to our model. For the special case of unit length packets, we show an upper bound of 2 while a result in [1] implies a lower bound of the golden ratio  $\phi \approx 1.614$  for this case.

We also analyze the offline version of the problem. For the case of variable value packets, we give a 3-approximation algorithm. The algorithm is pseudo-polynomial and fits the general scheme introduced in [3]. For the case of unit value packets, we derive a 3-approximation algorithm which is a simplification of the above algorithm and has a polynomial running time. We note our problem is NP-Hard even for the case of unit value packets and  $N = 1$  (by a reduction from the Partition Problem [11]).

**Related work.** The present paper is most closely related to the work of Lee and Chwa [17], where they study the parallel communication problem assuming the uniform value density (i.e. the value of a packet is proportional to its length). They present a 2-competitive algorithm and establish a lower bound of 1.5 for the case of unit length jobs under the slotted model of time. For the case of arbitrary length jobs, they derive a 11.656-competitive algorithm and show a lower bound of  $8 - \epsilon$  under the continuous time model. In the present paper we extend their work to the case of variable value density, but in contrast to [17], we consider only the slotted time model. Some of our algorithms are in the spirit of [17], however, we try to maximize the total *value* rather than the total *length* of the successfully transmitted packets.

An enormous amount of research has been done on the single processor scheduling problem. Baruah et al. [4] and Koren and Shasha [16] considered the preemptive version of the real-time scheduling problem and gave upper bounds of 4 and  $(\sqrt{\kappa} + 1)^2$  for variable length jobs with uniform and non-uniform value density, respectively. Baruah et al. [4] presented matching lower bounds showing that these upper bounds are tight. Lipton and Tomkins [18] considered the non-preemptive version of the interval scheduling problem in which jobs have zero slack and uniform value density. This model was later generalized by Goldman et al. [12] to include delays. Dolev and Kesselman [7] studied non-preemptive scheduling of tasks with uniform value density. Garay et al. [10] considered the effect of job interleaving by preemption on the throughput. Differently from the above works, in our model time is slotted and each job (packet) requires two resources, namely the input port and the output port. We call this problem the *bipartite scheduling* problem.

The bipartite scheduling problem arises in many contexts. In a satellite switched time division multiple access (SS/TDMA) system, the goal is to schedule all traffic demands in a minimum number of time slots (i.e., minimize the makespan of the schedule). Gopal and Wong [13] proposed heuristic algorithms for this problem. Jain et al. [14] studied parallel scheduling of I/O tasks, where the set of disks must be matched to the set of I/O processors. In

contrast to these works, we consider *online* algorithms and try to satisfy the deadline constraints of *individual* packets.

**Organization** The rest of the paper is organized as follows. The model is described in Section 2. We study unit and variable value packets in Section 3 and Section 4, respectively. In Section 5 we consider the offline version of the problem. Due to space limitations, some proofs are omitted from this abstract.

## 2. MODEL DESCRIPTION

We consider an  $N \times N$  IQ switch (see Figure 1). Packets arrive at input ports, and each packet is labeled with the output port on which it has to leave the switch. A packet is divided into a number of unit length cells that must be transmitted contiguously. Time is slotted and during a time slot, up to one cell can be removed from each input and up to one cell can be transmitted to each output (note that multiple cells can be transmitted in parallel as long as they are transmitted along a matching between input ports and output ports). Next we introduce some useful definitions.

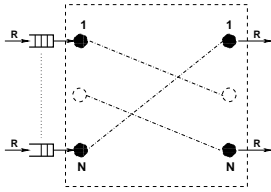


Figure 1: An example of IQ switch.

**DEFINITION 2.1.** For a packet  $p$ , we denote by  $l(p)$  the length of  $p$ , measured in cells, by  $w(p)$  the value of  $p$ , by  $r(p)$  the time  $p$  arrives to the switch, i.e., the release time of  $p$ , by  $d(p)$  the deadline of  $p$ , by  $in(p)$  the input port of  $p$  and by  $out(p)$  the output port of  $p$ . We denote by  $\rho(p) = w(p)/l(p)$  the value density of  $p$  and by  $\kappa$  the ratio of the maximal to the minimal value density. We denote by  $L$  the ratio between the length of the longest and that of the shortest packet. For a set of packets  $S$ , we denote by  $w(S)$  the total value of the packets in  $S$ .

**DEFINITION 2.2.** We define the remaining length of a packet  $p$  that is being transmitted at time  $t$  to be the number of  $p$ 's cells that have not yet been transferred to the output port.

Note that the scheduling problem of an IQ switch can be reduced to a matching problem in a bipartite graph, i.e., in each time slot the scheduler must compute a matching between the input and the output ports. We represent the state of a switch at a given time as an  $N \times N$  bipartite graph. The set of nodes  $V_I \cup V_O$  corresponds to the input and the output ports, and each packet  $p$  waiting in the queue of input  $i$  to be transferred to output  $j$  creates an edge  $e_p = (i, j)$ . (Note that a packet is present in the queue until its transmission is complete.)

The scheduler may *preempt* the transmission of a packet at any time, and possibly restart it later. However, the transmission of that packet must start from the beginning, and thus the time spent for the aborted transmission is wasted. A packet  $p$  is said to be *successfully transmitted* if it has been contiguously scheduled for  $l(p)$  time slots by time  $t = d(p)$ . The system accrues the value of a packet if the packet is successfully transmitted and gains no value otherwise. The aim of the scheduling algorithm is that of maximizing the total value of successfully transmitted packets.

**DEFINITION 2.3.** We define a transmission interval of a packet  $p$  to be a maximal continuous interval in which  $p$  is scheduled. We say that a packet is available at time  $t$  if its deadline is at least  $t + l(p) - 1$  and  $p$  has not yet been successfully transmitted. We also say that two packets conflict with each other if they share either the same input port or the same output port.

Note that a packet may have multiple transmission intervals (if its transmission is preempted and then restarted). W.l.o.g., we assume that an optimal offline algorithm *OPT* never preempts packets.

## 3. UNIT VALUE PACKETS

In this section we consider the case of unit value packets. Note that the goal of the scheduling policy is to maximize the total number of packets successfully transmitted. First we show a lower bound as a function of the maximum packet length,  $L$ , and the number of input/output ports,  $N$ . Thus, in contrast to the related problem of single processor scheduling, a constant competitive ratio is not achievable for our problem. Then we present two algorithms whose competitive ratios are given in terms of  $L$  and  $N$ .

### 3.1 Lower Bound

In this section we present a lower bound of  $K = \min\{\lceil \log L \rceil + 1, N\}$  on the competitive ratio of any deterministic online algorithm. Given a deterministic online algorithm *ALG*, the input sequence to *ALG* is created by the adversary *ADV* (see Figure 2). All packets generated by *ADV* have zero slack and thus each packet has a unique possible scheduling interval.

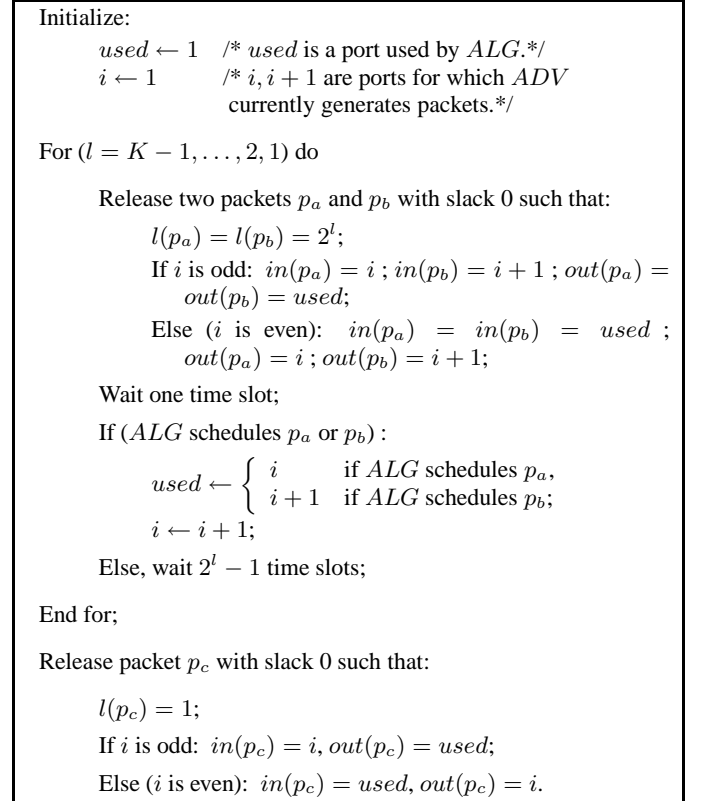
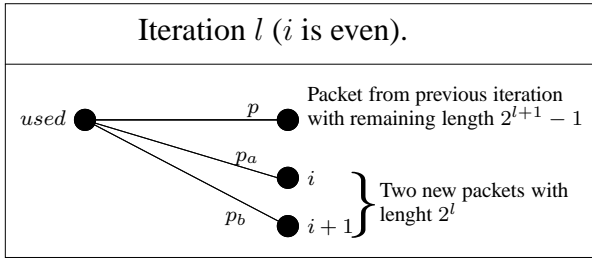


Figure 2: Adversary *ADV* for algorithm *ALG*.

*ADV* has two variables,  $used$  and  $i$ . If  $i$  is even (resp. odd) then the variable  $used$  holds the input (resp. output) port that is currently

used by *ALG* while  $i$  and  $i + 1$  are the output (resp. input) ports to which *ADV* generates the next packets. In each iteration, *ADV* generates two packets  $p_a$  and  $p_b$  with length (approximately) half the remaining length of the packet  $p$  currently transmitted by *ALG*. Packet  $p_a$  uses ports  $(used, i)$  and packet  $p_b$  uses ports  $(used, i + 1)$  (see Figure 3). Now all three packets  $p, p_a, p_b$  are conflicting at the port  $used$  and *ALG* can choose at most one of them. If *ALG* continues to schedule the longer packet  $p$ , *ADV* schedules to completion one of the short packets and then continues to the next iteration. If *ALG* schedules a short packet  $p_a$  (resp.  $p_b$ ) then *ADV* schedules  $p_b$  (resp.  $p_a$ ), updates  $used$  to be  $i$  (resp.  $i + 1$ ), increases  $i$  by one and then continues to the next iteration (one time step later). Thus, in each iteration *ADV* adds one packet to the set of packets that it can successfully transmit, while *ALG* can successfully transmit only one packet overall.



**Figure 3:** *ADV* - basic step

In what follows we refer to a particular iteration by the value of the variable  $l$  during this iteration. First we show that *ALG* can have at most one scheduled packet at any time and cannot successfully transmit any packet before *ADV* generates its last packet.

**LEMMA 3.1.** *At the end of any iteration  $l$ , *ALG* has not yet successfully transmitted any packet, and has at most one packet  $p$  that is being scheduled. If  $p$  exists then (1)  $p$  has a remaining length of  $2^l - 1$ , (2) if  $i$  is even then  $used = in(p)$ , (3) if  $i$  is odd then  $used = out(p)$ .*

**PROOF.** The proof is by reverse induction on  $l$ .

**Basis** ( $l = K - 1$ ). If *ALG* chooses not to schedule any packet, the claim holds. Otherwise, *ALG* schedules  $p_a$  or  $p_b$ , both of length  $2^{K-1} = 2^l$ . Denote the packet scheduled by *ALG* by  $p$ . By the definition of *ADV*, it waits one time unit, sets  $used = in(p)$ , increases  $i$  by one and the iteration ends. Thus, at the end of iteration  $K - 1$  the claim holds, since  $i = 2$ ,  $used = in(p)$  and  $p$  has remaining length of  $2^{K-1} - 1$ .

**Step :** Assume that the claim holds for iteration  $l$ . We prove that it holds for iteration  $l - 1$ . By the induction hypothesis, at the beginning of iteration  $l - 1$  (i.e., at the end of iteration  $l$ ) *ALG* has not yet successfully scheduled any packet, and *ALG* has at most one scheduled packet  $p$  and if  $p$  exists, it has remaining length of  $2^l - 1$ . W.l.o.g., assume that  $i$  is even at the beginning of iteration  $l - 1$ . We obtain that at the beginning of iteration  $l - 1$ , the new packets  $p_a$  and  $p_b$  require the input port  $used$ , and  $in(p) = used$ . There are two cases:

- *ALG* scheduled  $p_a$  (resp.  $p_b$ ). First we note that in this case *ALG* must preempt  $p$  (if it exists). Now *ADV* waits one time slot, updates  $used$  to be the output port

of  $p_a$  (resp.  $p_b$ ) and increments  $i$ . Therefore, at the end of iteration  $l - 1$ , *ALG* has not yet successfully scheduled any packet and the remaining length of  $p_a$  (resp.  $p_b$ ) is  $2^{l-1} - 1$ ,  $i$  is odd and  $used$  holds the output port of  $p_a$  (resp.  $p_b$ ), which yields the claim.

- *ALG* did not schedule either  $p_a$  or  $p_b$ . In this case *ADV* waits  $2^{l-1}$  time slots (first, one time slot and then another  $2^{l-1} - 1$  time slots). If *ALG* has a packet  $p$  scheduled at the beginning of iteration  $l - 1$ , then by the induction hypothesis at the end of iteration  $l - 1$ , *ALG* has not yet successfully scheduled any packet and  $p$  has remaining length of  $2^l - 1 - ((2^{l-1} - 1) + 1) = 2^{l-1} - 1$ . The claim holds for iteration  $l - 1$  since *ADV* does not change  $used$  and  $i$ . Note that if *ALG* does not have the packet  $p$  at the beginning of iteration  $l - 1$ , the claim holds trivially by the induction hypothesis.

□

Note that the packet generated by *ADV* in the last step,  $p_c$ , conflicts with the single packet that *ALG* may have (by Lemma 3.1) at the end of the last iteration. We get the following corollary.

**COROLLARY 3.2.** *ALG successfully transmits at most one packet out of the sequence generated by *ADV*.*

We now give a lower bound on the number of packets that can be scheduled by *OPT*. The next lemma demonstrates that *OPT* can gain at least one packet per iteration.

**LEMMA 3.3.** *Consider iteration  $l$  that ends at time slot  $t$ . There exists a set of packets  $S_l$  s.t.  $|S_l| = K - l$  and all packets in  $S_l$  can be successfully transmitted. If  $i$  is even (resp. odd) at the end of time slot  $t$ , then the input (resp. output) ports  $\{used\} \cup \{i + 1, i + 2, \dots, K\}$  and the output (reps. input) ports  $\{i, i + 1, \dots, K\}$  are not used by the packets in  $S_l$  after time slot  $t$  in this schedule.*

The proof of the above lemma proceeds by reverse induction on  $l$ . It is omitted from this abstract.

Note that from Lemma 3.3 it follows that there is a set  $S_1$  of  $K - 1$  packets that can be successfully transmitted without using ports  $i$  and  $used$  after the end of the last iteration. Thus, the packet  $p_c$  presented in the last step of *ADV* does not interfere with the packets in  $S_1$  and therefore the set  $S_1 \cup \{p_c\}$  has a feasible schedule and contains  $K$  packets. We obtain the following corollary.

**COROLLARY 3.4.** *OPT can successfully transmit  $K$  packets out of the sequence generated by *ADV*.*

We get that the competitive ratio of *ALG* is at least  $K$ . Note that  $K$  cannot be larger than  $N$  since during each iteration *ADV* may need to use a new port. On the other hand,  $K$  cannot be larger than  $\lfloor \log L \rfloor + 1$ , since in iteration  $K - 1$ , *ADV* generates a packet of length  $2^{K-1}$ .

**THEOREM 3.5.** *The competitive ratio of any deterministic on-line algorithm for the case of unit value packets is at least  $\min\{\lfloor \log L \rfloor + 1, N\}$ .*

### 3.2 A $(2^{2\sqrt{\log L}} + 1)$ -Competitive Algorithm

The Length-Sort and Schedule (*LSS*) algorithm is described in Figure 4. At each time slot, we assign to an edge  $e_p$  representing packet  $p$  a weight which is inversely proportional to  $p$ 's length. The parameter  $r$  specifies the preemption ratio. Specifically, we multiply the weight of a packet currently in transmission by a factor of

Each time slot  $t$  do the following:

Remove all packets that can no longer be fully scheduled by their deadlines.

Set  $S_t = \phi$ ;

Let  $w_t(e_p) = \begin{cases} r \frac{1}{l(p)} & \text{if } p \in S_{t-1}, \\ \frac{1}{l(p)} & \text{otherwise;} \end{cases}$

Scan the packets in order of non-increasing  $w_t(e_p)$ :

Add  $e_p$  to  $S_t$  if possible;

Schedule the packets in  $S_t$ .

**Figure 4: Algorithm  $LSS(r)$ .**

$r \geq 1$ . The intuition is that we try to avoid wasting time spent on preempted packets unless much shorter packets arrive.

We now analyze the performance of the  $LSS$  algorithm. In what follows we fix an input sequence  $\sigma$ . We consider the schedule of  $LSS$  and that of  $OPT$ . We denote by  $S^{LSS}$  and by  $S^{OPT}$  the set of packets successfully transmitted by  $LSS$  and  $OPT$ , respectively. We also denote by  $DROP = S^{OPT} \setminus S^{LSS}$  the set of packets successfully transmitted by  $OPT$  and lost by  $LSS$ .

Next we introduce a few useful definitions. We denote by  $P^{LSS}$  the set of all transmission intervals of packets transmitted by  $LSS$ . For each  $I \in P^{LSS}$ , denote by  $p_I$  the packet which is transmitted during  $I$ . For each  $p \in S^{OPT}$ , denote by  $I_p^{OPT}$  the transmission interval of  $p$  in  $OPT$ .

**OBSERVATION 3.6.** *If  $LSS(r)$  preempts a packet  $p$  scheduled during transmission interval  $I$  which ends at time slot  $t - 1$ , then there exists a packet  $q$  scheduled during transmission interval  $J \in P^{LSS}$  which begins at time slot  $t$  such that: (1)  $p$  conflicts with  $q$ , and (2)  $l(q) \leq l(p)/r$ .*

For each packet  $p$  and transmission interval  $I$  of  $p$  which does not end successfully, we denote by  $preempt(p, I)$  the transmission interval guaranteed by Observation 3.6. Note that there may be more than one such intervals and we arbitrarily choose one of them.

**OBSERVATION 3.7.** *For every  $p \in DROP$ , there is a transmission interval  $J \in P^{LSS}$  of a packet  $q$  with the following properties: (1)  $J \cap I_p^{OPT} \neq \phi$ , (2)  $p$  conflicts with  $q$ , and (3)  $l(q)/r \leq l(p)$ .*

Note that if  $LSS(r)$  started a transmission of  $p$  and then preempted it in favor of a packet  $q$  then  $r/l(p) \leq 1/l(q)$  and since  $1 \leq r$  we have that  $l(q)/r \leq r/l(p) \leq l(p)$ .

For each  $p \in DROP$ , we denote by  $block(p)$  the transmission interval guaranteed by Observation 3.7. Observe that there may be more than one such interval and we arbitrarily select one of them.

Now we define the *blame forest*  $F$ . The idea is to build trees in which a vertex is the transmission interval of a packet in  $P^{LSS}$  or in  $DROP$ . The root of each tree in  $F$  is the transmission interval of a packet in  $S^{LSS}$ . To derive the competitive ratio of  $LSS$ , we will bound the number of packets from  $DROP$  in such a tree. More formally, the blame forest  $F = (V, E)$  has the set of nodes  $V = \{v_I | I \in P^{LSS}\} \cup \{v_p | p \in DROP\}$  and the set of edges  $E = \{(v_p, v_I) | I = block(p)\} \cup \{(v_I, v_J) | J = preempt(p, I)\}$ . For each edge  $(v_I, v_J)$  s.t.  $J = preempt(p, I)$ , we say that  $v_J$  is the parent of  $v_I$  (or  $v_I$  is a child of  $v_J$ ). For each edge  $(v_p, v_I)$  s.t.  $J = block(p)$  we say that  $v_I$  blocks  $v_p$ .

**CLAIM 3.8.**  $F = (V, E)$  is a directed forest.

**PROOF.** Since the outdegree of all vertices is at most 1, we need to show that there are no cycles in  $F$ . Assume towards a contradiction that there is a cycle  $v_1, v_2, \dots, v_k, v_1$  in  $F$ . By the definition of  $E$  and  $block(p)$ , we have that  $\forall p \in DROP$ ,  $v_p$  is a leaf and therefore the cycle contains only vertices of the form  $v_I$  s.t.  $I \in P^{LSS}$ . Let  $I_1, I_2, \dots, I_k$  be the intervals in  $P^{LSS}$  corresponding to the vertices  $v_1, v_2, \dots, v_k$ , respectively. Since the outdegree of all vertices is one, the cycle must be directed. Let  $1 \leq j \leq k$  be index such that  $l(p_{I_j})$  is minimal. But by the definition of  $E$ , the existence of edge  $(I_{j-1}, I_j)$ , implies that  $I_j = preempt(p_{I_{j-1}}, I_{j-1})$ . Observation 3.6 implies that  $l(p_{I_{j-1}}) \leq l(p_{I_j})/r$ , which contradicts to the minimality of  $l(p_{I_j})$ .  $\square$

**CLAIM 3.9.**  $\forall I \in P^{LSS}$ : (1) there are at most two different intervals  $J, J'$  preempted by  $I$  and (2) there are at most  $2r$  packets in  $DROP$  blocked by  $I$ .

**PROOF.** By Observation 3.6, the intervals preempted by  $I$  must end exactly at the time slot before  $I$  starts. They also must share at least one port with  $p_I$ . Therefore,  $I$  preempts at most two intervals.

By Observation 3.7, each packet  $q$  blocked by  $I$  must have length of at least  $l(q)/r$ . Moreover, all packets blocked by  $I$  must share at least one port with  $p_I$ . Therefore,  $I$  blocks at most  $2r$  packets.  $\square$

**LEMMA 3.10.** *The number of vertices  $v_p$  s.t.  $p \in DROP$  in each tree  $T \subseteq F$  is at most  $2rL^{\frac{1}{\log(r)}}$ .*

**PROOF.** Let  $T$  be a tree in  $F$ . Recall that  $T$  contains vertices  $v_I$  s.t.  $I \in P^{LSS}$  and vertices  $v_p$  s.t.  $p \in DROP$ . By the definition of  $E$ , every vertex  $v_p$  is a leaf in  $T$ . Let us consider the sub-tree  $T' \subseteq T$  that contains only vertices of the form  $v_I$ . By Claim 3.9, each vertex  $v_I$  has at most two children and by the definition of  $E$  and  $preempt(p_I, I)$ ,  $v_I$  has at most one parent. According to Observation 3.6, if  $v_I$  is the parent of  $v_J$  then  $l(p_I) \leq l(p_J)/r$ . Thus, the depth of  $T'$  is at most  $\log_r L$  and the size of  $T'$  is at most  $2^{\log_r L} = L^{\frac{1}{\log(r)}}$ . Claim 3.9 implies that each vertex in  $T'$  blocks at most  $2r$  vertices  $v_p$  and therefore the number of vertices  $v_p$  s.t.  $p \in DROP$  in  $T$  is at most  $2rL^{\frac{1}{\log(r)}}$ .  $\square$

**THEOREM 3.11.**  $LSS(r)$  is  $(2rL^{\frac{1}{\log(r)}} + 1)$ -competitive.

**PROOF.** Note that the root of each tree  $T \in F$  corresponds to a packet transmitted successfully by  $LSS$ . By the definition of  $F$ , for each packet in  $DROP$  there is a vertex in  $F$ . Obviously,  $|S^{OPT}| \leq |DROP| + |S^{LSS}|$ , and by Lemma 3.10  $|DROP| \leq 2rL^{\frac{1}{\log(r)}} |S^{LSS}|$ . Thus we obtain that

$$|S^{OPT}| \leq (1 + 2rL^{\frac{1}{\log(r)}}) |S^{LSS}|.$$

$\square$

For a given  $L$ , we can optimize the value of  $r$ . We obtain that the competitive ratio of  $LSS(r)$  is minimized when  $r = 2^{\sqrt{\log L}}$ .

**COROLLARY 3.12.**  $LSS(2^{\sqrt{\log L}})$  is  $(2^{2\sqrt{\log L}+1} + 1)$ -competitive.

### 3.3 A $(N + 1)$ -Competitive Algorithm

The Shortest Remaining Time First ( $SRTF$ ) algorithm is presented in Figure 5. The  $SRTF$  algorithm always gives priority to packets with the closest completion time. The idea behind the proof is that every packet successfully transmitted by  $OPT$  and lost by  $SRTF$  must share the last time slot of its transmission interval in  $OPT$  with some packet  $p$  successfully transmitted by  $SRTF$ . Hence,  $OPT$  can schedule at most  $N$  packets per every packet scheduled by  $SRTF$ .

The proof of the next theorem is omitted from this abstract.

Each time slot  $t$  do the following:

Remove all packets that can no longer be fully scheduled by their deadlines.

Set  $S_t = \phi$ .

Scan the packets in order of non-increasing remaining length and add them to  $S_t$  if possible.

Schedule the packets given in  $S_t$ .

**Figure 5: Algorithm SRTF.**

THEOREM 3.13. *SRTF is  $(N + 1)$ -competitive.*

### 3.4 Lower Bound for Length-Oriented Algorithms

In this section we define a class of algorithms that generalizes the algorithms given in this paper. We give a lower bound for those algorithms which is stronger than the general lower bound proved above for the case of unit value packets. We characterize those algorithms in the following way. In each time slot the algorithm assigns a weight for every packet and determines a schedule for the next time slot by selecting packets greedily or by computing a maximum matching based on the weight function.

DEFINITION 3.1. *Let ALG be an online deterministic algorithm. ALG is length-oriented if it works in the general scheme described in Figure 6 and for each packet  $p$ , the weight  $w_p$  is a function of  $l(p)$  and/or the remaining length of  $p$ . The weight function is a non-increasing function of the remaining length (i.e., the weight of a packet cannot decrease while it is being transmitted).*

Each time slot  $t$  do the following:

Remove all packets that can no longer be fully scheduled by their deadlines.

Assigns a weight  $w_p$  to every packet  $p$ .

Compute a schedule for the next time slot as a maximum weight matching or as a greedy selection of edges.

**Figure 6: Length-Oriented Algorithms, general scheme.**

We start by showing some properties of length-oriented algorithms.

LEMMA 3.14. *Let ALG be a  $c$ -competitive length-oriented algorithm. Let  $r = \lfloor 2c + 1 \rfloor$ . Suppose that there are three packets  $p, q, s$  eligible for scheduling such that:*

- the remaining length of  $p$  and  $q$  is  $\ell$ ,
- $l(s) = \lfloor \ell/r \rfloor$ ,
- $in(s) = in(p) \neq in(q)$ ,
- $out(s) = out(q) \neq out(p)$ ,
- there are no other packets conflicting with  $p, q$  and  $s$ .

Then ALG schedules  $s$ , and rejects  $p$  and  $q$ .

PROOF. Assume that ALG does not schedule  $s$ . In this case it must schedule  $p$  and  $q$ . Consider the following scenario. In the next time slot arrives a set of packets  $s_2, s_3, \dots, s_r$  such that  $\forall i = 2, \dots, r$   $in(s_i) = in(s)$ ,  $out(s_i) = out(s)$ ,  $l(s_i) = l(s)$  and  $d(s_i) = \ell$ . Note that all these packets conflict with  $p$  and  $q$ . Since ALG is length-oriented, the weight of  $p$  and  $q$  can only increase over time and therefore ALG continues to schedule  $p$  and  $q$ . Now observe that it is possible to schedule  $s$  and  $s_2, \dots, s_r$  sequentially because  $r \lfloor \ell/r \rfloor \leq \ell$ . Thus, OPT schedules  $r$  packets, namely  $s$  and  $s_2, \dots, s_r$ , while ALG schedules only two packets, namely  $p$  and  $q$ . Therefore, ALG has a competitive ratio of at least  $r/2 = \lfloor 2c + 1 \rfloor / 2 > c$ , which contradicts our assumption.  $\square$

THEOREM 3.15. *The competitive ratio of any length oriented algorithm for the case of unit value packets is at least*

$$\min \left\{ \frac{1}{2} (2^{\sqrt{\log L}} - 1), N \right\}.$$

PROOF. Let ALG be a  $c$ -competitive length oriented algorithm and let  $r = \lfloor 2c + 1 \rfloor$ . Let  $k$  be the largest integer such that  $2^k \leq N$  and  $\sum_{i=0}^k r^i \leq L$ . For  $0 \leq i \leq k$  we define set  $S^i$  of packets:

$$S^i = \left\{ p_j^i \mid 0 \leq j \leq 2^{k-i} - 1 \right\}.$$

The parameters of packet  $p_j^i$  are defined as follows:

$l(p_j^i) = \sum_{t=0}^i r^t$ ,  $in(p_j^i) = 1 + j2^i$ ,  $out(p_j^i) = (j + 1)2^i$  and  $r(p_j^i) = i$ . All packets have zero slack and thus we do not specify their deadlines.

At time  $t = 0$ , ALG will schedule all packets from  $S^0$ . Note that every packet  $p_j^i$  in the set  $S^i$  s.t.  $1 \leq i \leq k$ , conflicts with exactly two packets in the set  $S^{i-1}$ , namely  $p_{2j-1}^{i-1}$  and  $p_{2j}^{i-1}$ . In addition, packets  $p_{2j-1}^{i-1}$ ,  $p_{2j}^{i-1}$  and  $p_j^i$  satisfy the condition of Lemma 3.14 as the packets  $p, q$  and  $s$ , respectively. Therefore, at time  $t = i$ , ALG schedules the packets in  $S^i$  and preempts the packets from  $S^{i-1}$ . In this way, ALG successfully transmits only one packet  $p_1^k$  overall. On the other hand, OPT schedules  $2^k$  packets, namely all the packets from  $S^k$ . We get that the competitive ratio of ALG is at least  $2^k$ .

If the value of  $k$  was determined by the constraint  $2^k \leq N$  then  $c \geq 2^k = N$ . If the value of  $k$  was determined by the constraint  $L \geq \sum_{i=0}^k r^i$  then, since  $k$  is the largest integer for which the constraint holds, we get that  $k = \lfloor \log_r(L(r-1) + 1) \rfloor - 1 \geq \log_r(L) - 1$ . Hence,  $2^k \geq \frac{1}{2} L^{\frac{1}{\log(r)}}$ . Since ALG is  $c$ -competitive, it must be the case that  $2^k \leq c$  and thus  $\frac{1}{2} L^{\frac{1}{\log(r)}} \leq c$ . Therefore,  $\log L \leq \log(r)(1 + \log(c))$ . Assigning  $r = \lfloor 2c + 1 \rfloor$  gives  $\log L \leq \log(2c + 1)(1 + \log(c))$ , and  $\log L \leq \log^2(2c + 1)$ . Finally, we obtain:

$$\frac{1}{2} 2^{\sqrt{\log L} - 1} \leq c.$$

Therefore,

$$c \geq \min \left\{ \frac{1}{2} (2^{\sqrt{\log L}} - 1), N \right\}.$$

$\square$

## 4. VARIABLE VALUE PACKETS

In this section we consider the case of variable value packets. Thus, the goal of the scheduling policy is to maximize the total value of the packets successfully transmitted.

In Figure 7 we describe the preemptive maximum weight matching (PMV) algorithm. Each edge representing a packet is assigned

a weight equal to the value of the packet. The parameter  $r$  specifies the preemption ratio. Namely, we multiply the weight of any edge representing a packet currently in transmission by a factor of  $r$ . The intuition is that we allow the preemption of a packet transmission only if ‘significantly more valuable packets’ can be scheduled after the packet is preempted, thus justifying the time wasted for the aborted transmission.

Each time slot  $t$  do the following:

Remove all packets that can no longer be fully scheduled by their deadlines.

Let  $w_t(e_p) = \begin{cases} r \cdot w(p) & \text{if } p \in M_{t-1}, \\ w(p) & \text{otherwise.} \end{cases}$

Compute a maximum weight matching  $M_t$ .

Schedule the packets corresponding to the edges in  $M_t$ .

**Figure 7: The  $PMV(r)$  algorithm for variable length packets.**

Next we analyze the performance of the  $PMV$  algorithm. In what follows we fix an input sequence  $\sigma$  and let the latest deadline of a packet in  $\sigma$  be  $d_l$ . We consider the schedule of  $PMV$  and  $OPT$ . We denote by  $S^{PMV}$  and by  $S^{OPT}$  the set of packets successfully transmitted by  $PMV$  and  $OPT$ , respectively. We also denote by  $DROP = S^{OPT} \setminus S^{PMV}$  the set of packets successfully transmitted by  $OPT$  and lost by  $PMV$ . Let  $DROP_t$  be the set of packets from  $DROP$  that  $OPT$  starts to schedule at time  $t$ .

The following claim states that when  $OPT$  starts to schedule a packet  $p$  from  $DROP_t$ , some packets with non-negligible value that conflict with  $p$  are being scheduled by  $PMV$ .

**CLAIM 4.1.** *Consider a packet  $p \in DROP_t$  and let  $S$  be the set of packets conflicting with  $p$  that are scheduled by  $PMV$  at time  $t$ . We have that  $rw(S) \geq w(p)$ .*

**PROOF.** Observe that  $p$  is available to  $PMV$  at time  $t$ . If  $p$  itself is scheduled at time  $t$ , we are done. Otherwise, suppose towards a contradiction that  $rw(S) < w(p)$ . In this case the weight of  $M_t$  can be increased by removing the edges corresponding to packets in  $S$  and adding  $e_p$ . That contradicts with the fact that  $PMV$  computes a maximum weight matching.  $\square$

Note that the packet  $p$  may be itself in the set  $S$ . In the next claim we consider the situation in which  $PMV$  preempts a packet transmission.

**CLAIM 4.2.** *Consider the set of packets  $R$  preempted by  $PMV$  at time  $t$  and let  $S$  be the set of new packets that  $PMV$  starts to schedule at time  $t$ . We have that  $w(S) \geq r \cdot w(R)$ .*

**PROOF.** Suppose towards a contradiction that  $w(S) < r \cdot w(R)$ . We argue that in this case the weight of  $M_t$  can be increased by removing the edges corresponding to packets in  $S$  and adding the edges corresponding to packets in  $R$ , which contradicts the maximality of the matching computed by  $PMV$ . Note that packets in  $R$  do not conflict with the rest of the packets that have been scheduled by  $PMV$  at time  $t - 1$ .  $\square$

We now introduce the following definitions.

**DEFINITION 4.1.** *Consider the transmission interval  $I$  of an  $OPT$  packet  $p$ . For a time slot  $t \in I$ , let  $p_i(p, t)$  be the packet*

*conflicting with  $p$  at the input that is scheduled by  $PMV$  at time  $t$  if any, or a dummy packet with zero value otherwise. Similarly, we define  $p_o(p, t)$  w.r.t. the output of  $p$ .*

**DEFINITION 4.2.** *Consider the transmission interval  $I$  of an  $OPT$  packet  $p$  and a transmission interval  $I'$  of a  $PMV$  packet  $p'$  conflicting with  $p$  at the input. We define the overlapping input transmission interval of  $I$  and  $I'$  to be their intersection, if any, or an empty interval otherwise. Similarly, we define the overlapping output transmission interval w.r.t. the output of  $p$ .*

**DEFINITION 4.3.** *Consider a sub-interval  $I'$  of the transmission interval  $I$  of an  $OPT$  packet  $p$  and let the length of  $I'$  be  $\ell$ . We say that  $OPT$  gains the value of  $\ell \cdot \rho(p)$  on  $I'$ .*

We will show that the competitive ratio of  $PMV$  is at most  $2r + 2\kappa + \frac{r+2\kappa}{r-1} + 1$  (recall that  $r$  is the preemption factor and  $\kappa$  is the importance ratio). We will assign the value of all packets in  $DROP$  to the packets successfully transmitted by  $PMV$  so that each  $PMV$  packet is assigned at most  $2r + 2\kappa + \frac{r+2\kappa}{r-1}$  times its value, and show that such an assignment is feasible.

The assignment routine is described in Figure 8. Consider a packet  $p \in DROP_t$  and let  $I$  be its transmission interval. (Remember that by our assumption each  $OPT$  packet has a unique transmission interval.) At Sub-Step 1(a) we assign the value gained by  $OPT$  on all sub-intervals of  $I$  which overlap at the input or at the output with a transmission interval of  $PMV$ ; at Sub-Step 1(b) we assign the rest of the value gained by  $OPT$  on  $I$ ; at Step 2 we re-assign the value currently assigned to the packets preempted by  $PMV$ . Note that at iteration  $t$ , we may assign some value to packets scheduled by  $PMV$  at time  $t' > t$ .

The following claims demonstrate that only packets that are successfully transmitted by  $PMV$  are assigned some value and that the assignment routine is feasible and the total value assigned is at least  $w(DROP)$ .

**CLAIM 4.3.** *After the assignment routine finishes, the total assigned value is assigned to packets that are successfully transmitted by  $MPV$ .*

**PROOF.** Note that at iteration  $t$  of the assignment routine, only packets that are scheduled by  $PMV$  at time  $t' \geq t$  are considered for assignment. In case a packet is preempted at some time  $t^*$ , all the value that has been assigned to it due to its current transmission interval is re-assigned at  $t^*$  when Step 2 of the assignment routine is executed. This value is assigned to new packets that  $PMV$  starts to schedule at  $t^*$ . Thus, when the assignment routine finishes, no value is assigned to packets that are not successfully scheduled. By the finiteness of the input sequence, the total assigned value is assigned to packets that are eventually successfully transmitted by  $MPV$ .  $\square$

**CLAIM 4.4.** *The assignment routine is feasible and after it finishes, the total value assigned is at least  $w(DROP)$ .*

**PROOF.** Observe that the value of any packet  $p$  in  $DROP_t$  is processed by the assignment routine at iteration  $t$ . Since  $p$  is available to  $PMV$  at time  $t$ , we have that  $w(p_i(p, t)) + w(p_o(p, t)) > 0$ . Thus, the assignment is well-defined and the value of  $p$  will be fully assigned by steps 1(a) and 1(b) of the assignment routine.  $\square$

The following lemma establishes an upper bound on the total value that can be assigned to a  $PMV$  packet that is successfully transmitted.

**For**  $t = 0$  up to  $d_i$  **Do**:

1. **For** each packet  $p$  from  $DROP_t$  **Do**:

Let  $I = [t_s = t, t_f]$  be the transmission interval of  $p$  in  $OPT$ .

(a) **For**  $t' = t_s$  up to  $t_f$  **Do**:

If  $w(p_i(p, t')) + w(p_o(p, t')) > 0$  then assign  $\rho(p) \frac{w(p_i(p, t'))}{w(p_i(p, t')) + w(p_o(p, t'))}$  to  $p_i(p, t')$  and the value of  $\rho(p) \frac{w(p_o(p, t'))}{w(p_i(p, t')) + w(p_o(p, t'))}$  to  $p_o(p, t')$ .

(b) If  $p_i(p, t)$  is successfully transmitted, let  $\hat{p}_i(p, t)$  be  $p_i(p, t)$ . Else, if  $p_i(p, t)$  is preempted, let  $\hat{p}_i(p, t)$  be the last packet in the sequence of preempting packets with value of at least  $w(p_i(p, t))$  whose overlapping input transmission intervals with  $I$  are *consecutive* and *non-empty*. In a similar way, we define  $\hat{p}_o(p, t)$  w.r.t. to the output of  $p$ . Let the value of  $p$  minus the value already assigned at Sub-Step 1(a) be  $v$ . If  $v > 0$  then assign the value of  $v \frac{w(p_i(p, t))}{w(p_i(p, t)) + w(p_o(p, t))}$  to  $\hat{p}_i(p, t)$  and the value of  $v \frac{w(p_o(p, t))}{w(p_i(p, t)) + w(p_o(p, t))}$  to  $\hat{p}_o(p, t)$ .

2. Consider the set of packets  $R$  preempted by  $PMV$  at time  $t$  and the set  $S$  of new packets that  $PMV$  starts to schedule at this time. Let  $v$  be the total value assigned to packets in  $R$  due to their current transmission intervals (\*) and let  $v'$  be the total value of packets in  $S$ . Assign to each packet  $p$  in  $S$  the value of  $w(p) \cdot v/v'$ .

(\*) For a transmission interval  $I$  of a  $PMV$  packet  $p$ , the value assigned to  $p$  due to  $I$  is the total value assigned to  $p$  in the context of a time slot  $\hat{t} \in I$ .

**Figure 8: The assignment routine for variable length packets.**

LEMMA 4.5. *After the assignment routine finishes, no  $PMV$  packet that is successfully transmitted is assigned more than  $2r + 2\kappa + \frac{r+2\kappa}{r-1}$  times its value for any  $r \geq 2$ .*

PROOF. Consider a transmission interval  $I = [t_s, t_f]$  of a  $PMV$  packet  $p$ . Let  $v$  be the value assigned to  $p$  due to  $I$  by Step 1. We will show that if  $p$  is successfully transmitted during  $I$  then  $v$  is bounded by  $(2r+2\kappa) \cdot w(p)$  and if  $p$  is preempted then  $v$  is bounded by  $(r+2\kappa) \cdot w(p)$ .

Clearly, the total value assigned to  $p$  by Sub-Step 1(a) is bounded by  $2\kappa \cdot w(p)$  since the sum of the lengths of the overlapping input and output transmission intervals of  $I$  with that of packets in  $DROP$  is bounded by  $2l(p)$ . (Note that transmission intervals of packets in  $DROP$  conflicting with each other cannot overlap.)

Now let us consider Sub-Step 1(b). We claim that  $p$  can be assigned fully or partially the value of at most two packets from  $DROP$  conflicting with  $p$  at the input and at the output, namely, the two packets of  $DROP$  whose transmission intervals overlap with the last time slot of  $I$ . To see that, consider a packet  $\hat{p}$  from  $DROP$  conflicting with  $p$  at the input whose value has been assigned to  $p$  by Sub-Step 1(b). Let  $\hat{I} = [\hat{t}_s, \hat{t}_f]$  be the transmission interval of  $\hat{p}$  in  $OPT$ . Suppose towards a contradiction that  $\hat{I}$  does not overlap with the last time slot of  $I$ . Obviously, if  $\hat{t}_s > t_f$ , the value of  $\hat{p}$  cannot be assigned to  $p$  since  $I$  and  $\hat{I}$  do not overlap. Thus, assume that  $\hat{t}_s \leq t_f$ . If  $\hat{t}_s \geq t_s$  then the value of  $\hat{p}$  is com-

pletely assigned by Sub-Step 1(a) since  $\hat{I}$  is contained in  $I$ . In case  $\hat{t}_s < t_s$ , the only way that the value of  $\hat{p}$  can be assigned to  $p$  by Sub-Step 1(b) is that  $p$  is the last packet in the sequence of preempting packets whose overlapping input transmission intervals with  $\hat{I}$  are consecutive and non-empty. However, in this case the value of  $\hat{p}$  is still completely assigned by Sub-Step 1(a) because  $\hat{I}$  is entirely covered by the transmission intervals of the  $PMV$  packets in this sequence. We get a contradiction to our assumption. Therefore,  $\hat{I}$  overlaps with the last time slot of  $I$ .

Let  $p'$  be the  $OPT$  packet whose transmission interval  $I' = [t'_s, t'_f]$  overlaps with the last time slot of  $I$  at the input, if any, or a dummy packet with zero value otherwise. Similarly, we define  $p''$  and  $I''$  w.r.t. the output of  $p$ .

By Claim 4.1, the total value of packets conflicting with  $p'$  at time  $t'_s$  when  $OPT$  starts to schedule it is at least  $w(p')/r$ . It follows that if  $p$  does not take part in a sequence of preempting packets (in this case  $t'_s \geq t_s$ ), it can be assigned at most  $r$  times its value when  $p'$  is processed by Sub-Step 1(b). If  $p$  is the last packet in a sequence of preempting packets (in this case  $t'_s < t_s$ ), by the definition of Sub-Step 1(b), the value of  $p$  is greater than or equal to that of the packet scheduled by  $PMV$  at the input of  $p$  at time  $t'_s$ . Hence,  $p$  can still be assigned at most  $r$  times its value when  $p'$  is processed by Sub-Step 1(b). A similar analysis can be done for  $p''$ . Therefore, in case  $p$  is successfully transmitted during  $I$ , it is assigned at most  $2r$  times its value by Sub-Step 1(b).

If  $p$  is preempted, at least one of the new packets conflicting with  $p$  that  $PMV$  starts to schedule at this time has value greater than or equal to  $w(p)$ ; denote this packet by  $\hat{p}'$ . To see that, observe that otherwise the weight of the matching scheduled by  $MPV$  can be increased by removing the edges corresponding to the new packets conflicting with  $p$  and inserting back  $e_p$ , since by our assumption  $r \geq 2$ . By the definition of Sub-Step 1(b), the unassigned value of either  $p'$  or  $p''$  will be assigned to  $\hat{p}'$  (or maybe to another packet that preempts  $\hat{p}'$ ). Thus,  $p$  can be assigned by Sub-Step 1(b) only the value of one  $OPT$  packet. This value is at most  $r$  times its own value, as we argued in the previous case.

Now let us proceed to Step 2 of the assignment routine. Suppose that  $p$  is successfully transmitted during  $I$ . Note that  $I$  is considered by Step 2 only once, namely at time  $t_s$ . Let  $R$  be the set of packets preempted by  $PMV$  and let  $S$  be the set of new packets that  $PMV$  starts to schedule at time  $t_s$ . By Claim 4.2,  $w(S) \geq r \cdot w(R)$ . If any packet in  $R$  has been assigned at most  $f$  times its value due to its current transmission interval then by the definition of Step 2, any packet in  $S$  is assigned at most  $f/r$  times its value. Since initially any preempted packet is assigned at most  $r + 2\kappa$  times its value by Step 1, we obtain a geometric progression whose sum is bounded by  $\frac{r+2\kappa}{r-1}$ , which yields the lemma.  $\square$

Now we are ready to prove the main theorem.

THEOREM 4.6. *The competitive ration of  $PMV(r)$  is at most  $(2r + 2\kappa + \frac{r+2\kappa}{r-1} + 1)$ .*

PROOF. Obviously,  $w(S^{OPT}) \leq w(DROP) + w(S^{PMV})$ . Lemma 4.5, Claim 4.3 and Claim 4.4 imply that  $w(DROP) \leq (2r + 2\kappa + \frac{r+2\kappa}{r-1}) \cdot w(S^{PMV})$ . We obtain that  $w(S^{OPT}) \leq (2r + 2\kappa + \frac{r+2\kappa}{r-1} + 1) \cdot w(S^{PMV})$ , which establishes the theorem.  $\square$

For a given  $\kappa$ , we optimize the value of  $r$  obtaining that the competitive ratio of  $PMV$  is minimized when  $r = 1 + \sqrt{\kappa + 1/2}$ . Observe that  $r \geq 2$  since  $\kappa \geq 1$  and the condition of Lemma 4.5 is satisfied.

COROLLARY 4.7. *The competitive ratio of  $PMV(1 + \sqrt{\kappa + 1/2})$  is at most  $2\kappa + 2\sqrt{\kappa + 1/2} + \frac{2\kappa + \sqrt{\kappa + 1/2} + 1}{\sqrt{\kappa + 1/2}} + 3$ .*

We note that for  $\kappa = 1$  we get a competitive ratio of approximately 11.123, which improves upon the results of [17]. We also observe that for the case of unit length packets  $PMV(1)$  is 2-competitive. Observe that in this case packets are not preempted, which significantly simplifies the analysis.

**THEOREM 4.8.** *The competitive ratio of  $PMV(1)$  is at most 2 for the case of unit length packets.*

## 5. OFFLINE ALGORITHMS

In this section we consider the offline version of our problem. The problem is NP-hard even for the special case of unit value packets and  $N = 1$ , which is also known as the *sequencing within intervals* problem [11]. We present a pseudo-polynomial 3-approximation algorithm for the general case. This algorithm falls into the general framework introduced in [3]. We note that [3] also gives a  $\frac{3}{1-\epsilon}$ -approximation algorithm with running time polynomial in  $1/\epsilon$ . For the case of unit value packets, we derive a 3-approximation polynomial time algorithm based on our algorithm for the general case.

### 5.1 Variable Value Packets

The input to our algorithm is a set of packets  $S$ . To define our algorithm we first define a new set of packets  $P$ . For each packet  $p \in S$ , we define a set of packets  $dup(p)$  that includes zero slack packets corresponding to all possibilities to schedule  $p$ . Specifically

$$dup(p) = \bigcup_{i=0}^{s(p)} \{q_i | r(q_i) = r(p) + i, d(q_i) = r(q_i) + l(q_i)\}.$$

Where  $s(p)$  is the slack of a packet and is defined as  $s(p) = d(p) - l(p) - r(p) + 1$ . The new set  $P$  is defined as  $P = \bigcup_{p \in S} dup(p)$ .

Note that the the number of packets in  $P$  depends on the slack of the packets in  $S$  and thus is pseudo-polynomial in the input  $S$ .

The algorithm  $SCHED(P, w)$  is described in Figure 9. Since all packets in  $P$  have zero slack, for every  $p \in P$  there is exactly one possible transmission interval of  $p$ ,  $I_p$ . If the algorithm  $SCHED(P, w)$  selects some packet  $p \in P$  to be scheduled, this means that in the actual solution the packet  $q \in S$  s.t.  $p \in dup(q)$  will be scheduled on the transmission interval  $I_p$ .

We define the set  $C(p)$  to contain all the packets in  $P$  that cannot be scheduled if  $p \in P$  is scheduled (including  $p$  itself). There are two types of such packets, packets that are in  $dup(p)$  and are actually just duplications of the original packet from  $S$ , and packets that are conflicting with  $p$  and share some time slot with  $I_p$ . More formally, we define  $C(p)$  as follows:

$$C(p) = \{q \in P | I_p \cap I_q \neq \emptyset \text{ and } q \text{ conflicts with } p\} \cup dup(p).$$

With slight abuse of notation,  $dup(p)$  for  $p \in P$  means  $dup(s)$  such that  $s \in S$  and  $p \in dup(s)$ .

Next we analyze the performance of our algorithm.

**LEMMA 5.1.** *Consider a set  $P$  of packets with zero slack and an arbitrary weight function  $w$ . Let  $p$  be a packet with the earliest deadline selected by  $SCHED(P, w)$ . Then a feasible solution that includes  $p$  and a feasible solution to which  $p$  cannot be added, are both 3-approximation w.r.t. the weight function  $w_1$  (as defined in Figure 9).*

**PROOF.** Observe that only packets in  $C(p)$  contribute positive weight to the solution. Since  $p$  has the earliest deadline and all packets have zero slack, each packet  $q \in C(p) \setminus dup(p)$  conflicts with  $p$  and its transmission interval  $I_q$  contains the last time slot of  $p$ 's transmission interval  $I_p$ . Therefore, any feasible solution w.r.t. the weight function  $w_1$  can include at most three packets from

```

Remove all packets  $p$  with  $w(p) \leq 0$  from  $P$ ;

If  $P = \emptyset$  return  $\phi$ ;

Select a packet  $p$  in  $P$  with the minimal  $d(p)$ .

Let  $w_1(x) = \begin{cases} w(p) & \forall x \in C(p), \\ 0 & \text{otherwise;} \end{cases}$ 

Let  $w_2(x) = w(x) - w_1(x)$ ;

 $T = SCHED(P, w_2)$ ;

If  $T \cup \{p\}$  is a feasible solution then  $R = T \cup \{p\}$ ;

Else,  $R = T$ ;

Return  $R$ .

```

**Figure 9:** Algorithm  $SCHED(P, w)$ .

$C(p)$ : at most one packet from  $dup(p)$ , and at most two packets from  $C(p) \setminus dup(p)$ . Thus, the weight of an optimal solution w.r.t. the weight function  $w_1$  is at most  $3w(p)$ . If  $p$  is in the solution, then the weight of that solution is at least  $w(p)$ . If  $p$  cannot be added to the solution, then there is a packet  $p' \in C(p)$  which is in the solution. However,  $w_1(p') = w(p)$ . We obtain that both solutions are 3-approximation.  $\square$

We are now ready to prove the main theorem.

**THEOREM 5.2.** *The algorithm  $SCHED(P, w)$  is a 3-approximation w.r.t. the weight function  $w$ .*

**PROOF.** The proof is by induction on the recursion depth.

**Basis:** Trivial.

**Step:** Denote by  $OPT_1$  an optimal solution w.r.t. the weight function  $w_1$  and by  $OPT_2$  an optimal solution w.r.t. the weight function  $w_2$ . Note that the solution returned by the algorithm,  $R$ , falls within one of the two categories considered in Lemma 5.1 and therefore  $w_1(R) \geq 3w_1(OPT_1)$ . By the induction hypothesis we have that  $w_2(T) \geq 3w_2(OPT_2)$ . Therefore,

$$\begin{aligned} w(R) &= w_1(R) + w_2(R) \\ &\geq w_1(R) + w_2(T) \\ &\geq 3w_1(OPT_1) + 3w_2(OPT_2) \\ &\geq 3w_1(OPT) + 3w_2(OPT) \\ &= 3w(OPT). \end{aligned}$$

$\square$

**THEOREM 5.3.** *The running time of  $SCHED(P, w)$  is polynomial in  $|P|$ .*

**PROOF.** In each iteration we remove at least one packet from  $P$  and therefore the number of iterations is bounded by  $|P|$ . Finding the minimal  $d(p)$  can be done in time  $O(|P|)$ . Updating  $w_1$  and  $w_2$  takes  $O(1)$  time per packet and in each iteration we have at most  $|P|$  updates. Checking whether  $T \cup \{p\}$  is a feasible solution can be done in  $O(|T|) \leq O(|P|)$  time. Therefore, the time complexity of each iteration is  $O(|P|)$  and the total time complexity is  $O(|P|^2)$ .  $\square$

We thus have a 3-approximation pseudo-polynomial time algorithm for the variable value case.

## 5.2 Unit Value Packets

For the unit value case, we will modify our algorithm in the following way. Note that when we use algorithm *SCEHD* on unit value packets, in each iteration, *SCEHD* removes a packet  $p$  and all the packets in  $C(p)$  completely. Therefore, the solution returned from the recursive call will not contain any packet in  $C(p)$ , and we can always add  $p$  to this solution. Thus, we can instead add  $p$  immediately to the solution, and then proceed to the recursive call. Instead of creating the set  $dup(q)$  for a packet  $q$ , we maintain the time window in which it can be scheduled and shrink it if necessary to avoid interference with the packets already scheduled. The algorithm is presented in Figure 10.

For each  $p \in S$  set  $r'(p) = r(p)$ .

While  $S \neq \phi$  do

Remove all packets in  $S$  that can no longer be fully scheduled by their time windows.

Select a packet  $p$  in  $S$  with minimal  $r'(p) + l(p)$ .

Schedule  $p$  during the time window  $[r'(p), r'(p) + l(p)]$ .

Remove  $p$  from  $S$ .

For each  $q \neq p$  with  $r'(p) \leq r'(q) \leq r'(p) + l(p)$  that conflicts with  $p$   
Set  $r'(q) = r'(p) + l(p) + 1$ .

**Figure 10: Algorithm *SCHED* - 1( $S$ ).**

LEMMA 5.4. *The running time of *SCHED* - 1( $S$ ) is polynomial in  $|S|$ .*

PROOF. In each iteration we remove at least one packet from  $S$  and therefore the number of iterations is bounded by  $|S|$ . Finding the minimal  $r(p) + l(p)$  can be done in time  $O(|S|)$ . Updating a time window for a given packet requires  $O(1)$  time and in each iteration we have at most  $|S|$  updates. Therefore, the time complexity of each iteration is  $O(|S|)$  and the total time complexity is  $O(|S|^2)$ .  $\square$

## 6. REFERENCES

- [1] N. Andelman, Y. Mansour and An Zhu, "Competitive Queueing Policies for QoS Switches," *The 14th ACM-SIAM SODA*, Jan. 2003.
- [2] T. Anderson, S. Owicki, J. Saxe and C. Thacker, "High speed switch scheduling for local area networks", *ACM Trans. on Computer Systems*, pp. 319-352, Nov. 1993.
- [3] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor and B. Schieber, "A unified approach to approximating resource allocation and scheduling", *Journal of the ACM*, Vol. 48(5), pp. 1069-1090, 2001.
- [4] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang, "On the Competitiveness of On-Line Real-Time Task Scheduling," *The Journal of Real Time Systems*, Vol. 4(2), pp. 124-144, 1992.
- [5] D. Black, S. Blake, M. Carlson, E. Davies, Z. Wang and W. Weiss, "An Architecture for Differentiated Services," *Internet RFC 2475*, December 1998.
- [6] A. Borodin and R. El-Yaniv, "Online Computation and Competitive Analysis," *Cambridge University Press*, 1998.
- [7] S. Dolev and A. Kesselman, "Non-Preemptive Real-Time Scheduling of Multimedia Tasks," *Journal of Real-Time Systems*, Vol. 17(1), pp. 23-39, July 1999.
- [8] S. Dolev and A. Kesselman, "Bounded Latency Scheduling Scheme for ATM Cells," *Journal of Computer Networks*, Vol 32(3), pp 325-331, March 2000.
- [9] Y. Ganjali, A. Keshavarzian, and D. Shah, "Input-Queued Switches : Cell Switching vs. Packet Switching," *Proceedings of INFOCOM 2003*.
- [10] J. A. Garay, J. Naor, B. Yener and Peng Zhao, "On-line Admission Control and Packet Scheduling with Interleaving," *Proceedings of INFOCOM 2002*.
- [11] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP Completeness," *W. H. Freeman*, San Francisco, 1979.
- [12] S. Goldman, J. Parwatikar and S. Suri, "On-line Scheduling with Hard Deadlines," *Journal of Algorithms*, Vol. 34, pp. 370-389, 2000.
- [13] S. Gopal and C. K. Wong, "Minimizing the Number of Switchings in a SS/TDMA System," *IEEE Trans. Commun.*, Vol. COM-33, pp. 497-501, June 1985.
- [14] R. Jain, K. Somalwar, J. Werth, and J. C. Browne, "Heuristics for Scheduling I/O Operations," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 8(3), 1997.
- [15] A. Kesselmann, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber and M. Sviridenko, "Buffer Overflow Management in QoS Switches," *Proceedings of STOC 2001*, pp. 520-529.
- [16] G. Koren and D. Shasha, " $D^{over}$ : An Optimal On-Line Scheduling Algorithm for Overloaded Uniprocessor Real-Time Systems," *SIAM J. Comput.* Vol. 24(2), pp. 318-339, 1995.
- [17] J. H. Lee and K. Y. Chwa, "Online Scheduling of Parallel Communications with Individual Deadlines," *Proceedings of ISAAC 1999*, pp. 383-392.
- [18] R. Lipton and A. Tomkins, "Online Interval Scheduling," *ACM-SIAM Symposium on Discrete Algorithms*, pp. 302-311, 1994.
- [19] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Packet Scheduling in Input-Queued Cell-Based Switches," *Proceedings of INFOCOM 2001*, pp. 1085-1094.
- [20] M. A. Marsan, A. Bianco, E. Leonardi and L. Milia, "RPA: A Flexible Scheduling Algorithm for Input Buffered Switches," *IEEE Transactions on Communications*, Vol. 47(12), pp.1921-1933, December 1999.
- [21] N. McKeown, "Scheduling Algorithms for Input-Queued Cell Switches," *Ph. D. Thesis*, University of California at Berkeley, 1995.
- [22] N. McKeown and A. Mekkittikul, "A Starvation Free Algorithm for Achieving 100% Throughput in an Input Queued Switch," *ICCCN 96*, Oct. 1996.
- [23] N. McKeown, P. Varaiya and J. Walrand, "Scheduling Cells in an Input-Queued Switch", *IEEE Electronics Letters*, pp. 2174-2175, Dec. 1993.
- [24] H. Moon and D. K. Sung, "Variable Length Packet Scheduling Algorithm for IP Traffic," *Proceedings of JCCI 2001*, April 2001.
- [25] D. Sleator and R. Tarjan, "Amortized Efficiency of List Update and Paging Rules," *CACM 28*, pp. 202-208, 1985.