

# RPA: A Flexible Scheduling Algorithm for Input Buffered Switches

Marco Ajmone Marsan, *Fellow, IEEE*, Andrea Bianco, *Member, IEEE*, Emilio Leonardi, and Luigi Milia

**Abstract**—This paper presents and evaluates a quasi-optimal scheduling algorithm for input buffered cell-based switches, named reservation with preemption and acknowledgment (RPA). RPA is based on reservation rounds where the switch input ports indicate their most urgent data transfer needs, possibly overwriting less urgent requests by other input ports, and an acknowledgment round to allow input ports to determine what data they can actually transfer toward the desired switch output port. RPA must be executed during every cell time to determine which cells can be transferred during the following cell time. RPA is shown to be as simple as the simplest proposals of input queuing scheduling, efficient in the sense that no admissible traffic pattern was found under which RPA shows throughput limitations, and flexible, allowing the support of packet-mode operations and different traffic classes with either strict priority discipline or bandwidth guarantee requirements. The effectiveness of RPA is assessed with detailed simulations in uniform as well as unbalanced traffic conditions and its performance is compared with output queuing switches and the optimal maximum weighted matching (MWM) algorithm for input-buffered switches. A bound on the performance difference between the heuristic weight matching adopted in RPA and MWM is analytically computed.

**Index Terms**—Input buffering, scheduling algorithms, switch architectures.

## I. INTRODUCTION

SWITCH designs with output buffers are very popular because they provide optimal performance. However, output buffered switch designs require a switching fabric speedup equal to  $N$ , i.e., an internal data transfer rate  $N$  times higher than the external link speed,  $N$  being the number of switch input-output ports. Since data rates on point-to-point fiber links keep growing very rapidly, providing within switching fabrics the required speedup for output buffered designs is becoming harder. Moreover, when adopting the output buffered architecture, within a cell time,  $N$  cells could need to be transferred to the same output port, thus requiring

Paper approved by P. E. Rynes, the Editor for Switching Systems of the IEEE Communications Society. Manuscript received June 30, 1998; revised April 19, 1999 and April 28, 1999. This work was supported in part by a research contract between Politecnico di Torino and CSELT, and in part by the Italian Ministry for University and Research. This paper was presented in part at the IEEE Symposium on Computers and Communications (ISCC), Athens, Greece, June 1998, and also in part at the SCS Symposium on Performance Evaluations of Computer and Telecommunications Systems (SPECTS), Chicago, IL, July 1999.

M. Ajmone Marsan, A. Bianco, and E. Leonardi are with the Dipartimento di Elettronica, Politecnico di Torino, 10129 Turin, Italy (e-mail: ajmone@polito.it; bianco@polito.it; leonardi@polito.it).

L. Milia is with Centro Ricerche Fiat, 10043 Orbassano, Italy (e-mail: l.milia@crf.it).

Publisher Item Identifier S 0090-6778(99)09774-3.

either a memory with access rate  $N$  times the link speed, or complex parallel memory architectures. For these reasons, nonoutput buffered switch designs have recently received a lot of attention.

Input buffered switches are designed to operate with a switching fabric running at an internal rate equal to the external links speed; unfortunately, when using a first-input first-output (FIFO) queuing discipline at input queues, due to the head-of-the-line (HoL) blocking phenomenon, they provide a maximum achievable throughput limited to about 60% of the link speed, under uniform traffic conditions [1]. In order to either reduce, or completely overcome, the throughput penalty, separate queues are required at each input port for the storage of cells directed to different output ports.

Once cells have been sorted into separate queues, the performance of an input buffered switch essentially depends on its scheduling algorithm. With the term scheduling algorithm, we refer in this paper to the algorithm required to select the cells that must be transferred within a cell time from input queues to output ports, with the constraint of selecting not more than one cell from each input port and not more than one cell directed to each output port. These algorithms must be efficient, to provide throughput comparable with that of output buffered switches, and simple, since they must be executed within a cell time in high-speed switches.

A different issue is the ability to deal with the requirements of different traffic classes in input buffered switches, i.e., providing the means to give priority to cells belonging to classes of traffic with more stringent quality-of-service (QoS) requirements. Unfortunately, also the algorithms used for this purpose in the literature are called scheduling algorithms; we will use the term QoS-aware scheduling for such algorithms [2], [3].

Several scheduling algorithms were presented in the technical literature to address these issues (mostly trying to overcome HoL blocking) in input buffered switches under uniform traffic conditions [4]–[7]. Most of them focus on architectures where  $N$  separate queues are available at each input port, each one storing cells directed toward a different output port. However, those algorithms do not succeed in providing the maximum achievable throughput under nonuniform traffic patterns. Some of them can even lead to starvation in some traffic scenarios.

To the best of our knowledge, the first proposal of an algorithm that leads to the optimal and fair exploitation of the switch bandwidth under every admissible traffic pattern appeared recently (see the maximum weighted matching (MWM) algorithm of [8]). It is based on solving a bipartite graph

matching problem, where one node partition is associated with input ports and the other node partition with output ports; edges are labeled with a number representing the edge weight according to a suitable criterion, and the algorithm selects the matching that provides the maximum total weight. Unfortunately, these algorithms are not simple to implement, requiring a computational complexity  $O(N^3 \log N)$ , and they do not allow the management of different traffic classes. As a consequence, they may not be adequate for the solution of all problems in the implementation of large high-speed switches.

Simpler solutions can be based on a maximum size matching (MSM) algorithm (that can be seen as a MWM with weight set to 1 on all edges), whose complexity is  $O(N^{5/2})$ . However, MSM algorithms can lead to starvation under inadmissible traffic (i.e., with traffic patterns not satisfying the relations  $\sum_{i=1}^N \lambda_{ij} \leq 1$ ,  $\sum_{j=1}^N \lambda_{ij} \leq 1$ , where  $i, j$  represent, respectively, the input and output port indices, and  $\lambda_{ij}$  is the average load from input port  $i$  to output port  $j$ ), and instability, i.e., throughput loss, under admissible traffic.

In this paper, we describe reservation with preemption and acknowledgment (RPA), a scheduling algorithm previously presented and examined by the authors in [9] and [10], whose computational complexity is only  $O(N^2)$ . RPA leads to an efficient and fair exploitation of the switch bandwidth under several critical traffic conditions; at the same time, it provides simple approaches to deal with different traffic classes and packet-mode (as opposed to cell-by-cell) operations.

The novel contribution of this paper is threefold. First, we show how the original RPA algorithm [9] can be adapted to deal with multiple traffic classes providing a strict priority discipline, presenting detailed simulation results under several traffic patterns and comparing RPA with output buffered architectures and with MWM. Part of these results was presented in [10], although here we consistently use confidence intervals in all simulation runs. Second, we extend RPA to jointly deal with cell scheduling and QoS-aware scheduling, although in this context, performance is less satisfactory than that obtained in output buffered switches using well-known QoS-aware scheduling [3]. Third, we prove that the weight of the matching obtained with RPA in the worst case cannot be less than half the maximum weight, which is obtained with the MWM algorithm.

## II. RELATED WORK

Several scheduling algorithms for input buffered switches were recently presented. Among them are iSLIP in [6], 2DRR in [11], iLQF and iOCF in [12], RPA in [9], MUCS in [13], and iLPF in [14].

Their characteristics can be examined from several points of view; for a detailed comparison in terms of throughput, delay, complexity, and flexibility, see [15]. We provide in this paper a brief taxonomy and highlight only the peculiar characteristics of RPA.

Given the complexity of the MWM (and MSM) algorithm, most of the algorithms proposed in the literature rely on heuristics to approximate the behavior of an exact MWM

or MSM algorithm. The heuristic algorithm characterizes and distinguishes the various proposals.

Whereas iSLIP and 2DRR propose heuristics to mimic MSM, all other proposals (including RPA) try to emulate the more efficient (but more complex) MWM algorithm. RPA, as detailed in the following sections, requires ordered decisions at each input port; decisions are taken on the basis of some local information and a limited amount of global information, stored in a reservation array. This approach can be implemented in distributed architectures, but it makes RPA difficult to run on parallel processors, although a time-pipeline can be envisioned.

Another important aspect is related to the metrics used by the algorithms to choose a subset of cells to be transmitted in each cell time (via the edge weights). Excluding the MSM-like algorithms, whose admissible edge weights are only 0 (no cell to transfer) and 1 (at least one cell to transfer), we can identify the following metrics: 1) queue length (QL) metrics, where edge weights are associated with the queue length, used in iLQF and RPA; 2) port occupation (PO) metrics, where edge weights are based on input and output port occupation, used in iLPF; 3) MUCS metrics, a peculiar metric defined and used in MUCS; and 4) oldest cell (OC) metrics, where edge weights are computed as cell queuing delays, adopted in iOCF. Of course, each metric has its advantages (simplicity for QL, good delay control for OC) and disadvantages (potential starvation for lightly loaded queues for QL, implementation complexity for OC and MUCS, inefficiency under nonuniform traffic for PO). Note that although RPA and iLQF share the same metrics, since they are based on different heuristic algorithms, they provide (slightly) different performance.

Concerning algorithmic complexity, measured as the number of operations to be executed on a single central processor running the scheduling algorithm, we showed in [15] that MUCS has complexity  $O(N^3)$ , although it has been efficiently implemented in analog hardware, iSLIP, iLQF, and iOCF have complexity  $O(N^2 \log N)$ , adopting the suggested number of iterations,  $\log N$  [6], and RPA, iLPF, and 2DRR have complexity  $O(N^2)$ . As a consequence, RPA shares with 2DRR and iLPF the lowest complexity; however, RPA, 2DRR, and iLPF cannot be run on parallel processors, a nice feature presented by iSLIP, iLQF, and iOCF, which could reduce the algorithmic complexity, if measured as the number of operations to be executed on each parallel processor. Recall that the scheduling algorithm must be executed during every cell time to determine which cells must be transferred during the following cell time. It is thus very important to keep complexity at a minimum.

RPA can be easily adapted to run in packet-mode operation, it can support multiple classes of traffic in a strict priority discipline, and it can support QoS-aware scheduling algorithms. The only previous proposal of QoS-aware scheduling algorithm for IQ switches that appeared in the literature so far is called weighted probabilistic iterative matching (WPIM) [16]. WPIM allows the allocation of the bandwidth on each output channel to connections entering the switch at different inputs. However, WPIM cannot separately consider the bandwidth allocations to flows with the same input and output

ports (WPIM implements what the authors call a ‘connection-level’ scheduling, as opposed to a ‘flow-level’ scheduling). Moreover, the random choices on which WPIM is based limit the maximum throughput on each output channel to about 90%.

Finally, RPA can be also modified to handle multicast traffic, using an approach similar to that adopted in the iPOINT multicast contention resolution algorithm (iMCRA), since both scheduling algorithms are based on the use of a vector where the information for contention resolution is stored [17]. We do not tackle the multicast problem in this paper, whereas iMCRA mainly focuses on this issue. Other multicast scheduling algorithms for input-queued switches were presented in the literature, for example in [18].

### III. RPA SCHEDULING

RPA stands for reservation with preemption and acknowledgment. The name indicates that the scheduling algorithm is based on a reservation round where the switch input ports can indicate their most urgent cell transfer needs, possibly overwriting less urgent requests by other input ports, and an acknowledgment round to allow input ports to determine what cell they can actually transfer toward the desired switch output port.

Denote by  $N$  the number of input and output ports of the switch, and label input and output ports with a subscript  $j$ ,  $0 \leq j \leq N - 1$ . We shall denote by  $i_k$  and  $o_k$ , respectively, the  $k$ th input and output ports.

In the description of RPA, we first concentrate on the case of one traffic class, in Section III-A, providing some indications about the extensions required to implement packet-mode operations. Then, in Section III-B, we illustrate priority-RPA (P-RPA), the version of RPA that is capable of supporting multiple traffic classes with a strict priority discipline. Finally, in Section III-C, we describe fair-RPA (F-RPA), the version of RPA supporting QoS-aware scheduling.

#### A. RPA with One Class of Traffic

RPA requires, at each input port, the availability of  $N$  separate FIFO queues for the storage of cells directed to different output ports. More precisely, the  $j$ th queue at  $i_k$ , denoted by  $Q_{k,j}$ , contains cells directed from the considered input port toward  $o_j$ .

RPA operates on an array where reservations are written by input ports at each cell time. This reservation array will be denoted by RES; it contains  $N$  elements, denoted by RES( $j$ ) with  $0 \leq j \leq N - 1$ , that refer to the output ports in order (RES( $j$ ) refers to  $o_j$ ).

Each RES( $j$ ) element comprises three fields.

- 1) RES( $j$ ).port\_id contains the address of the input port trying to reserve a cell transmission to  $o_j$ .
- 2) RES( $j$ ).urg contains the urgency (a value  $>0$ ) of the cell transfer from RES( $j$ ).port\_id to  $o_j$ . The urgency defines the importance of the cell transfer; several approaches may be adopted for the computation of the urgency values; packet-mode operations as well as multiple classes

of traffic can be managed with clever definitions of the urgency, as we shall see.

- 3) RES( $j$ ).busy is set to 1 during the acknowledgment round by the input port being granted the right to transfer a cell to  $o_j$ .

The three fields are initialized to null values at the beginning of every reservation round.

Input ports access array RES following a preestablished order. For the description of the RPA operations, suppose that, within some arbitrary cell time,  $i_a$  is selected as the first input port in the access order; the other input ports follow, for example, in ascending subscript order.

In the reservation round,  $i_a$  selects its most urgent cell (i.e., the cell with the highest urgency among the cells at the head of its  $N$  input queues  $Q_{a,j}$ ,  $0 \leq j \leq N - 1$ ) and issues a reservation for a cell transfer toward the output port of this most urgent cell. If the output port to which the most urgent cell must be transferred is  $o_w$ , port  $i_a$  records its index  $a$  in the RES( $w$ ).port\_id field, and the urgency of the selected cell in the RES( $w$ ).urg field.

The reservation array is accessed next by  $i_b$ , the input port following  $i_a$  in the access order (if the order is by increasing index,  $b = |a + 1|_N$ ). Input port  $i_b$  first evaluates a weight function  $W$  for each output port  $o_j$

$$W(o_j) = U(Q_{bj}) - \text{RES}(j).\text{urg}$$

where  $U(Q_{bj})$  is the urgency of the cell at the head of queue  $Q_{bj}$ . Then  $i_b$  computes  $W(j_{\max}) = \max_j W(o_j)$  and records the value  $x = j_{\max}$  at which the function reaches its maximum. If  $W(x) > 0$ ,  $i_b$  is allowed to issue a reservation for a cell transfer toward  $o_x$ , writing its index  $b$  in RES( $x$ ).port\_id and  $U(Q_{bx})$  in RES( $x$ ).urg; this implies that  $i_b$  may overwrite a previous reservation for a cell transfer toward  $o_x$ , thus preempting it. If  $W(j_{\max}) \leq 0$ , no reservation can be issued.

The reservation round then continues; array RES is next processed by input port  $i_c$ , where  $c = |b + 1|_N$ , which executes the same algorithm. After all input ports have processed array RES, the reservation round terminates.

At this point, the acknowledgment round immediately starts, and array RES is processed for the second time by all input ports in the same order followed during the reservation round.

First,  $i_a$  checks whether its reservation toward  $o_w$  has been overwritten by any other input port with more urgent cell transfers toward  $o_w$ . If the reservation has not been overwritten,  $i_a$  is granted access to  $o_w$ , and the RES( $w$ ).busy field is set to 1. Otherwise,  $i_a$  cannot transfer a cell to  $o_w$ ; however, if at least one ‘idle’ output port is found, i.e., an output port toward which no reservation has been made (the port\_id field is null),  $i_a$  is allowed to transfer a cell toward any one of the idle output ports. In practice,  $i_a$  selects the idle output port for which it has the most urgent cell and sets to 1 the corresponding busy field. Note that in an  $N \times N$  switch, if a reservation has been overwritten, at least one idle output port exists. The same may not be true when the number of output ports is smaller than the number of input ports.

Next, port  $i_b$  processes array RES to check whether its reservation toward  $o_x$  has been overwritten; if not, it can access the desired output port, and it sets the RES( $x$ ).busy field to

1. Otherwise, it checks whether any idle output port exists; if some idle output ports are found,  $i_b$  is granted access toward the output port for which it has the most urgent cell, and the appropriate *busy* field is set to 1.

This acknowledgment algorithm is orderly executed by all input ports; when all input ports have processed array RES, the scheduling algorithm terminates. One cell can now be transferred from each input port to the output port toward which it has been granted access.

The ordering of input ports in their access to the reservation vector can either be the same in any cell time (in our example, this means that  $i_a$  always is the first input port,  $i_b$  the second, and so on), or it can change. We call static the former version of the scheduling and dynamic the latter. In the presentation of numerical results, we shall consider the static version, as well as a dynamic version where a round-robin selection of the starting point of the cyclic rounds is performed. Of course, the static version leads to some unfairness (only for delays, not for throughputs), due to the fact that input ports always have the same position within the round. The simple dynamic version that we consider somewhat remedies such unfairness.

Several metrics can be adopted to quantify the cell urgency. When just one class of traffic flows enters the switch, the number of cells stored at each input queue can be chosen as the urgency for the cell at the head of the queue; this is the urgency metric that we used in our simulation experiments. With this metric, we prove in Section V that the weight of the matching obtained with RPA in the worst case cannot be less than half the maximum weight, produced by the MWM algorithm.

1) *RPA in Packet-Mode Operation*: In order to obtain a packet-mode operation, which may be quite useful if the switch should, for example, offer a service consisting in the routing of IP packets, it is possible to dynamically alter priorities whenever the first cell of a packet is transferred from the switch input port to the switch output port, so that the transfer of a sequence of cells corresponding to one packet cannot be interrupted.

More formally, whenever input port  $i_x$  has been granted access toward output port  $o_w$  for the first cell of a packet, it sets the urgency value of the queue  $Q_{xw}$ ,  $U(Q_{xw}) = \text{MAX\_URGENCY}$ , the maximum admissible value for the urgency metrics. This value is kept until the last cell of the packet has been transferred toward output port  $o_w$ .

Note that this allows a contiguous transfer of cells belonging to the same packet, thus avoiding any requirement of packet reassembly at output ports; cut-through output transmission could be envisioned in this context, to reduce the packet transit delay.

### B. P-RPA: Multiclass RPA with Strict Priority Discipline

The support of several different traffic classes in a strict priority framework requires some modifications to the RPA algorithm: we call this version priority-RPA (P-RPA). First of all, denoting by  $p$  the number of traffic classes, each input port must be equipped with a number of queues equal to  $pN$ , in order to separate cells directed to a specific output port within each traffic class. Second, the reservation and acknowledgment

algorithms must be revised so as to separate the service of different traffic classes. Finally, the definition of the urgency metrics must be changed aiming at the support of different types of QoS requirements.

In this section, we first describe the modifications to be introduced in the RPA algorithms, and later we focus on the urgency metrics that allows the implementation of a static priority service discipline among traffic classes.

When  $p$  traffic classes are present at the switch input ports, in order to achieve acceptable performance, the RPA reservation round must comprise  $R$  reservation cycles to be executed in sequence, before the acknowledgment round. This is necessary because, as we shall see, successive overwriting of reservations made by different input ports toward the same output port can occur, if higher priority cells must be transferred by input ports that access the reservation array later in the round (according to the preestablished ordering). This could unnecessarily forbid transmissions of cells destined toward different output ports from input ports accessing the reservation array early in the round. A small number of reservation cycles is sufficient to reduce this undesirable phenomenon.

During the whole reservation round, like in single-class RPA, each input can reserve the transmission of no more than one cell toward one output port. During each one of the  $R$  reservation cycles, each input port that has not yet been successful in its reservation can overwrite previous reservations of other input ports if it has more urgent cells at the heads of its queues. At the end of the  $R$ th reservation cycle, the reservation round terminates, and an acknowledgment round starts, where each input checks whether its reservation has been overwritten by any other input port. If the reservation has not been overwritten, the input port is granted access to the desired output. Otherwise, the input port cannot transfer any cell toward any output. Note that in P-RPA, reservations of cell transfers toward idle output ports (i.e., ports toward which no reservation has been made) are not allowed during the acknowledgment round; this slight modification is required in order to guarantee a static priority service discipline among traffic classes.

Consider now the definition of an urgency metric that allows the implementation of a static priority service discipline among traffic classes. In a multiclass traffic scenario, the urgency value must be defined for each cell at the head of every transmission queue ( $pN$  urgencies for each input port).

Since we wish to transmit cells destined to the same output port according to a static priority discipline, the transmission schedule of higher priority cells must be completely independent of the lower priority traffic load. This implies that the urgency of every higher priority class cell must be greater than the urgency of any lower class priority cell. Hence, the urgency function must return values ranging in disjoint intervals for different traffic class cells. However, this condition is not sufficient; for the implementation of a priority service, it is necessary to guarantee that a reservation for a lower priority cell directed to output port  $o_x$  is impossible if a higher priority cell directed to output port  $o_y$ , with urgency larger than  $\text{RES}(y).\text{urg}$ , is present at the same input. Thus, the minimum urgency value returned for cells belonging to traffic class  $k$

must be greater than twice the maximum value returned for lower priority cells. Moreover, any difference among values returned by the urgency function for  $k$ -class cells must exceed the maximum value taken by lower priority cells.

Let  $Q_{nmk}$  be the queue at input port  $i_n$  storing cells directed to output  $o_m$  and belonging to traffic class  $0 \leq k < p$ . Let  $Lq_k$  be the capacity of queues at input ports storing cells of traffic class  $k$ . The urgency value  $U(Q_{nmk})$  assigned to the cell at the head of queue  $Q_{nmk}$  depends on the traffic class  $k$  and the number of cells in the transmission queue  $Nq_{nmk}$ . A possible expression of the cell urgency is

$$U(Q_{nmk}) = \frac{Nq_{nmk}}{Lq_k} \frac{1}{A^k}$$

where  $A = 3 \max_k Lq_k$ .

Observe that the modification of the reservation round and of the urgency metrics with respect to the original RPA algorithm allow a strict priority service discipline to be obtained; lower priority cells never access the switching matrix prior to higher priority cells. This occurs independently of the number of reservation cycles  $R$ ; increasing the number of reservation cycles only improves the overall switch performance.

### C. F-RPA: Multiclass RPA with Fair Bandwidth Allocation

By changing the urgency metrics, RPA can be extended to deal with QoS requirements. We present fair-RPA (Fair-RPA), an adaptation of RPA, that allows a good bandwidth separation among different traffic flows to be obtained.

According to the self-clocked weighted fair queueing (SC-WFQ) algorithm proposed in [3] for OQ architectures, when the  $n$ th cell belonging to flow  $f$  (that enters the switch at  $i_k$  and exits from  $o_j$ ) arrives at the (output) queue, a tag  $F_f^{(n)}$  is computed and associated with the cell

$$\begin{aligned} F_f^{(n)} &= 1/r_f + \max\left(F_f^{(n-1)}, V_j^t\right) \\ F_f^{(0)} &= 0 \end{aligned} \quad (1)$$

where  $r_f$  is the negotiated rate for flow  $f$ , and the virtual time  $V_j^t$  associated with the output queue is set equal to the tag of the last cell that has been transmitted on  $o_j$ . Tag  $F_f^{(n)}$  represents the time stamp of the cell, and cells are transmitted according to increasing tag values.

In the case of IQ switches, the same formulation can be used, setting the virtual time  $V_j^t$  equal to the tag of the last cell that has been transferred (from any input) toward  $o_j$ ; note that this cell will be transmitted on the output link with no delay, since no queuing at output ports is required in IQ switches.

QoS-aware scheduling in IQ switch architectures must select the cells to be transferred from input to output ports, considering two requirements: 1) throughput optimization; 2) QoS provision. If priority is given to throughput optimization, QoS cannot be guaranteed; conversely, if priority is given to QoS provision, unacceptable switch performance is obtained. Trying to achieve a compromise between the two extremes implies that some sacrifice is necessary in performance and that the implementation of WFQ scheduling is not perfect; this means that cells directed toward any output  $o_j$  are not always transmitted in strictly increasing time-stamp order.

As a consequence, since the algorithms used in IQ switches cannot guarantee that cells directed to a specific output  $o_j$  are transmitted in increasing tag value order, results obtained in IQ switches are different from those of OQ switches, even if the same SC-WFQ algorithm is used.

We assume that the value of  $V_j^t$  can be known at all inputs simply by observing the cells that are transferred through the switching matrix. Should this not be possible, the values of  $V_j^t$  can be broadcast to all inputs by adding a field  $\text{RES}(j).\text{Vt}$  to array RES. The value of  $\text{RES}(j).\text{Vt}$  at step  $n$  gives the value of  $V_j^t$  at step  $n + 1$ .

In F-RPA, it is necessary to couple the tag definition with the urgency metrics characteristics of RPA. This is a peculiar requirement introduced by the IQ architecture. We define the cell urgency  $U(Q_{kj})$ , associated with the cell at the head of the queue storing cells directed toward output port  $o_j$  at input port  $i_k$ , as the smallest tag associated with cells belonging to the queue.

The initial value for each element of the array  $\text{RES}[j].\text{urg}$  (i.e., the urgency value associated with no reservation) is set equal to  $V_j^t + (1/(\min(r_j)))$ , where  $\min(r_j)$  is the smallest rate allocated to flows directed to output  $o_j$ .

Given the above assumptions, the F-RPA algorithm can be executed as the normal RPA algorithm, with only a minor modification: since with our definition, smaller tags correspond to more urgent cells, the  $\max()$  function must be replaced by a  $\min()$  function. Moreover, if the field  $\text{RES}(j).\text{Vt}$  is used, during the acknowledgment round, input  $i_k$  that can transfer a cell to  $o_j$  must set the value of  $\text{RES}(j).\text{Vt}$  equal to the urgency of the cell to be transferred. This value will be read by all inputs during the following reservation round.

## IV. COMPLEXITY

We now prove that the computational complexity of the RPA scheduling algorithm is  $O(N^2)$ , as claimed in Section I. We first focus on the single traffic class RPA, then we also discuss P-RPA and F-RPA.

During the reservation round,  $O(N)$  operations are required at each input port to compute  $W(o_j)$  for each  $o_j$ ; in addition,  $O(N)$  operations are required to determine the output port  $o_{j_{\max}}$  for which  $W(o_j)$  is maximized; one operation is sufficient to verify whether  $W(j_{\max}) > 0$ .

The computational complexity of the reservation round is thus  $O(N^2)$ , since it is composed of  $N$  sequential steps, each one of complexity  $O(N)$ .

During the acknowledgment round,  $O(1)$  operations are required at each input port to verify whether the reservation has been overwritten; in addition,  $O(N)$  operations may be necessary to obtain the set of idle outputs and to determine the one for which the input port has the most urgent cell.

As a consequence, the computational complexity of the acknowledgment round is also  $O(N^2)$ . Hence, the total complexity of the scheduling algorithm is  $O(N^2)$ . Recall that this should be compared with the complexity of the MWM algorithm that has been shown to be  $O(N^3 \log N)$  in [8].

The computational complexity of P-RPA, the multiclass RPA with strict priority, is  $O(RN^2)$ . Indeed, during each one of the  $R$  reservation rounds,  $O(N^2)$  operations are required,

while during the final acknowledgment round only  $O(N)$  operations are needed.

Finally, with F-RPA, since only a new urgency definition is required, the complexity remains  $O(N^2)$ .

### V. WORST-CASE WEIGHT DIFFERENCE BETWEEN RPA AND MWM

In this section, we prove that the worst-case weight of any matching obtained with RPA is larger than half the weight of the corresponding matching generated by the MWM algorithm. The result is formally expressed by the following theorem.

*Theorem:* For any given occupancy of the virtual output queues of an input-queued switch, the sum of the weights in the matching obtained with the RPA algorithm cannot be less than half the sum of the weights in the matching obtained with the optimal MWM algorithm.

*Proof:* We consider an  $N \times N$  input-queued switch, where  $N$  separate virtual output queues exist at each input port, organized according to the VOQ architecture.

Let  $\mathcal{Q} = \{Q_{ij}\}$  represent a set of queues, where element  $Q_{ij}$  is the queue at input  $i$  storing packets directed to output  $j$ . Each queue  $Q_{ij}$  has an associated weight, which, according to the metrics adopted by RPA, coincides with the queue length.

Let  $\mathcal{Q}_{RPA}$  be the set of queues in the matching obtained at the end of the first reservation round of the RPA algorithm. Let  $\mathcal{Q}_{MWM}$  be the set of queues in the matching obtained with the MWM algorithm. Note that  $\max|\mathcal{Q}_{RPA}| = \max|\mathcal{Q}_{MWM}| = N$ . Also, according to the constraints imposed by the input-queued switch architecture, both  $\mathcal{Q}_{RPA}$  and  $\mathcal{Q}_{MWM}$  can contain at most one queue from each input and at most one queue storing cells directed toward each output.

Consider a set of queues  $\mathcal{Q} = \{Q_{ij}\}$  such that only one of its elements,  $Q_{ik}$ , stores data units directed to output  $k$ , and such that only one of its elements,  $Q_{kj}$ , stores data units at input  $k$ .

Let  $O(\mathcal{Q}, k)$  be the function defined on  $\mathcal{Q}$  that returns  $Q_{ik}$ ;  $O(\mathcal{Q}, k) = \emptyset$  if no queue containing data units directed to  $k$  is found in  $\mathcal{Q}$ .

Let  $I(\mathcal{Q}, k)$  be the function defined on  $\mathcal{Q}$  that returns  $Q_{kj}$ ;  $I(\mathcal{Q}, k) = \emptyset$  if no queue at input  $k$  is found in  $\mathcal{Q}$ .

The set  $\mathcal{Q}_{RPA}$  is incrementally created starting with an empty set. Each input can modify the set  $\mathcal{Q}_{RPA}$  following the same cyclic ordering adopted during the reservation round; let us assume that input 0 is the first input in the reservation round, while input  $N - 1$  is the last. Let  $\mathcal{Q}_{RPA}^i$  be the set of queues included in the matching by the RPA algorithm, after input  $i$  has made its reservation; obviously,  $\mathcal{Q}_{RPA}^N = \mathcal{Q}_{RPA}^{N-1}$ . Note that  $\mathcal{Q}_{RPA}^i$  is not guaranteed to be either a subset of  $\mathcal{Q}_{RPA}$  or of  $\mathcal{Q}_{RPA}^{i+1}$ , since some queues belonging to  $\mathcal{Q}_{RPA}^i$  can be removed later by other inputs.

Let  $W(\mathcal{Q})$  be a function defined on  $\mathcal{Q}$  that returns the sum of the weights associated with all queues in  $\mathcal{Q}$ . By definition,  $W(\emptyset) = 0$ .

Let  $\Delta W(i) = W(\mathcal{Q}_{RPA}^i) - W(\mathcal{Q}_{RPA}^{i-1})$  be the weight increment obtained with the reservation performed by input  $i$  in the RPA algorithm. Note that the RPA algorithm guarantees that  $\Delta W(i) \geq 0$ .

Let us compare  $W(\mathcal{Q}_{MWM})$  and  $W(\mathcal{Q}_{RPA})$ , and suppose that  $\mathcal{Q}_{MWM} \neq \mathcal{Q}_{RPA}$ ; otherwise, trivially  $W(\mathcal{Q}_{MWM}) = W(\mathcal{Q}_{RPA})$ , and the theorem is proved. Note that in the comparison, we will use the sets  $\mathcal{Q}_{RPA}^i$  defined during the cyclic processing performed by each input.

Let us examine all inputs  $i = 0, \dots, N - 1$ , in the same order followed during the RPA reservation round, and let  $k$  be an input for which

$$I(\mathcal{Q}_{RPA}, k) \neq I(\mathcal{Q}_{MWM}, k). \tag{2}$$

Let  $Q_{km}$  be the queue selected at input  $k$  by the MWM algorithm, and  $Q_{kr}$  be the queue selected at input  $k$  by the RPA algorithm; i.e.,  $Q_{km} = I(\mathcal{Q}_{MWM}, k)$ , and  $Q_{kr} = I(\mathcal{Q}_{RPA}, k)$ . Note that it is possible that either  $Q_{km} = \emptyset$  or  $Q_{kr} = \emptyset$ .

Define

$$\mathcal{Q}^* = \mathcal{Q}_{RPA}^{k-1} \cup Q_{km} - O(\mathcal{Q}_{RPA}^{k-1}, m)$$

the set that would be obtained by adding  $Q_{km}$  instead of  $Q_{kr}$  to  $\mathcal{Q}_{RPA}^{k-1}$  and, if necessary, removing the queue in  $\mathcal{Q}_{RPA}^{k-1}$  directed to output  $m$  (the sign  $-$  is used to indicate removal from the set).

Since  $Q_{kr}$  has been chosen at input  $k$  according to the RPA algorithm, it holds

$$W(\mathcal{Q}_{RPA}^k) \geq W(\mathcal{Q}^*).$$

However, since

$$W(\mathcal{Q}^*) = W(\mathcal{Q}_{RPA}^{k-1} \cup Q_{km} - O(\mathcal{Q}_{RPA}^{k-1}, m))$$

we get

$$W(\mathcal{Q}_{RPA}^k) \geq W(\mathcal{Q}_{RPA}^{k-1}) + W(Q_{km}) - W(O(\mathcal{Q}_{RPA}^{k-1}, m))$$

thus

$$W(Q_{km}) \leq \Delta W(k) + W(O(\mathcal{Q}_{RPA}^{k-1}, m)).$$

The RPA algorithm guarantees that

$$W(O(\mathcal{Q}_{RPA}^{k-1}, m)) \leq W(O(\mathcal{Q}_{RPA}, m))$$

so, in conclusion

$$W(Q_{km}) \leq \Delta W(k) + W(O(\mathcal{Q}_{RPA}, m)). \tag{3}$$

By repeating the same process over all inputs  $k$  for which  $I(\mathcal{Q}_{RPA}, k) \neq I(\mathcal{Q}_{MWM}, k)$ , we can prove that (3) holds for each input satisfying (2), and obviously for all other inputs.

By summing over all inputs, we finally obtain

$$\begin{aligned} W(\mathcal{Q}_{MWM}) &\leq \sum_{i=1}^N \Delta W(i) + \sum_{i=1}^N W(O(\mathcal{Q}_{RPA}, i)) \\ &= 2W(\mathcal{Q}_{RPA}). \end{aligned} \quad \square$$

### VI. SIMULATION SCENARIO

For our simulation experiments, we considered  $8 \times 8$  as well as  $16 \times 16$  switch configurations, with two different traffic patterns. Both traffic patterns assume that the loads of all input ports are equal; however, in the first case (uniform load), the loads of all output ports are equal, whereas in the second (hot-spot load), one of the output ports (the hot spot) is subject to twice the load of all others.

Concerning the input traffic characterization, we consider the following four types of statistics.

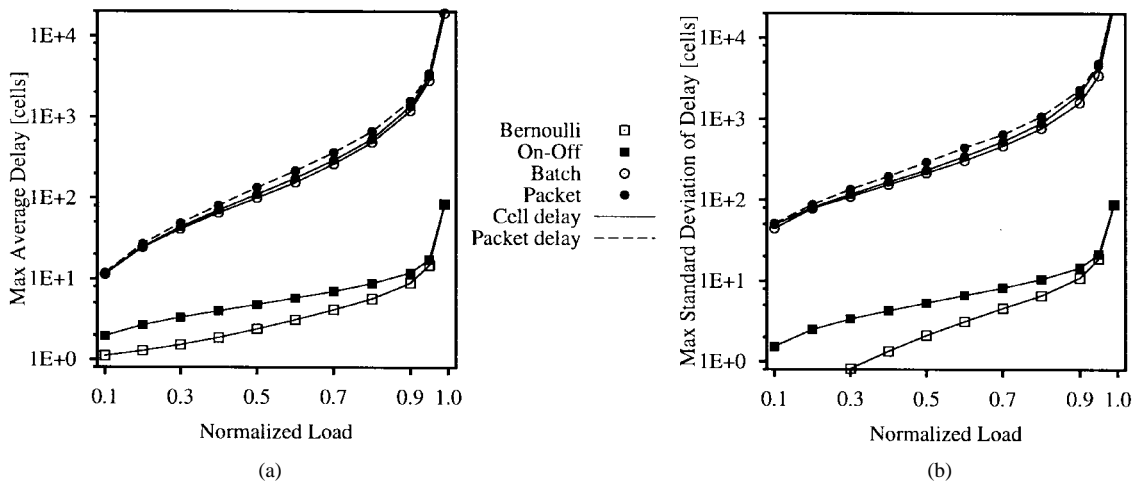


Fig. 1. Worst-case cell access delays: (a) average and (b) standard deviation versus the normalized switch load, for static single class RPA in an  $8 \times 8$  switch, under uniform traffic load.

- *Bernoulli* input traffic: cells arrive at input ports according to a Bernoulli process; the cell output ports are selected with random, independent choices.
- *ON-OFF* input traffic: cells continuously arrive at input ports during geometrically distributed ON periods, whose average duration is 100 cells; no cells arrive during geometrically distributed OFF periods, whose average duration is selected so as to obtain the desired load; the cell output ports are selected with random, independent choices.
- *Batch* input traffic: similar to ON-OFF input traffic, but now the cell output port is the same for all cells arriving during one ON period, and is selected with a random choice at the beginning of the batch.
- *Packet* input traffic: similar to batch input traffic, but now batches logically correspond to packets.

From the viewpoint of input traffic characterization, batch and packet arrivals are identical; we distinguish the two cases because we assume that within the switch, batch traffic is transferred with a cell-by-cell approach, whereas packet traffic uses the packet-mode version of RPA.

The queue capacity is set to 10 000 cells for each input queue, so that a total of 80 000 cells per input port can be stored in an  $8 \times 8$  switch. This limitation of queue capacities is obviously responsible of a saturation of delay curves; this saturation is not shown in the figures presented in Section VII.

All numerical results were obtained by stopping simulation runs when a 5% confidence interval width was reached with 95% confidence level.

## VII. SIMULATION RESULTS

In this section, we briefly overview a sample of the simulation results that were obtained in the performance investigation of RPA. We only illustrate results obtained for an  $8 \times 8$  switch configuration because no significant difference was observed for  $16 \times 16$  switches.

We use as performance metrics the curves of the mean and standard deviation of the cell and packet access delays as functions of the normalized load of input ports (a load equal to

1 corresponds to a saturated input channel). Note that due to the uniformity assumption for input loads, the horizontal axes of our plots also indicate the normalized load of the whole switch.

We mainly consider the cell access delay, defined as the time between the cell arrival at the input port and the successful reservation of the cell transfer toward the desired output port.

The packet access delay is defined as the time between the arrival of the first cell of the packet at the input port and the successful reservation of the transfer of the same first cell toward the desired output port. Since the packet-mode operation of RPA guarantees that when the transfer of the first cell of the packet is successfully reserved, all subsequent cells follow with no interruption, the access delay is the same for all cells of the same packet. Nevertheless, when using packet-mode RPA, both cell and packet access delays will be presented, since the variable number of cells per packet induces a (small) difference between the two numerical results.

When plotting numerical results for the static version of RPA, if not otherwise stated, we choose the worst-case mean and standard deviation, i.e., the mean and standard deviation experienced by the least favored input port. Instead, when considering the dynamic version of RPA, we present the value obtained by averaging over all input ports.

We first present performance results for single-class RPA, then we compare RPA with pure output queueing and with the MWM of [8]. Then, we illustrate results for P-RPA in the case of two classes of traffic (i.e.,  $p = 2$ ). Finally, we compare F-RPA with the performance of a classical WFQ algorithm in output buffered architectures.

### A. Results for Single-Class RPA

To begin with, we present in Fig. 1 the curves of the cell (and packet, for packet input traffic) access delay mean and standard deviation versus the normalized load of the switch for static single-class RPA in the case the least favored input port under the uniform load pattern.

First of all, it should be noted that the access delay mean and standard deviation curves in the cases of Bernoulli and

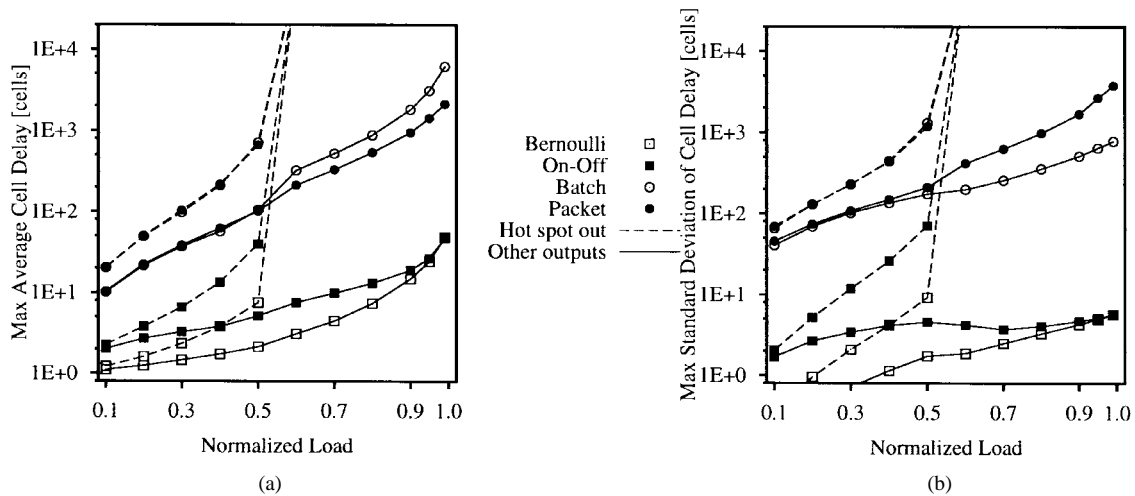


Fig. 2. Worst-case cell access delays: (a) average and (b) standard deviation versus the normalized switch load, for static single class RPA in an 8 × 8 switch, under hot-spot traffic load.

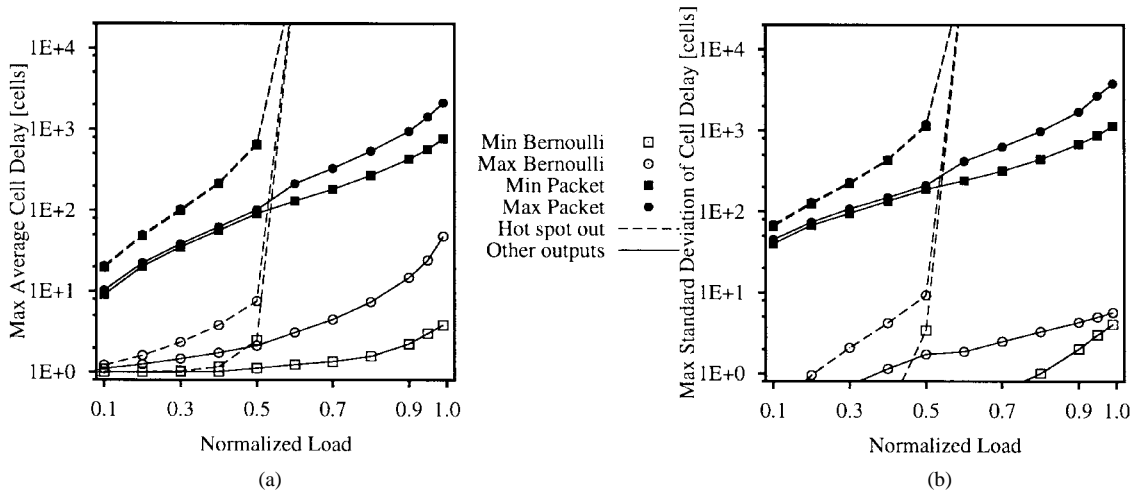


Fig. 3. Worst and best cell access delays: (a) average and (b) standard deviation versus the normalized switch load, for static single class RPA in an 8 × 8 switch, under hot-spot traffic load.

ON-OFF traffic are not very sensitive to the normalized switch load, and that for all traffic statistics, they remain bounded also for a load equal to 99% (see the right-most marker of each curve).

Moreover, it is interesting to observe that in the case of packet input traffic (black circle markers, cell delays shown as solid lines, packet delays as dashed lines), we obtain performance indices very close to those of batch input traffic (white circle markers); this means that forcing the contiguity of cells belonging to the same packet does not jeopardize switch performance. Bernoulli and ON-OFF input traffic provides better, and very similar, performance; increasing the load makes the two input traffic patterns almost identical, and as a consequence, performance results tend to become closer. Finally, it is worth noting that standard deviation values are comparable with means.

In Fig. 2, we present the same performance indices for the hot-spot loading pattern. Curves show the performance obtained either when cells are directed toward the hot-spot output (dashed lines) or toward one of the other outputs (solid

lines). Since the horizontal axis refers to the normalized load of the switch as a whole, and the load of the hot-spot output is twice that of normal outputs, the curves referring to the hot spot saturate at load 0.5625 (this is the value of input port load that gives 1 when multiplied by 8 to obtain the total load of the switch, and further multiplied by 2/9 to account for the fact that the hot-spot output receives twice the load of the other seven output ports).

Again, we see that the performance obtained with Bernoulli input traffic is best, somewhat better than that yielded by ON-OFF input traffic and much better than the performance obtained with either batch or packet input traffic. These numerical results lead us to the conclusion that also with the hot-spot loading pattern, the packet-mode operations of RPA yield very similar performance to the cell-mode operations, even providing a beneficial impact on the performance indices when considering a normalized load higher than 0.5.

In Fig. 3, we present the worst (circle markers) and best (square markers) cell access delay curves for Bernoulli (white markers) and packet (black markers) input traffic, with the

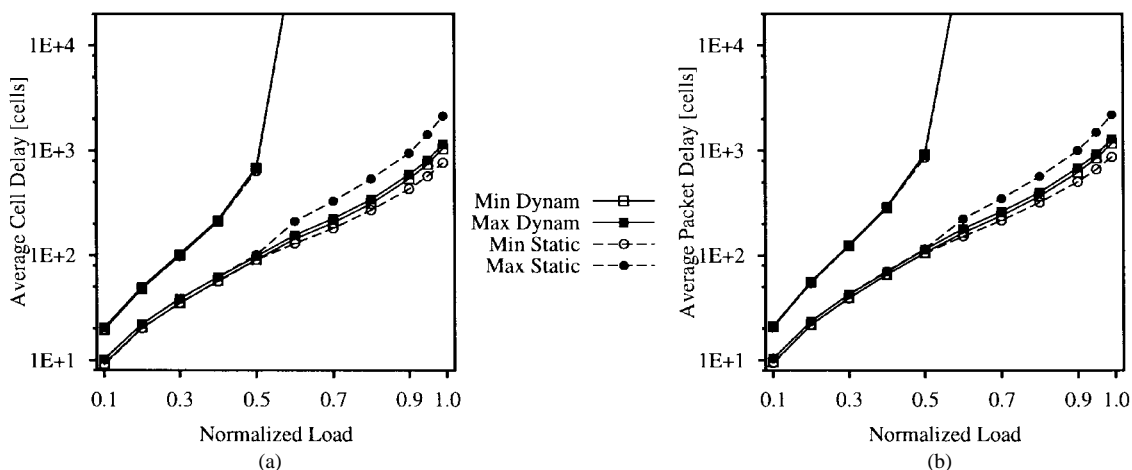


Fig. 4. (a) Worst and best cell and (b) packet access delays: averages and standard deviation versus the normalized switch load, for static and dynamic RPA in an  $8 \times 8$  switch, under hot-spot packet traffic load.

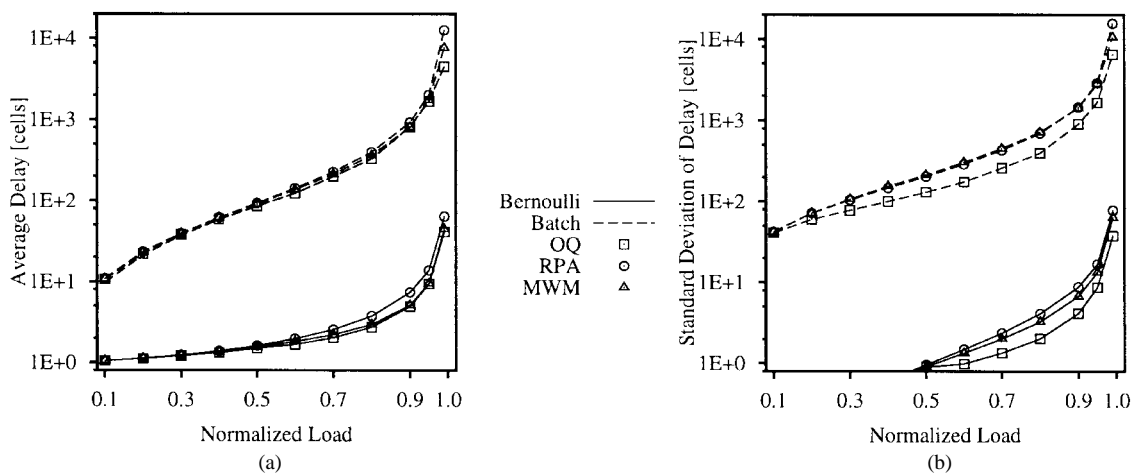


Fig. 5. Comparison among dynamic RPA, output buffer and MWM: (a) average and (b) standard deviation of cell access delays versus the normalized switch load, in an  $8 \times 8$  switch, under uniform traffic load.

hot-spot loading pattern. The differences between the best and worst performance are not very significant, for both the mean and the standard deviation.

In Fig. 4, we present the worst and best cell and packet access delays for packet input traffic in the hot-spot loading pattern for both the static and dynamic RPA versions. The dynamic version allows a somewhat better fairness to be obtained, achieving only marginally worse performance when considering the hot-spot output. It is fair mentioning that even with the dynamic RPA, some slight unfairness among the switch input ports still exists.

Finally, it must be emphasized that no throughput limitation was observed for any configuration, and no cell losses were experienced, as long as the load of all input and output ports was kept less than 1. This cannot be completely ascribed to the rather large buffer sizes that we used in the simulation experiments, and it is one of the main results of RPA (see [9]). Indeed, different scheduling algorithms were observed to produce both throughput limitations and cell losses with the same buffer sizes. However, it must be recognized that the behavior of RPA (like that of more complex switch algorithms [8]) becomes quasi-optimal (in the sense that it can guarantee a

full utilization of the switch bandwidth) only under asymptotic conditions, i.e., with infinite queue capacities.

### B. Single-Class RPA versus Output Queuing and MWM

In this section, we compare the performance achieved by  $8 \times 8$  switches adopting either the dynamic RPA version, or the output queuing architecture, or the MWM algorithm of [8].

Figs. 5 and 6 refer, respectively, to uniform and hot-spot traffic loads; in both cases, we consider Bernoulli (solid lines) as well as batch (dashed lines) input traffic. Results for dynamic RPA are presented with round markers, results for output queuing as square markers, and results for MWM as triangular markers.

In both figures, we see that the performance achieved with the three algorithms is quite similar. However, as expected, when differences become noticeable, we see that output queuing performs best, MWM is second, and RPA performs worst. However, the small performance differences are largely compensated for by the simplicity of RPA.

Moreover, we see that for all algorithms, the performance with batch input traffic is much worse than with Bernoulli in-

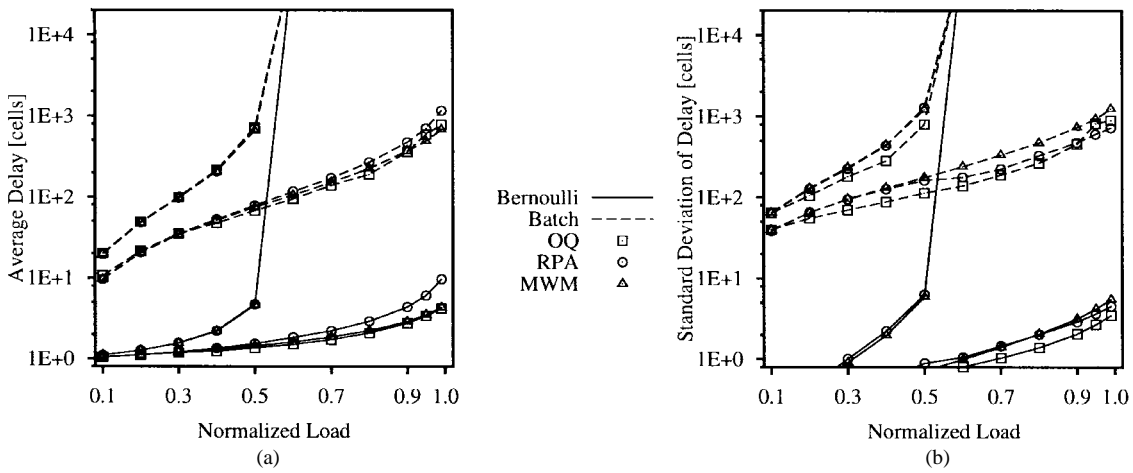


Fig. 6. Comparison among dynamic RPA, output buffer and MWM: (a) average and (b) standard deviation of cell access delays versus the normalized switch load, in an  $8 \times 8$  switch, under hot-spot traffic load.

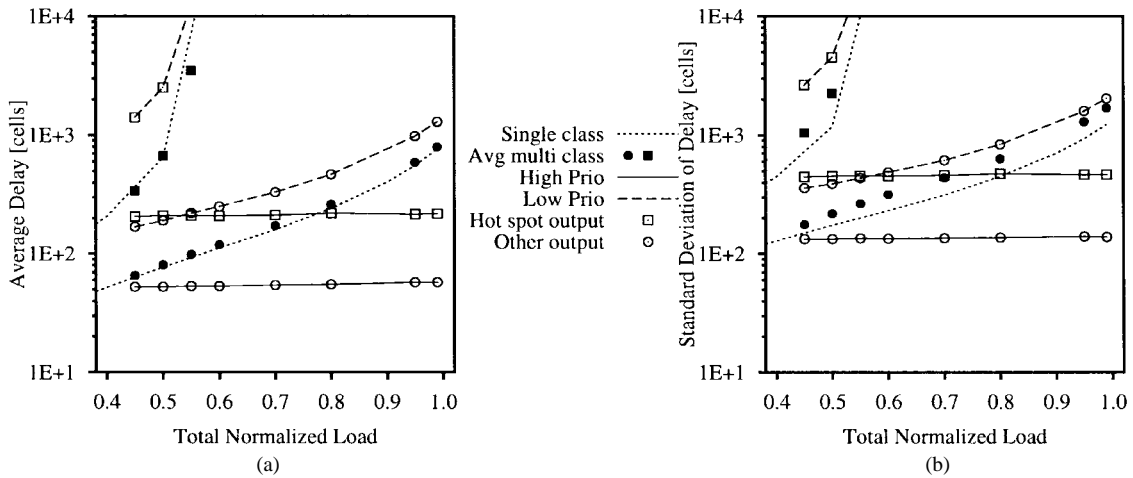


Fig. 7. Dynamic RPA with two different traffic classes: (a) average and (b) standard deviation of cell access delays versus the normalized switch load, in an  $8 \times 8$  switch, under hot-spot batch traffic load.

put traffic (the ratio between averages and standard deviations of cell access delays are close to one hundred).

C. Results for P-RPA: Multiclass RPA with Strict Priority

The performance achievable with multiclass RPA with strict priority is illustrated by the curves in Fig. 7 that refer to average and standard deviation of cell access delays in an  $8 \times 8$  switch with hot-spot batch traffic load using dynamic RPA with two different traffic classes. The low-priority curves are plotted with solid lines, while the high-priority curves are plotted with dashed lines. Square markers refer to the hot-spot output, whereas circular markers refer to lightly loaded output ports. The high-priority normalized load is always equal to 0.4, and the low-priority normalized load is increased from 0 to 0.6, so that the total normalized load varies in the range from 0.4 to 1.0, as indicated by the horizontal axis. The number of reservation rounds is  $R = 2$ .

Numerical results clearly show that the high-priority traffic performance is insensitive with respect to changes in the low-priority traffic load, as desired. Instead, as expected, the performance of low-priority cells heavily depends on the total switch load.

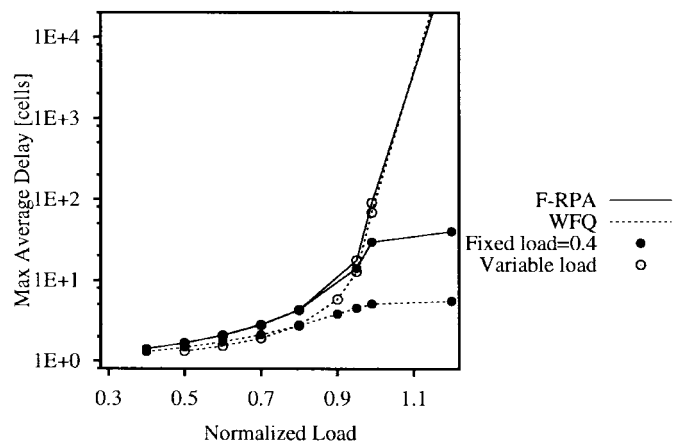


Fig. 8. F-RPA with one fixed load (0.4) traffic class and one variable load (0.0-0.8)—average cell access delays versus the normalized switch load, in an  $8 \times 8$  switch, under uniform Bernoulli traffic load.

We have also reported the delay mean and standard deviation, averaged over the two priorities in the two-class scenario (black markers) for comparison with the single-class scenario (dotted lines). It can be seen that RPA manages efficiently the multiclass scenario with strict priority, since

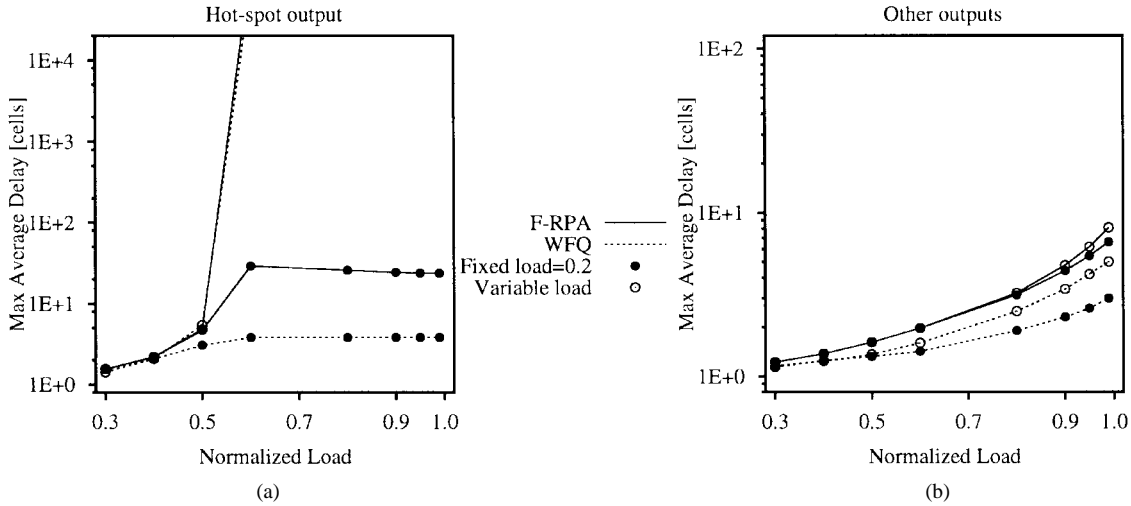


Fig. 9. F-RPA with (a) one traffic class with fixed load (0.2) and (b) one traffic class with variable load (0.1–0.8): average cell access delays versus the normalized switch load, in an  $8 \times 8$  switch, under hot-spot Bernoulli traffic load.

TABLE I  
DYNAMIC RPA WITH TWO TRAFFIC CLASSES AND BERNOULLI INPUT TRAFFIC, WITH A VARIABLE NUMBER  $R$  OF RESERVATION CYCLES

	R=1	R=2	R=4	R=8
Uniform Traffic Load				
Low prio. delay	4.9E+5	263.3	223.6	221.7
High prio. delay	3.27	3.05	3.05	3.05
Hot Spot Traffic Load				
Low prio. hot spot delay	70.9	68.6	68.6	68.6
High prio. hot spot delay	2.22	2.22	2.22	2.22
Low prio. delay	3.26	2.67	2.67	2.67
High prio. delay	1.35	1.35	1.35	1.35

TABLE II  
DYNAMIC RPA WITH TWO TRAFFIC CLASSES AND BATCH INPUT TRAFFIC, WITH A VARIABLE NUMBER  $R$  OF RESERVATION CYCLES

	R=1	R=2	R=4	R=8
Uniform Traffic Load				
Low prio. delay	2.1E+6	3.6E+5	41544	38131
High prio. delay	418.8	300.5	293.0	292.8
Hot Spot Traffic Load				
Low prio. hot spot delay	13028	12033	12093	12073
High prio. hot spot delay	213.1	208.9	208.7	208.7
Low prio. delay	361.0	221.2	220.1	219.3
High prio. delay	60.9	53.4	53.5	53.1

the means averaged over the two classes are quite close to the means obtained with just one class. Standard deviations instead are obviously higher, as expected when enforcing priorities.

Finally, in Tables I and II, we report average delays versus the number of reservation cycles, for several traffic patterns. For uniform traffic, the total load is 0.99, 0.72 being the load of high-priority traffic and 0.27 of low-priority traffic. For hot-spot traffic, the total load is 0.55, 0.40 being the load of high-priority traffic and 0.15 of low-priority traffic. Results show that a significant average delay reduction is obtained

by increasing the number of reservation cycles from 1 to 2. A marginal improvement is observed under uniform traffic pattern for  $R = 4$ , while increasing  $R$  to 8 does not provide additional benefits.

*D. Results for F-RPA: Multiclass RPA with Fair Bandwidth Allocation*

We now study the performance of F-RPA with two traffic classes and two different traffic patterns: uniform and hot-spot load.

In the first scenario, at each input port, cells belonging to both classes arrive according to a Bernoulli process with parameter  $p_c$ ,  $c = 1, 2$ , with a uniform distribution toward output ports. The load of the first traffic class is kept constant ( $p_1 = 0.4$ ), whereas the load of the second class  $p_2$  ranges from 0.0 to 0.8. A nominal bandwidth  $r_f = 0.5$  is assigned to each flow  $f$  at any input port  $i$  for both traffic classes; as a consequence, a nominal load equal to 1 is assigned at any output port. In this scenario, traffic flows of class 1 represent well-behaved sources, whereas traffic flows of class 2 represent sources trying to exploit more than their reserved fair share (for  $p_2 > 0.5$ ).

Results not reported show that a good bandwidth separation is achieved by F-RPA. However, a significant penalty has to be paid in terms of delay protection of well-behaved flows with respect to badly-behaved flows, as can be observed comparing the two curves with black circle markers in Fig. 8.

In the hot-spot scenario, the load of the first class is kept constant,  $p_1 = 0.2$ , whereas the load of the second class,  $p_2$ , ranges from 0.1 to 0.8. Also, in this case, a nominal bandwidth equal to 0.5 is assigned to each flow.

In the left-hand plot of Fig. 9, the average delay of cells directed to the hot spot is shown, while the right-hand picture presents the average delay for cells directed to other outputs. Results show that also in this case, a good bandwidth separation can be achieved with F-RPA.

As a final observation, recall that WFQ algorithms in general are aimed at providing bandwidth separation, which we indeed

obtain successfully, not at delay guarantees; however, the increase in the average delay due to badly-behaved sources is not a nice property of F-RPA. Still, the fact that F-RPA exhibits acceptable behavior in the case of unbalanced switch loads, coupled with the observation that traffic hardly ever is uniform within switches, is a point in favor of the applicability of the algorithm.

## VIII. CONCLUSION

RPA, a scheduling algorithm for input buffered switches, was described and evaluated, showing that it can provide performance close to those of optimal algorithms.

RPA is simple (as simple as the simplest among previous proposals of efficient input queuing scheduling algorithms), efficient, and flexible, allowing the support of different traffic classes and packet-mode operations.

The effectiveness of RPA was assessed with detailed simulations in uniform, as well as unbalanced, traffic conditions.

RPA is capable of supporting the uninterrupted flow of cells belonging to one packet with marginal performance penalties with respect to the case in which cells of different packets are multiplexed within the switch.

RPA has been shown to be able to deal with multiple traffic classes, enforcing a strict priority among classes, without a significant performance degradation. A limited number of reservation cycles is sufficient to obtain good overall performance.

RPA has been adapted to provide bandwidth guarantees to traffic flows; whereas a good bandwidth separation has been successfully obtained, performance results in terms of delay are less satisfactory if compared to fair queuing algorithms applied at output ports in output buffered switches. This remains an interesting and open field for further research.

Finally, we proved that the worst-case weight of any matching obtained with RPA is larger than half the weight of the corresponding matching generated by the MWM algorithm.

## REFERENCES

- [1] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queueing on a space division switch," *IEEE Trans. Commun.*, vol. COM-35, pp. 1347–1356, Dec. 1987.
- [2] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *J. Internetworking: Res. Experience*, vol. 1, no. 1, pp. 3–26, Oct. 1990.
- [3] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. IEEE INFOCOM'94*, Toronto, ON, Canada, June 1994, vol. 2, pp. 636–646.
- [4] M. Karol, K. Eng, and H. Obara, "Improving the performance of input-queued ATM packet switches," in *Proc. IEEE INFOCOM'92*, Firenze, Italy, May 1992, vol. 1, pp. 110–115.
- [5] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High speed switch scheduling for local area networks," *ACM Trans. Comput. Syst.*, vol. 11, no. 4, pp. 319–352, Nov. 1993.
- [6] N. McKeown, P. Varaiya, and J. Walrand, "Scheduling cells in an input-queued switch," *Electron. Lett.*, vol. 29, no. 25, pp. 2174–2175, Dec. 1993.
- [7] M. Chen, N. D. Georganas, and O. W. W. Yang, "A fast algorithm for multi-channel/port traffic assignment," in *Proc. IEEE ICC'94*, New Orleans, LA, May 1994, vol. 1, pp. 96–100.
- [8] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," in *Proc. IEEE INFOCOM'96*, San Francisco, CA, Mar. 1996, vol. 1, pp. 296–302.

- [9] M. Ajmone Marsan, A. Bianco, and E. Leonardi, "RPA: A simple, efficient and flexible policy for input buffered ATM switches," *IEEE Commun. Lett.*, vol. 1, pp. 83–86, May 1997.
- [10] M. Ajmone Marsan, A. Bianco, E. Leonardi, and L. Milia, "Quasi-Optimal algorithms for input buffered ATM switches," in *Proc. IEEE ISCC'98 Workshop*, Athens, Greece, June 1998, pp. 336–342.
- [11] R. LaMaire and D. Serpanos, "Two dimensional round-robin schedulers for packet switches with multiple input queues," *IEEE/ACM Trans. Networking*, vol. 2, pp. 471–482, Oct. 1994.
- [12] N. McKeown and A. Mekkittikul, "A starvation free algorithm for achieving 100% throughput in an input queued switch," in *Proc. ICCCN'96*, Rockville, MA, Oct. 1996, pp. 226–229.
- [13] H. Duan, J. Lockwood, S. Kang, and J. Will, "A high performance OC12/OC48 queue design prototype for input buffered ATM switches," in *Proc. IEEE INFOCOM 97*, Kobe, Japan, Mar. 1997, vol. 1, pp. 20–28.
- [14] N. McKeown and A. Mekkittikul, "A practical scheduling algorithm to achieve 100% throughput in input-queued switches," in *Proc. IEEE INFOCOM'98*, San Francisco, CA, Apr. 1998, vol. 2, pp. 792–799.
- [15] M. Ajmone Marsan, A. Bianco, E. Filippi, P. Giaccone, E. Leonardi, and F. Neri, "On the behavior of input queuing switch architectures," *Eur. Trans. Telecommun.*, vol. 10, no. 2, pp. 111–124, Mar./Apr. 1999.
- [16] D. Stiliadis and A. Varma, "Providing bandwidth guarantees in an input-buffered crossbar switch," in *Proc. IEEE INFOCOM'95*, Boston, MA, Apr. 1995, vol. 3, pp. 960–968.
- [17] J. W. Lockwood, "Design and implementation of a multicast, input-buffered ATM switch for the iPOINT testbed," Ph.D. dissertation, Univ. of Illinois at Urbana-Champaign, 1995.
- [18] R. Ahuja, B. Prabhakar, and N. McKeown, "Multicast scheduling for input-queued switches," *IEEE J. Select. Areas Commun.*, vol. 15, pp. 855–866, May 1996.



**Marco Ajmone Marsan** (S'76–M'78–SM'86–F'99) received the Dr. Ing. degree in electronic engineering from Politecnico di Torino, Torino, Italy, and the M.S. degree from the University of California at Los Angeles (UCLA).

Currently, he is a full Professor at the Electronics Department of Politecnico di Torino, where from November 1975 to October 1987, he was first a Researcher and then an Associate Professor. From November 1987 to October 1990, he was a full Professor at the Computer Science Department, University of Milan, Milan, Italy. During the summers of 1980 and 1981, he was with the Research in Distributed Processing Group, Computer Science Department, UCLA. During the summer of 1998, he was an Erskine Fellow at the Computer Science Department of the University of Canterbury, New Zealand. He has co-authored over 200 journal and conference papers in the areas of communications and computer science, as well as the books, *Performance Models of Multiprocessor Systems* (Boston, MA: MIT Press, 1986) and *Modelling with Generalized Stochastic Petri Nets* (New York, Wiley, 1995). His current research interests include the performance evaluation of communication networks and their protocols.

Prof. Marsan received the Best Paper Award at the Third International Conference on Distributed Computing Systems, in 1982.



**Andrea Bianco** (M'99) was born in Torino, Italy, in 1962. He received the Dr. Ing. degree in electronics engineering, in 1986, and the Ph.D. degree in telecommunications engineering, in 1993, both from the Politecnico di Torino, Torino, Italy.

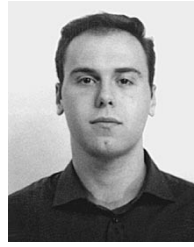
Since 1994, he has been an Assistant Professor at the Politecnico di Torino, first with the Dipartimento di Sistemi di Produzione and later with the Dipartimento di Elettronica. In 1993, he worked with Hewlett-Packard Labs, Palo Alto, CA. His current research interests include the fields of protocols for all-optical networks and switch architectures for high-speed networks.



**Emilio Leonardi** received the Dr. Ing. degree in electronics engineering, in 1991, and a Ph.D. degree in telecommunications engineering, in 1996, both from the Politecnico di Torino, Torino, Italy.

In 1995, he spent one year with the Computer Science Department at the University of California at Los Angeles, where he was involved in the Supercomputer-SuperNet (SSN) project. Currently, he is an Assistant Professor with the Electronics Department at Politecnico di Torino. His research interests include the fields of all-optical networks,

high-speed wormhole routing networks, and high-speed switches.



**Luigi Milià** received the Dr. Ing. degree in electronics engineering in 1998 from the Politecnico di Torino, Torino, Italy.

In 1998, he joined the Software Development Department as a Researcher at Centro Ricerche Fiat, Orbassano, Italy. His research interests include real-time software for automotive embedded systems, from fuel injections systems to on-vehicle data acquisition and communications systems.