RPA: a Simple, Efficient and Flexible Policy for Input Buffered Switches

RPA: A Simple, Efficient, and Flexible Policy for Input Buffered ATM Switches

M. G. Ajmone Marsan, *Senior Member, IEEE,* A. Bianco, and E. Leonardi

***Abstract*—Reservation with Preemption and Acknowledgment (RPA) is a simple, efficient, and flexible queuing discipline and scheduling algorithm for input buffered asynchronous transfer-mode (ATM) switches. This letter describes the RPA algorithms, and presents simulation results to demonstrate the effectiveness of the proposed approach.**

I. Introduction

THE DESIGN of asynchronous transfer-mode (ATM) switches rests on three main elements: 1) the *switching fabric*, i.e., the interconnection architecture used to transfer cells from input ports to output ports; 2) the *scheduling algorithm* used to arbitrate requests of cells that arrive at different input ports, but must reach the same output port; and 3) the *queuing discipline* at input and/or output ports, i.e., the rule for the selection of the next cell to be handled.

ATM switch designs with output buffers have been very popular in recent years, mainly because they are not prone to the well-known Head of the Line (HoL) blocking that may adversely affect the performance of input buffered switches.

However, a negative characteristic inherent in output buffered switches is the necessity for an internal cell transfer rate N times higher than the external link speed, being that N is the number of input (and output) ports.

Since data rates on point-to-point fiber links keep growing very rapidly and the number of ports of ATM switches follows a similar trend, providing the cell transfer rates required for output buffered designs within switching fabrics is becoming a problem. Researchers have thus started reconsidering input buffered ATM switch designs.

In order to either reduce or completely overcome the penalty introduced by the HoL blocking, that can limit the maximum achievable throughput to 60% of the output channel speed under uniform traffic conditions [1], 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 ATM switch essentially depends on its queuing discipline and scheduling algorithm. Thus it is very important to search for simple but efficient policies[1] that must also be able to: 1) deal with the requirements of different traffic classes; 2) provide the means to give priority to cells belonging to classes of traffic with more stringent Quality of Service (QoS) requirements; and 3) support multicast addressing.

Several policies were presented in the technical literature to address some of these issues (mostly trying to overcome HoL blocking) in input-buffered switches under uniform traffic conditions [2]–[5]. 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 policies 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.

The first proposals of policies that lead to the optimal and fair exploitation of the switch bandwidth under every traffic pattern appeared very recently (see the maximum weighted matching policies of [6] and [7]). Unfortunately, these policies 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 ATM switches.

In this letter, we propose a novel policy whose computational complexity is only $O(N^2)$ and that leads to an efficient and fair exploitation of the switch bandwidth under all investigated load patterns that include several critical traffic conditions; at the same time, it can be extended to deal with different traffic classes and multicast traffic (however, these aspects are not addressed in this letter).

II. The RPA Policy

RPA stands for Reservation with Preemption and Acknowledgment. The name indicates that the queuing discipline and scheduling algorithm that we describe next are based on a reservation round where sources can indicate their most urgent cell transfer needs, possibly overwriting less urgent requests by other sources, and an acknowledgment round to allow sources to determine what cell they can actually transmit. 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 to a minimum.

Manuscript received January 13, 1997. This work was supported in part by a research contract between Politecnico di Torino and CSELT, in part by the EC through the Copernicus Project 1463 ATMIN, and in part by the Italian Ministry for University and Research. The associate editors coordinating the review of this letter and approving it for publication were Dr. C. Siller and Prof. Y. Bar-Ness.

The authors are with the Dipartimento di Elettronica, Politecnico di Torino, 10129 Torino, Italy (e-mail: ajmone@polito.it).

Publisher Item Identifier S 1089-7798(97)04608-5.

[1]The term *policy* in this letter refers to the specific combination of queuing discipline and scheduling algorithm.

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 the kth input and output ports by i_k and o_k, respectively,

RPA requires, at each input port, the availability of N separate first in–first out (FIFO) queues for the storage of packets directed to different output ports. More precisely, the jth queue, denoted by Q_j, contains cells directed from the considered input port toward o_j.

In the description of RPA we concentrate on the case of a single traffic class.

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 $0 \leq j < N-1$, that orderly refer to output ports [RES(j) refers to o_j].

Each RES(j) element contains three fields:

RES $(j).port\_id$ the address of the input port trying to reserve a cell transmission to o_j;

RES $(j).urg$ the urgency (by convention 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; multiple classes of traffic can be managed with clever definitions of the urgency;

RES $(j).busy$ is set to one 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 pre-established 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_j) 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

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

where $U(Q)$ is the urgency of the cell at the head of queue Q. 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 $U(Q_x)$ 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 $c = |b+1|_N$, that 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 one. 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 zero), 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 one 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 one. 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 one.

This acknowledgment algorithm is executed in order 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 change; we call static the former version of the scheduling policy, and adaptive the latter. In this letter, for the sake of brevity, we do not consider adaptive policies, but a number of those can be devised: we could envision a round-robin selection of the starting point of the cyclic rounds, a random selection of the starting point of the cyclic rounds, or a completely random order. Each one of these alternatives probably has pros and cons, but a careful investigation of those is left for future work. Of course, the static policy 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.

Several metrics can be adopted to quantify the cell urgency. When just one class of traffic flows is present in the switch, the number of cells stored in each input queue can be chosen as the urgency for the cell at the head of each queue; this is the urgency metric we used in our simulation experiments. When several classes of traffic flows are present, no significant modification to our scheduling policy is required,

and a straightforward generalization is sufficient; however, the definition of a suitable urgency metric is crucial for the support of different types of QoS requirements. Moreover, in this case each input port may need a number of queues equal to $p \times N$ to efficiently deal with p traffic classes.

III. Complexity

The complexity of the RPA scheduling algorithm is not difficult to compute.

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 $O(N^2)$.

IV. Simulation Results

We briefly present in this section a small sample of the simulation results that were obtained in the performance investigation of RPA; we focus on the static version of the scheduling policy, but the general trends of the algorithm behavior remain the same also when considering adaptive versions.

To begin with, in Fig. 1 we present average cell access delays versus the normalized load of the ATM switch, for the least favored input port. The cell access delay is 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 normalized load of the ATM switch is the fraction of slots that contain cells on all input ports. Results are obtained for an 8×8 switch configuration under uniform traffic conditions, with queuing capacity at each input port equal to either 100 or 10 000 cells per input queue (a total of either 800 or 80 000 cells per input port). We report average delays for three types of input traffic.

- *Bernoulli* input traffic (white square markers): cells arrive at input ports according to a Bernoulli process; the cell output ports are selected with random, independent choices; the queue capacity is set to 100 cells for each input queue.
- *batch* input traffic—case 1 (white round markers): cells continuously arrive at input ports during geometrically distributed ON periods, whose average duration is ten cells; no cells arrive during geometrically distributed OFF periods, whose average duration is selected so as to obtain the desired load (that cannot exceed $10/11$ on the average, since the OFF period cannot be shorter than one cell); the cell output port is the same for all cells arriving during one ON period, and is selected by random choice at the beginning of the batch; the queue capacity is set to 10 000 cells for each input queue.
- *batch* input traffic—case 2 (white triangular markers): same as case 1 except that the average duration of ON periods is 100 cells, so that the maximum achievable load grows to $100/101$.

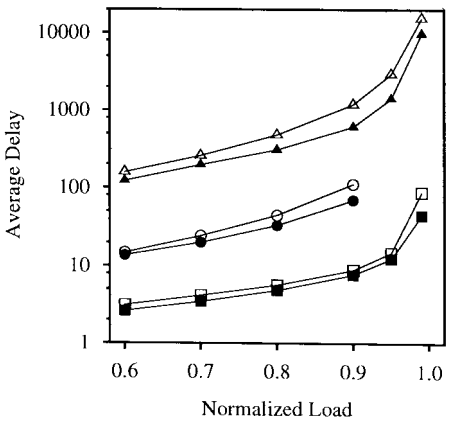

Fig. 1. Average cell access delays for the static scheduling policy.

TABLE I
First Traffic Scenario

	o_w	o_x	o_y
i_a	λ		
i_b		$1-\lambda$	
i_c	$1-\lambda$	λ	

Black markers with the same shape show the "optimal" performance that could be obtained with the same input traffic pattern in an output buffered switch with infinite queue lengths.

Three main observations can be drawn from the simulation results: 1) no throughput limitation can be observed in any of the presented scenarios; 2) no losses were experienced; 3) the presented delays are quite close to the "optimal" values. Note that even more complex scheduling policies (see [6], [7]) guarantee a full utilization of the switch bandwidth only under asymptotic conditions, i.e., with infinite queue capacities.

The performance of RPA was studied also under traffic scenarios similar to those described in [6], [7]. Such conditions can be easily proved to induce throughput limitations in all of the simple scheduling techniques proposed up to now (to the best of our knowledge) and also in the maximum size matching algorithm described in [6], whose complexity is $O(N^{5/2})$. In order to describe these input traffic patterns, we use a matrix notation in Tables I–IV; rows are associated with input ports, columns with output ports; we report in the table the amount of traffic transmitted on average from the input port toward the output port.

We run simulations with different values of λ, scaling the input load so as to avoid overload (in practice, the values

TABLE II
SECOND TRAFFIC SCENARIO

	o_w	o_x	o_y
i_a	λ	$1-\lambda$	
i_b		λ	$1-\lambda$
i_c	$1-\lambda$		λ

TABLE III
THIRD TRAFFIC SCENARIO

	o_w	o_x	o_y	o_z
i_a	λ	$1-\lambda$		
i_b			$1-\lambda$	λ
i_c	$1-\lambda$		λ	
i_d		λ		$1-\lambda$

shown in the table were multiplied by 0.999), observing no throughput limitation. This confirms our conjecture that the static version of RPA is close to being optimally efficient, in the sense that, provided that: 1) the normalized offered load of each input port is smaller than one and 2) the sum of the normalized offered loads toward each output port is smaller than 1, all of the input traffic can be successfully transferred to output ports.

TABLE IV
FOURTH TRAFFIC SCENARIO

	o_w	o_x	o_y	o_z
i_a	λ	$1-\lambda$		
i_b		λ	$1-\lambda$	
i_c			λ	$1-\lambda$
i_d	$1-\lambda$			λ

REFERENCES

[1] M. Karol, M. Hluchyj, and S. Morgan, "Input versus output queuing on a space division switch," *IEEE Trans. Commun.*, vol. COM-35, pp. 1347–1356, Dec. 1987.

[2] M. Karol, K. Eng, and H. Obara, "Improving the performance of input-queued ATM packets switches," presented at the IEEE INFOCOM'92, Firenze, Italy, May 1992.

[3] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High speed switch scheduling for local area networks," presented at the 5th Int. Conf. Arch. Support for Programming Languages and Operating Systems, Oct. 1992.

[4] N. McKeown and P. Varaiya, "Scheduling cells in an input-queued switch," *Electron. Lett.*, vol. 29, no. 25, pp. 2174–2175, Dec. 1993.

[5] M. Chen, N. D. Georganas, and O. W. W. Yang, "A fast algorithm for multi-channel/port traffic assignment," presented at the IEEE ICC'94, New Orleans, LA, May 1994.

[6] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," presented at the IEEE INFOCOM'96, San Francisco, CA, Mar. 1996.

[7] A. Mekkittikul and N. McKeown, "A starvation-free algorithm for achieving 100% throughput in an input-queued switch," presented at the ICCCN'96, Rockville, MD, Oct. 1996.