.

FORTH-ICS /TR-325 October 2003

Design Issues of Variable-Packet-Size, Multiple-Priority Buffered Crossbars

Nikos Chrysos

ABSTRACT

Buffered crossbars provide several advantages over unbuffered ones; one of them is that they can directly switch variable-size packets. The first part of this report considers several issues that arise when designing a variable-packet-size buffered crossbar with considerable Round-Trip Time: (1) we observe that an acknowledgment semantics for credits reduces the backpressure communication overhead and the storage of credits within the crossbar chip; (2) we show that cut-through can be implemented at the crosspoints and we discuss for a reasonable crosspoint buffer size under cut-through and store&forward; (3) we present a novel architecture for scheduling the operations at the contention points. The second part is concerned with the crosspoint queuing organization when handling multiple priorities. In buffered crossbars, separate buffers/queues (or lanes) per priority at each crosspoint are required to prevent HOL blocking and buffer hogging. In this report we propose a buffered crossbar that effectively supports *multiple* priorities with only *two* such lanes per crosspoint, even with no internal speed-up. Our method maps packets to crosspoint lanes using adaptive criteria, and dynamically changes the effective priority of crosspoint queues. Through simulations we show that when eight priorities are supported, under a typical, uniform traffic pattern, our method will not increase the average delay of any priority by more than **15** percent, compared to the aforementioned ideal system, that requires **4** times more buffering. Even under a highly irregular traffic pattern, the respective discrepancy is bellow **75** percent. Under a non-uniform/imbalanced traffic pattern, the lower priority levels have considerably increased average delay, but this is due to inefficiencies of the round-robin scheduling that we currently employ, and is also related to crosspoint buffers dimensioning; it is not a matter of priority handling. We also compare variable-size-packet buffered crossbars to fixed-cell ones; through simulations we verify that the latter need significant speed-up to reach the formers' performance. Finally, we discuss the complexity of our methods, and we show that it only affects the ingress line-cards.

Design Issues of Variable-Packet-Size, Multiple-Priority Buffered Crossbars

Nikos I. Chrysos[1,2]

Computer Architecture & VLSI Systems (CARV) Laboratory,
Institute of Computer Science(ICS)
Foundation for Research and Technology — Hellas(FORTH)
Science and Technology Park of Crete
P.O. Box 1385, Heraklion, Crete, GR-711-10 Greece
email: nchrysos@ics.forth.gr

Technical Report FORTH-ICS/TR-325 — October 2003

Work Performed as part of Doctoral Dissertation at the Depart. of Computer Science, Univ. of Crete, under the supervision of prof. Manolis Katevenis

Keywords: Variable Packet Size Buffered Crossbar, Switch Design, Multiple Priorities, Credit Handling, HOL blocking, Buffer Hogging, Adaptive Mapping, Dynamical Priorities

[1]ICS-FORTH, P.O. Box 1385, GR-711-10 Heraklion, Crete, Greece. E-mail: nchrysos@ics.forth.gr
[2]Department of Computer Science, University of Crete, Heraklion, Crete, Greece.
E-mail: nchrysos@csd.uch.gr

Design Issues of Variable-Packet-Size, Multiple-Priority Buffered Crossbars

Nikos Chrysos, Computer Science Department,
ICS-FORTH, Technical Report 325, October 2003
http://archvlsi.ics.forth.gr/bufxbar

Abstract*— Buffered crossbars provide several advantages over unbuffered ones; one of them is that they can directly switch variable-size packets. The first part of this report considers several issues that arise when designing a variable-packet-size buffered crossbar with considerable Round-Trip Time: (1) we observe that an acknowledgment semantics for credits reduces the backpressure communication overhead and the storage of credits within the crossbar chip; (2) we show that cut-through can be implemented at the crosspoints and we discuss for a reasonable crosspoint buffer size under cut-through and store&forward; (3) we present a novel architecture for scheduling the operations at the contention points. The second part is concerned with the crosspoint queuing organization when handling multiple priorities. In buffered crossbars, separate buffers/queues (or lanes) per priority at each crosspoint are required to prevent HOL blocking and buffer hogging. In this report we propose a buffered crossbar that effectively supports *multiple* priorities with only *two* such lanes per crosspoint, even with no internal speed-up. Our method maps packets to crosspoint lanes using adaptive criteria, and dynamically changes the effective priority of crosspoint queues. Through simulations we show that when eight priorities are supported, under a typical, uniform traffic pattern, our method will not increase the average delay of any priority by more than 15 percent, compared to the aforementioned ideal system, that requires 4 times more buffering. Even under a highly irregular traffic pattern, the respective discrepancy is bellow 75 percent. Under a non-uniform/imbalanced traffic pattern, the lower priority levels have considerably increased average delay, but this is due to inefficiencies of the round-robin scheduling that we currently employ, and is also related to crosspoint buffers dimensioning; it is not a matter of priority handling. We also compare variable-size-packet buffered crossbars to fixed-cell ones; through simulations we verify that the latter need significant speed-up to reach the formers' performance. Finally, we discuss the complexity of our methods, and we show that it only affects the ingress line-cards.

1 . Introduction

Switches with an increasing number of faster ports are needed to compensate for the ever increasing demand for network bandwidth. At the same time, mechanisms are sought for higher sophistication in quality of service (QoS) guarantees. The crossbar is the simplest and most popular organization for high performance (internally non-blocking) switches. It is the architecture of choice for up to several tens of ports, although for higher port counts, N, the growth rate of the crossbar cost, O (N^2), makes alternative topologies more attractive. The crossbar is also, commonly, the building block for switching fabrics with higher port counts.

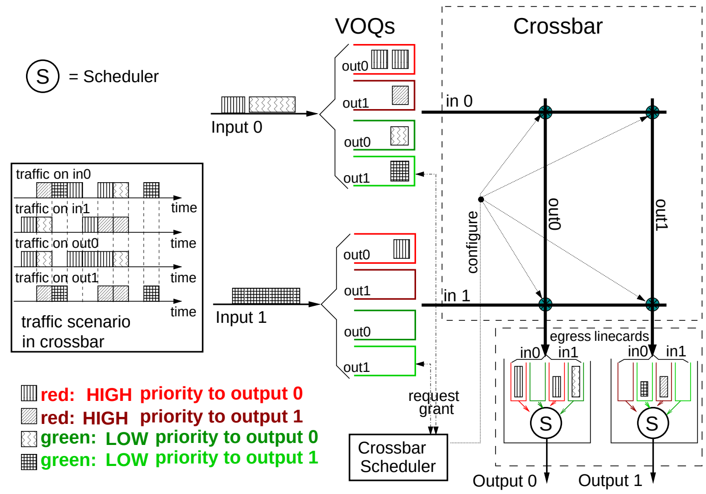

Fig. 1. The CIOQ switch with unbuffered crossbar

1.1. Unbuffered Crossbars

The crossbar scheduling problem, inherent in all unbuffered crossbar architectures, consists of selecting a conflict-free match (i.e., a full or partial permutation) of inputs to outputs. This match, which is normally computed in every time-slot (i.e., cell-time), can only be found in a centralized fashion, since when one input selects some output no other input is allowed to select the same output, and vice versa.

1) WRR Scheduling: Existing, practical crossbar schedulers ([1] [2] [3] [4]) cannot simultaneously support high crossbar-utilization and sophisticated Quality of Service, e.g. multiple priority levels or weighted round-robin (WRR) scheduling: when some connections are preferentially selected over others, multiple iterations are required to produce near maximal matches [5].

For instance, if we replace the round-robin (RR) arbiters in the iSLIP architecture by WRR arbiters, then, the matching algorithm will normally need multiple iterations to produce large matches. Under RR, the iSLIP algorithm guarantees that when the switch is flooded, a full match at time-slot t will be followed by a different full match at time-slot $t+1$, even with a single iteration of the algorithm: when the switch is flooded, all virtual-output-queues (or VOQs) become persistent and thus, all inputs request all outputs in the first phase of the iSLIP algorithm; if during time-slot t, the N output arbiters grant N different inputs (i.e., full match at time slot t), they will also grant N different input at time-slot $t+1$. (Given that the N round-robin arbiters use a common ordering of inputs when they perform round-robin.)

By contrast, the N "next-to-grant" pointers of the WRR output arbiters in the hypothetical WRR-iSLIP switch, will not necessarily move to positions that correspond to *different* inputs at time slot $t+1$, even if the match at the previous time-slot was full. For example, some WRR output arbiters may grant the same input in consecutive time-slots only because the corresponding edge has large weight, and consequently, it is possible that an input receives grands from multiple outputs at time-slot $t+1$. Thus, if this hypothetical switch employs only one iteration for producing matches, the crossbar utilization can be poor[1] [3]. This is one of the reasons explaining why practical crossbar schedulers either ignore QoS issues, or provide only RR scheduling, sometimes coupled with a pair of priority levels.

2) Speed-Up: The solution commonly used today is to provide significant internal speed-up, i.e., *combined-input-output queuing*, or CIOQ (fig. 1): the crossbar port rate is higher than line rate by a factor of f, considerably greater than one (e.g. two to three) [6]. In this way, (a) imperfect crossbar scheduling is acceptable, since an average utilization of $1/f$ for the crossbar outputs suffices for the egress lines to get fully utilized; (b) we can accommodate the rate increase that occurs when variable-size packets are segmented into fixed-size cells; and (c) the emphasis for QoS enforcement is shifted to the egress-line sub-system, since queues now tend to build up on the output side of the crossbar. Using the latter property, one can implement e.g. WRR or priority scheduling on the output queues, although, for traffic overloads higher than f, queues also build up on the input side, where crossbar schedulers cannot typically implement sophisticated disciplines.

3) Cost of Speed-Up: While internal speed-up is a good solution, it does incur significant cost: (1) the crossbar is more expensive since it must provide f times higher throughput, (2) the central-scheduler must run f times faster, (3) the buffer memories must provide $(1+f)/2$ times higher throughput, (4) and the number of buffer memories is doubled, since besides input queues, output queues are needed as well. (Note that output queues are also needed for cell-to-packet reassembly, and for sub-port demultiplexing, when provided.).

Concerning power, the peak consumption is significantly increased when speed-up is deployed, since most of the parts of the switch run at a higher frequency. Particularly, consider that the crossbar chip power consumption is often the limiting factor for the aggregate line rate that can supported by the system, and power consumption translates directly into (mostly I/O pins) throughput; I/O throughput increases linearly with speed-up. Thus, a router that uses a speed-up of two usually ends up providing only half of the aggregate line rate that it could otherwise offer. In overall, speed-up considerably increases the cost of all major parts of the switch and severely limits the maximum line rate that can be supported.

[1] A similar problem of buffered crossbars is presented in section 3.2

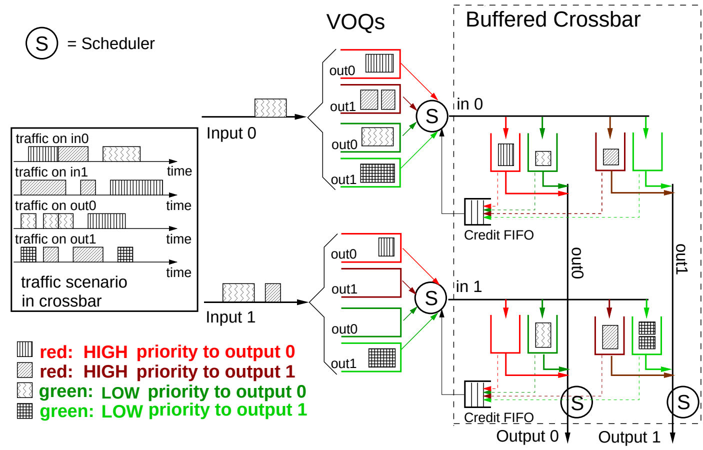

Fig. 2. A CICQ switch with a separate crosspoint lane for each priority level, i.e., the MQDB queuing organization –see section 4.1.1

1.2. Buffered Crossbars

An alternative solution, with the potential to yield both faster and less expensive switches, is to use *buffered crossbars*. The *combined-input-crosspoint-queuing* (CICQ, or buffered crossbar) architecture significantly simplifies scheduling [7] [8] [9] [10] [11]: the 2·N schedulers in CICQ switches – N at the input and N at the output lines of the crossbar– schedule *collectively* the traffic through the crossbar, but still, these work *independently* of each other since each of them deals with only a single resource. They are coordinated in longer timescales (i.e., a few to a few tens of time-slots), through the backpressure feedback from the crosspoint buffers. In buffered crossbars, WRR and strict priority scheduling can be implemented at each input and output contention point, without negatively affecting switch throughput [10]; thus no speed-up is required to compensate for scheduling inefficiencies.

1) WRR Scheduling: For instance, in [10] [11] [12] we demonstrate that a buffered crossbar with VOQs and no internal speed-up, which employs WFQ schedulers at the input and the crossbar-output lines, distributes bandwidth to flows –for the moment, consider them as distinct input/output pairs–, in a way that closely approximates the Weighted Max-Min (WMM) Fair allocation. Although WMM fairness does not always ascertains full line utilization, we observed that it does so in the common case when the weights of the flows are selected randomly. But even when a line is underutilized under WMM fairness, this happens by limiting the rate of some flows, which otherwise, would "steal" rate from equally congested flows.

We studied this behavior under persistent VOQ sources (i.e., either constantly full or constantly empty VOQs), under various distributions for the weights of the flows and under different buffer space at the crosspoints; using extensive simulations on a 32×32 cell-discrete system, we found that with 3-8 cells per crosspoint queue, the worst-case discrepancy to the WMM fair rates falls down to 5% while the average discrepancy is less than 1%. Note that there exist traffic arrival scenarios where the WFQ buffered crossbar does not stabilize very close to the WMM fair rates, unless large crosspoint buffers are employed –see for instance, arrivals with imbalanced destinations in [9].

Using a fluid model, we show in [12] that the time it takes to the WFQ buffered crossbar to convergence to WMM fairness is usually small; the worst-case stabilization delay after a change in the weight of some flow, or after a change in some VOQ (meaning the transition from empty to non-empty, or vice-versa), is proportional to the size of the crosspoint buffers, proportional to N, and inversely proportional to the magnitude of the change in bandwidth allocation.

2) Buffers Costs: Of-course, the major cost of buffered crossbars is that N^2 additional buffers are needed, one at each crosspoint. The minimum amount of buffering per separate crosspoint queue, required for good performance in a buffered crossbar operating on cells, is RTT×IR, where RTT stands for the Round-Trip time and IR stands for the internal line rate, i.e., speed-up times the line rate [13]; additional buffer space improves performance [10] [11].

By dedicating 100 mm^2 of a 0.13 μm ASIC to SRAM, 40 Mbits of buffer memory will become available, i.e., at 2.5 μm^2 per bit [14]. This suffices for a 32×32 buffered crossbar with four priority levels, employing a separate 1 KByte crosspoint buffer/queue for each priority level. This buffer space at each crosspoint queue is two times the RTT×IR, with RTT near 400ns (i.e., $\approx$ 20 meters), port rate 10 Gb/s and speed-up equal to 1.6×; this speed-up might be necessary if the system operates in a variable-packet-size network to compensate primarily for the segmentation overhead. A multiple chip implementation of a 64×64, fixed-size cell, buffered crossbar with speed-up 1.6×, supporting 40 Gb/s per port and large RTT is presented in [15].

3) Implementation Advantages: From the implementation point of view, an advantage of buffered crossbars is that they do not require clock or cell-time synchronization among the ingress lines. In an unbuffered crossbar, central arbitration at each cell time implies that the cells entering the crossbar chip through links operating in different clock domains have to be synchronized before being switched on the crossbar-output links, so that output conflicts do not occur due to discrepancies among the different clocks; normally, this requires to deploy elastic buffers at the input-side of the crossbar chip.

Instead, in the buffered crossbar architecture, output contention is resolved in the crosspoint queues which can operate asynchronously of each other; hence, no synchronization is required among the ingress and the cells entering the crossbar chip have to be synchronized only with the clock domain of their corresponding output link, before the output scheduler can considered them as eligible and transmit them on the output link. As we demonstrate in [16], this synchronization can be supported efficiently at the crosspoint buffers, if the latter are implemented by dual-ported SRAMs. Thus we eliminate the need of the elastic buffers. The only control that we synchronize with the clock of the output link is a *new-packet* signal, which notifies the output scheduler for the arrival of a new packet at a crosspoint queue.

Observe that the distributed architecture of a buffered crossbar, and the independent/asynchronous operation of the parts that compromise it, also enhance fault-tolerance and substitutability. While the market demand shifts from switches of multi-Gb/s to multi-Tb/s aggregate capacity, the requirement for distributed and asynchronous operation becomes even more pervasive since the new generation switches are built over multiple racks, which, for cooling purposes, are placed in an area of a few to a few tens of square meters [15]. Under these circumstances, it is extremely difficult to apply synchronization and close coordination among the system's parts as required in unbuffered crossbars.

1.3. Variable-Packet-Size Buffered Crossbars

A related, significant advantage of buffered crossbars is their capacity to directly switch variable-size packets [7] [17]. Since the 2·N schedulers in a buffered crossbar can operate asynchronously, there is no global "time-frame", that constrains the system to transmit packets in fixed-size units (segments or cells) –fig. 2. This does not hold for the unbuffered architecture, where the centralized scheduling has only be shown to operate efficiently when it manipulates fixed-size cells –fig. 1.

In turn, directly switching variable-size packets eliminates the other reason for internal speed-up, i.e., to compensate for the segmentation overhead when the packet size is not an integer multiple of the segment size. It also removes the cost of large memories and reassembly mechanisms in the egress line-cards. Speed-up and buffering are major contributors to cost, hence variable-packet-size buffered crossbars have the potential of significantly lowering the cost of packet switches and routers. Concerning power, variable size packets will generally reduce average consumption, since operations like scheduling and forwarding configuration will be made on larger units (packets) and hence less frequently.

The challenge is that if each crosspoint buffer/queue has the capacity of one maximum-packet-size (or MxPS in short), output buffers can be eliminated, thus reducing the overall cost of the switch even more. In [16], we demonstrate that a 32×32 buffered crossbar supporting all network packets of size up to 1500 bytes is typically feasible with current ASIC technology, even within a single chip (with 2 KBytes buffer space per crosspoint).

1.4. Multiple Priority Levels

Multiple priorities are desirable to provide means for service differentiation; e.g. IEEE 802.ID/Q defines eight classes of service by means of priorities. Proper priority mapping, on the application or the packet level, can provide efficient utilization of network resources and increase the "user-perceived" utility [18] [19]. For instance, the HIGHEST priority level can be used for network-control packets, other, relatively high intermediate priority levels can be used for highly-interactive low-rate applications like TELNET, and lower intermediate levels for policed/bounded rate sources, like streaming/multimedia applications; delay insensitive applications, like FTP or mail, can use the LOWEST priority level.

1) Adaptive Priority Mapping: An even more sophisticated approach is to dynamically map "critical" packets, like TCP connection establishment or acknowledgment packets, to the highest priority levels, and to use the remaining levels for less "critical" packets [18]. An interesting approach is also presented in [19]; this study shows that if the nodes of the network are priority-aware, dynamic/adaptive priority mapping performed on the packet level can be used to sustain the rate of a flow above a minimum, without requiring explicit admission control.

The models in [18] and [19] assume that the network's nodes take into account the priority of the packets' only regarding queuing, i.e., RED dropping policies. It becomes evident though, that if the schedulers at the nodes of the network are also priority-aware, similar or even better results will be obtained. Much of the work in this report was inspired by these papers.

2) Multiple Priorities in CICQ Switches: A CICQ switch supporting multiple priority levels, normally requires separate crosspoint buffers/queues (or lanes) per priority to prevent HOL blocking and buffer hogging; these effects can make a high-priority packet receive low-priority service (see section 4.2). However, each additional such queue incurs a significant cost, hence we want to economize on their number.

With 32 Mbits of on-chip memory, we can construct in a single chip, a variable-packet-size buffered crossbar with no output buffers in the egress line-cards, that supports effectively two priority levels by employing a separate crosspoint lane per priority –assuming packets of size up to 1500 bytes and 2 KByte buffer space per crosspoint queue. By increasing the number of priority levels, L, that we want to support beyond two, the memory required inside the crossbar chip easily becomes unfordable; e.g. 64 Mbit for $L = 4$ and N = 32.

1.5. Contents

This report studies a buffered crossbar that directly switches variable-size packets and supports multiple priorities while economizing on the number of lanes per crosspoint. Section 2 pictures previous work on buffered crossbars and multiple priorities. Section 3 presents our baseline architecture, regarding flow-control, scheduling, queuing organization and operations in the ingress line-cards, credit semantics, crosspoint buffers cut-through and dimensioning. It also presents some alternative schemes for handling credits and a novel alternative positioning of the input schedulers. Section 4 presents possible queuing organization at the crosspoint and the related HOL and buffer hogging effects.

In section 5.1, we propose a method, SQP, that resolves, in the short term, HOL blocking and buffer hogging which occur when many different priorities are multiplexed in a shared lane; even with one queue per crosspoint, SQP prevents unjustified starvation of high priority flows. Section 5.2 proposes an enhanced version of SQP, method 2B-ADAPT, that effectively supports multiple priorities with two distinct lanes per crosspoint. Section 6 discusses the incremental hardware cost of our methods and is shows that the only complexity introduced is located in the ingress line-cards. In section 7, we show through simulation that when eight priorities are supported, even under highly irregular traffic, 2B-ADAPT will not increase the average delay of the HIGHEST priority level by more than **75** percent, when compared to a system with a distinct lane per priority, i.e., four (**4**) times more buffering; intermediate priorities experience much smaller discrepancies. Under smoother patterns, like typical network traffic or Poisson arrivals, we find that the performance of the two systems is almost identical. We also compare the variable-packet-size buffered crossbar to a fixed-size-cell one and to pure output queuing.

In many cases, while we define our architecture, we present plots from simulations to consolidate our positions, or to demonstrate the emerging trade-offs. These simulations assume a 32×32 variable-packet-size CICQ switch with port speed 10 Gb/s employing no internal speed-up and schedulers that implement the Start-Time Fair Queuing [20] variant of FQ [2]. For information regarding the simulation models and the simulation environment please refer to section 7 and to appendix *Simulator Architecture*.

2 . Previous Work & Contribution

2.1. Buffered Crossbars

Buffered crossbar proposals date at least as far back as 1987: Nojima e.a. [21] described a "bus matrix" switch with buffers only at the crosspoints (no input buffers), operating on variable-size packets; Katevenis [22] proposed a switch with small crosspoint buffers, large input buffers, and backpressure between them (figures 10-13). Recently, with the availability of technology for single-chip buffered crossbars, a number of groups studied *fixed-size-cell* buffered crossbars –see for instance [8] [9] and our previous work [10] [11]; from industry, a representative example is [15].

2.2. Variable-Packet-Size Buffered Crossbars

Variable-packet-size crossbars have been proposed also in [7] by Stephens and Hang and in [17] by Chrinstense et. al. Our present work differs from these studies in that we consider several design issues in variable-packet-size buffered crossbars: Firstly, we demonstrate the pros and the cons of placing the input lines' schedulers within the crossbar chip and we show that cut-through can be implemented at the crosspoint buffers; we discuss for a reasonable buffer space per crosspoint queue under cut-through and under store&forward. We analyze various schemes for handling credits, with different storage strategies within the crossbar chip and different backpressure communication overheads; we present an "acknowledgment"

[2]These results assume a single priority level; multiple-priorities simulations are presented in section 7 .

semantics for the credits, which minimizes the backpressure bandwidth required and also reduces the amount of credit storage that must be deployed within the crossbar chip. We also propose an algorithm to program scheduling operations in the variable-packet-size context. These issues have not been examined in previous works, or have been implicitly neglected. Secondly, in a related work, [16], among other things, we analyze the cost of an actual hardware implementation of a variable-packet-size crossbar; in section 3 of this report we give several cost estimates regarding gates, area and power. The only other hardware study, [23], is for an FPGA implementation.

Stephens and Zhang [7] consider variable-size *internal* packets in their simulations, but limit their length up to twice the minimum packet size, i.e., up to 80 bytes; larger external packets are still segmented. Their focus is on providing rate and delay guarantees to individual flows, and for this they present simulations with overloaded output ports. Our simulations explicitly model variable-size packets and study several arrival patterns. Yoshigoe and Christensen [17] evaluate the performance of the buffered crossbar only for crosspoint buffer size of 1500 bytes (i.e., one MxPS), without specifying the backpressure RTT and the bandwidth dedicated to credits, while, we explicitly study the dependence of performance on these parameters. In addition, our simulations implement cut-through at the crosspoints, while theirs implement store and forward.

2.3. Multiple Priorities

Concerning the multiple priorities, the only relevant studies on buffered crossbar have been in [17] and in [15]. In [17] a single experiment is conducted for only two priority levels and Poisson arrivals. This paper does not show how to handle multiple priority levels into a reduced number of crosspoint buffers; we consider exactly this aspect, and our work applies on both variable-packet-size and cell systems.

In [15] eight priorities are considered in a fixed-size cell buffered crossbar, but the methods they employ to handle multiple priorities are in our view costly and inefficient: by implementing multiple "logical" queues in a shared crosspoint buffer space, one for each priority level, (1) the complexity of the crossbar chip is considerably increased; (2) the scheme as presented in [15] can not resolve buffer hogging –this effect is equivalently harmful with the HOL blocking, which is prevented by employing per priority, logical, crosspoint queues –, and (3) can not be extended straightforwardly in variable-packet-size switches, since up to now, no study has shown how to implement multiple variable-packet-size queues in a shared buffer space without considerable fragmentation overhead.

In contrast, our methods (a) do not increase the complexity of the crossbar-chip since we employ only crosspoint queues implemented in private buffer space; (b) as we demonstrate in this report, perform considerably better than a shared crosspoint-memory system with multiple logical queues at each crosspoint, and (c) are applicable in both variable-packet-size and cell buffered crossbars.

To the best of our knowledge, this is the first published study of how to effectively map multiple priority levels onto a reduced number of queues in a buffered crossbar –and perhaps in any kind of switch in general. The importance of the report stems from this novelty and from the importance of buffered crossbars –especially variable-packet-size ones– as the probable emerging architecture of choice for future commercial crossbar products.

The most relevant methods with our adaptive schemes, SQP, SQP-opt and 2B-ADAPT, can probably be found in the telecommunication/networking literature, for instance in TCP flow-control. While we are only a little familiar with the respective literature, we regard that our methods are novel in this framework as well, since we take the impact of priority scheduling under consideration in determining and avoiding congestion.

3 . Basic Architecture

The system that we examine in this report is a buffered crossbar with advanced input queuing (i.e., with virtual output queues –in short, VOQs), that directly switches variable-size packets; separate VOQs are maintained at each input for each priority level (see fig. 2). We consider that traffic consists of flows identified by distinct input/output/priority-level triples, i.e., $N^2 \cdot L$ flows, where L equals to the number of priority levels that are supported. We refer to the VOQ that corresponds to flow f as VOQ[f].

3.1. Flow-Control

1) Credit-based Backpressure: Credit-based flow-control is employed in order to provide lossless transmission at the link level between the input line-cards and the buffers within the crossbar. When a packet is selected for service at the output, a credit informing about the packet departure is sent to the respective input. To simplify the design, we consider that the credits are transmitted from a different interface than (user) packets are.

2) Backpressure Bandwidth Overhead: The credit rate per input port is set equal to one credit per minimum-packet-size (or MnPS in short) time on the packet lines. This is the minimum backpressure communication overhead that guarantees stable operation for the switch: the average rate of credit arrivals at an ingress line-card, i, cannot exceed the peak rate of packet departures from ingress i (assuming unicast traffic). In the worst-case, up to N credits with destination a single input line-card can be generated in a MnPS time, if at some point, all output schedulers serve crosspoint queues from the same row of the crossbar and the HOL packets in these queues have size equal to MnPS; such credit bursts must be stored within the crossbar. In this report, we consider that the credits for input i are stored in a FIFO queue, separate from all other inputs. More sophisticated queuing/scheduling disciplines than simple FIFO could be employed, but as demonstrated in [13], the gain would not be very significant; still, this issue deserves further investigation.

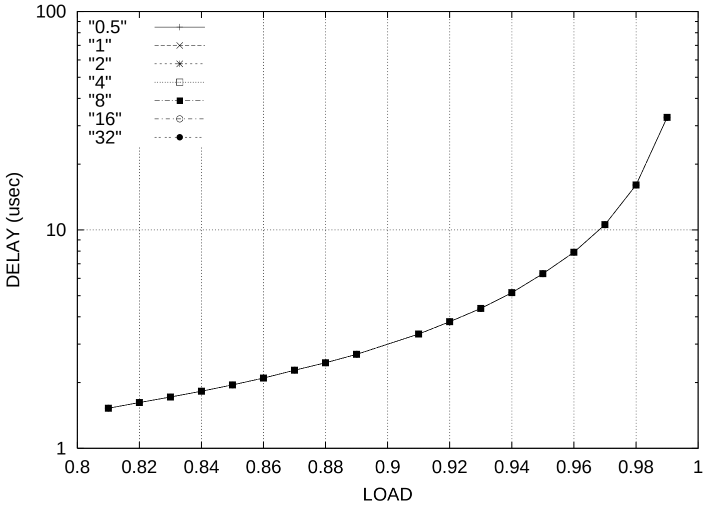

Fig. 3. 32×32 switch, 10 Gb/s port speed, 2 KByte buffer space per crosspoint; payload consists of packets with size that follows the Pareto distribution (average-packet-size 400 bytes, MnPS 40 bytes, MxPS 1500 bytes)

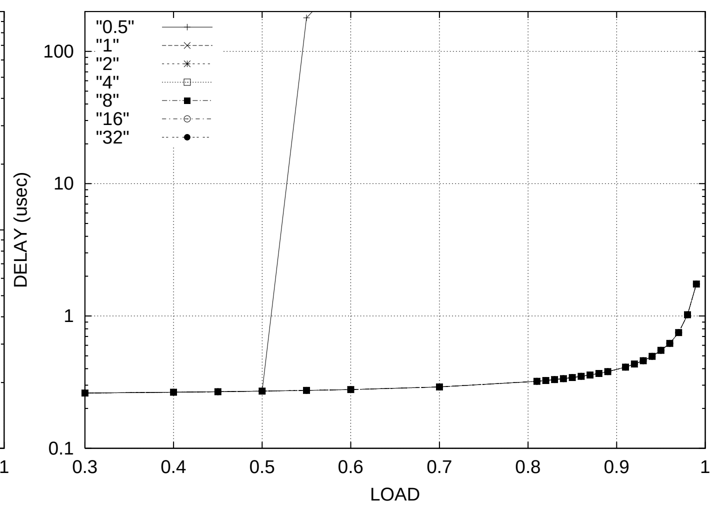

Fig. 4. 32×32 switch, 10 Gb/s port speed, 2 KByte buffer space per crosspoint; payload consists of MnPS packets only

3) Flow-control & Eligible Flows: Throughout this report we consider that each flow has a dedicated VOQ; so we consider that scheduling at the inputs is performed among flows, which is equivalent to scheduling among VOQs. A flow, f, of priority l, which arrives from input port i and goes to output port j, is eligible for service at the contention point located at ingress line-card i, only if: (1) VOQ $[f]$ contains a packet p at its HOL and (2) the flow-control ascertains that sending p toward the crosspoint queue, Q, assigned to p, will not cause Q to overflow. All the packets of f use crosspoint queues located at the crosspoint $\mathrm{x}_{i\to j}$. Note, that in some methods that we propose, a policy can mark a flow as ineligible for scheduling at the input, even when conditions (1) and (2) are true.

If a flow can use multiple different queues at the same crosspoint, a routing decision is involved when checking the eligibility of f, in order to determine the queue at crosspoint $\mathrm{x}_{i\to j}$, that will be used by packet p. In all our schemes, except for 2B-ADAPT, all the packets of a flow, f, have a predetermined crosspoint queue that they use, which is dedicated to packets of f, or is being shared by packets of multiple flows.

Only two of the scheme that we consider in this report maintain a dedicated/private crosspoint queue for each flow, i.e., MQDB and MQSB; most of our schemes multiplex multiple flows, of different priority level, in the crosspoint queues. So it is more simple to define output scheduling among crosspoint queues. A crosspoint queue, Q, is eligible for service at the output line of the crossbar, when a packet, p, is stored in Q. We assume cut-through operation at the VOQs and the crosspoints queues, so by the time a packet starts being enqueued in an empty such queue, Q, Q becomes eligible for service.

4) Credit-Rate Experiments: Diagram 3 demonstrates the relaxed dependency of backpressure bandwidth to the performance of the switch. In this experiment we use uniform Poisson arrivals and we increase the backpressure bandwidth per input port, starting from one credit every two MnPS times (0.5) up to 32 credits every MnPS time, i.e., when the system accommodates the peak rate that credits are generated. We see that no observable improvement is accomplished by increasing the backpressure communication overhead.

It is interesting that the 0.5 plot matches the 1 plot: since, in this experiment, the average packet size is 400 bytes (packet sizes follow the Pareto distribution), the worst case arrival pattern, that we used to claim that the credit bandwidth should be at least one credit every MnPS time, does not dominate. This does not mean, by any way, that 0.5 is a good design choice; it implicitly means though, that with variable-size packet CICQ switches, the average power consumption on the backpressure signals will be lower compared to fixed-size cell CICQ switches and that the variable-packet-size switch, is more stable under temporal malfunctions of the backpressure protocol.

In diagr. 4 we perform the same experiment like in fig. 3, but we use only MnPS packets; as we can see in the figure, all delays have been reduced by a factor of ten. This is due to packets having on average ten times smaller size. Again, increasing the credit bandwidth above 1 credit every MnPS time does not improves performance. But here, the 0.5 configuration performs very poorly for input load higher than 0.5, since when the incoming packet rate is greater than credit rate, the generated credits built-up in the credit queues within the crossbar chip, and the outgoing packet rate is limited by the credit rate.

3.2. Positioning of Input Schedulers

An alternative architecture that we examined during the early stages of this work, was to implement the input lines'

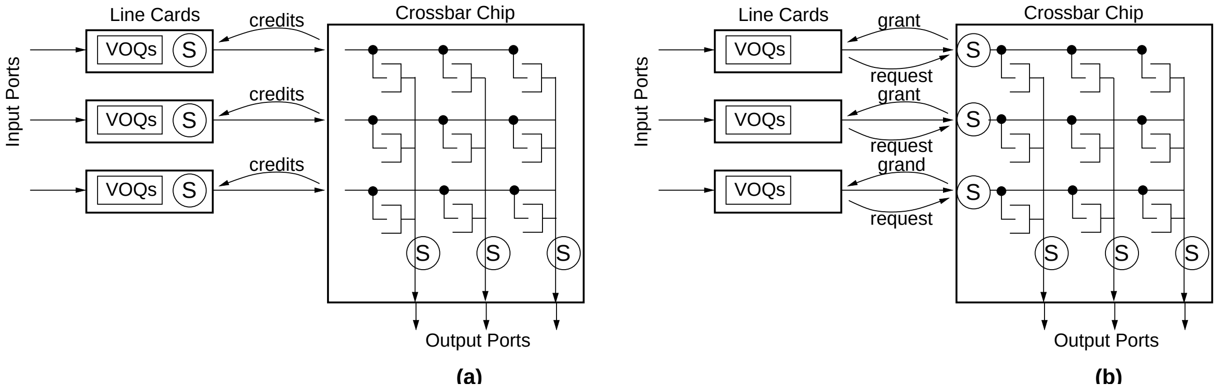

Fig. 5. A 3×3 CICQ switch, (a) with the input schedulers at line-cards and (b) with input schedulers inside the crossbar chip

schedulers within the crossbar chip –see fig. 5. If these schedulers are placed in the crossbar chip, they will have fast and full access to the crosspoint buffer occupancy information, and thus, the need for credit communication and credit storage will be eliminated. But (i) these schedulers will add to the cost of the crossbar chip; (ii) the ingress line-cards will need to communicate to the crossbar the size of the head packet of each VOQ [3]; (iii) the scheduler's decision would need to travel to the line-card before the next packet can depart from the line-card to the crossbar, effectively increasing the scheduler's latency and (iv) the schedulers would have delayed information regarding the state of the VOQs.

For these reasons we abandon this alternative, although it has the potential to increase the performance of the system: assuming that the N schedulers corresponding to the N input lines are maintained inside the crossbar chip, then these (a) will have absolute knowledge of the current state at the crosspoint buffers; this contrasts with the delayed information, which they otherwise extrapolate from credits that are subject to queuing and transmission delays; (b) each input scheduler will have access to the state of *all* crosspoint buffers and not only of the crosspoints at the corresponding row of the crossbar; such information is useful in implementing scheduling disciplines at the inputs that take into account the congestion at the outputs lines, since the latter can be estimated from the occupancy of the respective crosspoint buffers [24]; finally, (c) this architecture enables a closer form of coordination among the input schedulers, which otherwise, when they are working isolated in ingress line-cards distant from each other, they can take independent decisions, that when combined, reduce the short term utilization of the output lines.

For instance, when all input schedulers select packets that go to the same output o, short term underutilization of the crossbar can appear, if no packet is eligible at any other output. This behavior can temporarily reduce the aggregate utilization of the output lines by N times, since if the input schedulers had selected packets destined to different outputs, all the outputs would have been busy. Fortunately, in time proportional to a crosspoint buffer's size, the flow-control will impose on some inputs to stop sending toward output o, and thus, this behavior can not continue in the medium or in the long term.

[3]One message every time a VOQ is served, plus one message every time a packet arrives into an empty VOQ.

3.3. Pending Packets & Pending Credits

A terminology that we extensively use in this report, is that of *pending* packets and of *pending* credits. We name as pending each packet p, that has been sent toward crosspoint queue Q from the ingress, but its corresponding credit c has not yet been received at the ingress. This means either (1) that the packet has not reached the crosspoint queue, i.e, it is being propagated; or (2) that the output scheduler has not yet started serving p, or (3) that it has, but c is currently stored within the crossbar or is traveling toward the ingress.

Pending credits are these credits that have been generated by the transmission of a packet, but are currently stored inside the crossbar chip, waiting for the their turn to start traveling toward the ingress. So pending credits presume an associated pending packet at the ingress, but the inverse relationship does not hold in general: the p packet may be pending at the input context, with p traveling toward the crossbar or with p waiting for service at the crosspoint queue, that is with no associated pending credit.

3.4. Credits Semantics: Acknowledgments & PSS FIFOs

In order to reduce the backpressure overhead in the variable-packet-size switch that we consider, we employ the following method. Each ingress line-card maintains a small FIFO for each crosspoint queue, Q, that it feeds; each such FIFO stores the size of each packet that is pending in the respective crosspoint queue. We name these FIFOs, packets-sent-size FIFOs, or PSS in short.

When the ingress line-card starts transmitting a packet toward a crosspoint queue, Q, it also enqueues the size of p in PSS [Q]. With this data structure, a credit needs only to convey an id that discriminates Q between all other crosspoint queues that are fed by the same input. Given this queue id, the ingress line-card can retrieve the amount of the buffer space that is deallocated in Q, by performing a dequeue operation in PSS [Q]. This method (a) saves us log (MxPS) bits per MnPS time, bandwidth that otherwise must be accommodated at the

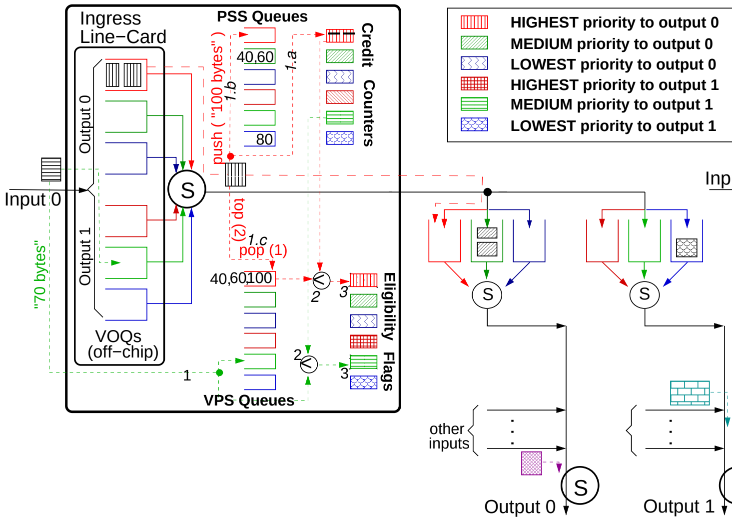

Fig. 6. A N×2 buffered crossbar, supporting three priority levels with a separate crosspoint queue for each priority.

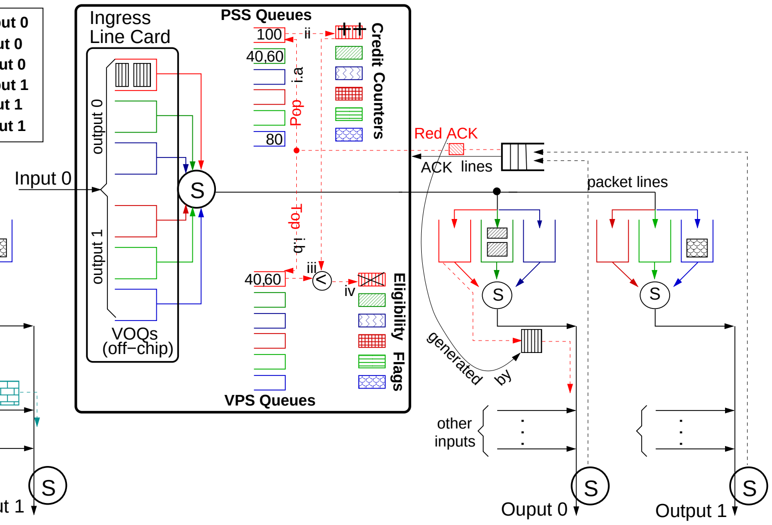

Fig. 7. A N×2 buffered crossbar, supporting three priority levels with a separate crosspoint queue for each priority.

interface from the crossbar chip toward each ingress line-card; and also (b) simplifies credit handling within the crossbar chip (see section 3.6). Note that in the worst-case, a PSS FIFO may need to store B/MnPS packets' sizes, where B is the size of the respective crosspoint queue. Under this semantics, we name credits as acknowledgments (in short, ACKs).

3.5. Computation of VOQs Eligibility

One important issue in a variable-packet-size switch with input buffering is the way that the ingress line-cards handle variable-size packets, in terms of queuing and flow-control. Regarding the credit flow-control that we employ, each input line-card, i, maintains a credit counter (in short, CC) for each crosspoint queue Q that it feeds; CC [Q] accounts for the free space in Q that has been acknowledged to the input.

In order for a controller located at an ingress line-card, to decide if the transmission of a packet, p, toward one such queue, Q, will cause buffer overflow, the size of p must be also available. This intricacy which is present only in variable-size-packet switches can impose a prohibitive number of off-chip memory accesses in the ingress line-cards, if packets' sizes are stored only in the VOQs, e.g. inside the header of the respective packet.

1) VPS FIFOs: So we employ on-chip queues at the ingress line-cards, one for each VOQ that the line-card maintains, where we store the size of the packets at the respective VOQ. With this data structure, that we name virtual-output-queue-packets-size (in short, VPS), the controller at the input can easily compute the eligibility flags of the competing flows without additional off-chip memory accesses.

2) Rate of VOQs Eligibility Checks: As figures 6 and 7 demonstrate, this controller needs to compute the eligibility flag of a flow f, only at times when packet of f, p arrives or is served at the input, or at times when a credit concerning f arrives. After computing this flag, the controller stores it into the eligibility registers which are used by the scheduler when a new scheduling decision is required. The situation complicates if multiple flows, G, share a crosspoint buffer or queue, in which case, a credit from this queue can change the eligibility flags of all flows in G. Assume for the moment that each flow f has a dedicated crosspoint buffer/queue, Q, so that no other flow can use Q; also assume that the the input knows a priori which crosspoint queue to use for f's packets.

3) Packet Events: When a packet p of size s, belonging to flow f, arrives to the input, (1) s is enqueued in VPS [f], (2) the eligibility flag of f is computed by comparing s with CC [Q] (i.e., f is eligible iff $s \leq$ CC [Q]) and (3) it is stored in the respective eligibility register (fig. 7). When p departs from the ingress, (1.a) CC [Q] is decremented by s and (1.b) s is enqueued in PSS [Q]; in parallel, (1.c) first a dequeue (pop) operation and afterward a return-head (top) operation are performed in VPS [f], in order to find the size of the new HOL packet in VOQ [f]; finally (2) the new eligibility flag of f is computed and (3) it is stored in the eligibility registers –see fig. 7.

4) Credit Events: Figure 7 illustrates the operations that take place when a credit (ACK) from crosspoint queue, Q, arrives at the input: (i.a) a dequeue (pop) operation is performed in PSS [Q] and (i.b) a return-head (top) in VPS [f]; next, (ii) the CC [f] register is incremented by the value dequeued from PSS [Q] in (i.a) and (iii) the eligibility state of f is computed, by comparing CC [f] with the value returned from the top operation in the VPS [f] in (i.b); (iv) this flag is then stored in the eligibility register of f.

Special methods can be used to eliminate the need of this on-chip FIFO memory, but are more specific to the particular implementation of the VOQs' controller. For instance, each read access/fetch to a packet in the VOQs can be followed by an access to the header of the next packet in the same queue that dispatches the size field. If these packets are maintained in neighbor locations, then the two accesses can be bundled together with negligible overhead. We have only recently started to investigate schemes for logical queues that accommodate variable-size packets; in this report, we do pertain how the variable-size packet are maintained at the VOQs, hence we do not consider such implementation specific methods.

3.6. Credits Handling within Crossbar Chip

In this section, we investigate maintenance schemes for the credits that have not yet been transmitted toward their corresponding ingress line-card, and thus remain pending inside the crossbar chip. Such methods are required when the communication bandwidth that is dedicated to flow-control between the crossbar chip and each input line-card, i, in separate, cannot accommodate the peak rate that credits are generated for input i. Consider for the moment input i only, and thus, only the crosspoint queues that are fed by this input and only the credits generated in these queues.

1) Credits Conveying Size: First suppose that the input does not maintain the PSS FIFOs. In this case, the crossbar-chip must explicitly inform the input about the space that is deallocated from packets' transmission. Two strategies can be used to store the credits that contain this information and which are pending within the crossbar.

(a) Each crosspoint queue maintains in a separate *counter* the amount of the deallocated buffer space in the respective queue, that the input has not yet been notified about. Thus C$\cdot$ N^2 such counters must be employed within the crossbar, where C is equal to the number of queues per crosspoint. Each of these counters must be $\log B$ bits wide, where B is the size of each crosspoint queue in words [4](total, additional memory on chip C$\cdot$ N$^2\cdot$ $\log B$ bits). No more bits are required since the input is not allowed to have more than B words pending in a crosspoint queue [5]. Under this organization, if we assume a *non-uniform* credit-format, the communication bandwidth between the crossbar chip and the ingress line-card must be at least $\log Y$ bits per Y-words time on the crossbar lines', plus $\log$ (C$\cdot$ N) bits per MnPS time: the former term is required to transfer the buffer space that is being deallocated on average at the corresponding crosspoints (i.e., equal to the rate that words from input i enter the crossbar) and the latter term is required to transfer the crosspoint queue ids, when back-to-back MnPS packets leave the crossbar.

If a *uniform* credit format is used instead, then, we can either permit bundling of credits generated by the transmission of multiple packets from a single crosspoint queue, or we can dedicate each credit-message to the transmission of a single packet. In the latter case, the bandwidth communication bandwidth required is $\log$ MxPS + $\log$ (C$\cdot$N) bits per MnPS time. This is not efficient though, since it will cause underutilization of the credit lines, when multiple credits are generated in consecutive MnPS intervals, by the transmission of multiple MnPS packets from a single crosspoint queue: each such credit will be transmitted in a $\log$ MxPS + $\log$ (C$\cdot$N) bits credit-message, where only $\log$ MnPS + $\log$ (C$\cdot$N) bits will be used, and thus, a part of the credit lines' throughput will be wasted.

If credits generated by multiple packets from the same crosspoint queue can be bundled together in a single credit-message, then the respective communication bandwidth is $\log G$ + $\log$ (C$\cdot$N) bits per MnPS time, where G is the maximum buffer space that can be acknowledged within a single credit-message. G is certainly smaller than B and must be greater than MnPS.

In any case, under the uniform credit format we can set the size of the credit-messages to a value smaller than it is imposed by the MxPS packets (e.g., G lower than MxPS), if we are allowed to send the credit information of larger packets in two or more credit-messages. This may reduce the backpressure bandwidth that is wasted when credits are generated by the transmission of small packets.

(b) Each credit for input i is maintained as an individual credit-message inside a buffer (e.g., a FIFO) dedicated to the credits for input i; N such buffers are required inside the crossbar chip, one for each input. Each credit-message is generated by the transmission of a packet from a crosspoint queue on the output lines, and contains the size of the packet and the id of queue. Thus each entry in that buffer has width $\log$ (C$\cdot$ N) bits for the queue id plus $\log$ MxPS for the deallocated buffer-space in that queue; assuming that one credit is transmitted toward each input every MnPS time, this buffer must have approximately C$\cdot$N$\cdot\frac{B}{MnPS}$ entries.

This buffer space compensates for the case, when all outputs continuously service MnPS packets from crosspoint queues corresponding to input i, thus generating N credits for this input in every MnPS time while that backpressure bandwidth can accommodate only one; obviously, since the respective crosspoints cannot hold more than C$\cdot$N$\cdot\frac{B}{MnPS}$ pending packets, this will not continue for very long. Hence, the total buffer space inside the crossbar chip for credit-messages will be approximately C$\cdot$N$^2\cdot\frac{B}{MnPS}\cdot$[$\log$ (C$\cdot$ N) + $\log$ MxPS] bits. The read rate in a buffer corresponding to a single input must be $\log$ (C$\cdot$ N) + $\log$ MxPS bits per MnPS time and the write rate N$\cdot$ [log (C$\cdot$ N) + $\log$ MxPS] per MnPS time.

2) Acknowledgments: Now consider that the input maintains the PSS queues, hence, each credit needs only to carry a crosspoint queue id (ACK). Again, as with the credits-conveying-size semantics, two strategies are possible:

(i) Each crosspoint queue maintains a counter for its ACKs that are pending inside the crossbar chip; thus we need C$\cdot$ N^2 such counters in total. The width of each counter must be $\log\frac{B}{MnPS}$ bits. No more bits are needed since the input is not allowed to have more than $\frac{B}{MnPS}$ pending/unacknowledged packets into a single crosspoint queue. Similarly to (a), the backpressure bandwidth required for each input port is $\log Y$ per Y-MnPS time plus $\log$ (C$\cdot$ N) bits per MnPS time under a *non-uniform* credit format, and $\log W$ + $\log$ (C$\cdot$ N) per MnPS

[4] Meaning the width of the datapath inside the crossbar chip

[5] We assume that each crosspoint queue has a private buffer, of size B words, and thus, the buffer space at each crosspoint is C$\cdot$ B words; if the C queues in a crosspoint share the B words buffer space, then the C term is eliminated.

	EXPLICIT-CREDITS			Vs ACK-CREDITS		
STORAGE TYPE:	COUNTER		BUFFER	COUNTER		BUFFER
CREDIT-FORMAT:	uniform	non-uniform	uniform	uniform	non-uniform	uniform
backpres. B/W (per input port) bits per	$\log G+$ $\log (C \cdot N)$ MnPS time	$\log Y$ 1 word time plus $\log (C \cdot N)$ MnPS time	$\log (C \cdot N)+$ $\log (MxPS)$ MnPS time	$\log W+$ $\log (C \cdot N)$ MnPS time	$\log Y$ Y×MnPS time plus $\log (C \cdot N)$ MnPS time	$\log (C \cdot N)$ MnPS time
storage size in crossbar-chip (in bits)	$C \cdot N^2 \cdot \log B$		$C \cdot N^2 \cdot \frac{B}{MnPS} \cdot [\ \log (C \cdot N) + \log MxPS\ ]$	$C \cdot N^2 \cdot \log (\frac{B}{MnPS})$		$C \cdot N^2 \cdot \frac{B}{MnPS} \cdot \log (C \cdot N)$
R&W B/W of each cred. buffer (1 per input-port) in bits per	–		$(N+1) \cdot [\ \log (C \cdot N) + \log MxPS\ ]$ MnPS time	–		$(N+1) \cdot \log (C \cdot N)$ MnPS time

TABLE I

COST ESTIMATES FOR DIFFERENT CREDIT HANDLING SCHEMES

time under a *uniform* one, where W is the number ACKs from a single crosspoint queue, that can be bundled together in single credit-message. As in (a), W must be greater than 1 and is lower than $\frac{B}{MnPS}$; intermediate values are also possible.

(ii) A buffer (e.g. FIFO) stores all the ACKs that are pending inside the crossbar and are destined to the same input; again we need N such buffers in the crossbar, one for each input. If one ACK is being transmitted toward each one of the inputs every MnPS time, then the depth of each buffer must be approximately $C \cdot N \cdot \frac{B}{MnPS}$ entries –see (a); each entry must be $\log$ (C· N) bits wide, and thus $C \cdot N^2 \cdot \log (C \cdot N) \cdot \frac{B}{MnPS}$ bits are required within the chip . The access rate (read and write) to each buffer must be (N + 1)· $\log$ (C· N) bits per MnPS time.

Table I summarizes our points regarding the credit handling schemes that we consider in this section. This table shows that the acknowledgment semantics for credits achieves significant savings in communication and storage cost. Regarding the counters or buffers dilemma, one point is that the former need the counter logic in addition to counters registers (memory), whereas, buffer storage can be implemented as a simple shift register or a register file; SRAMs can be used if extensive storage for pending credits is required. Finally the non-uniform credit format reduces backpressure communication overhead but complicates the design.

Under strategy (ii) we can make the following optimization: (iii) Instead of employing a separate buffer entry for each distinct ACK generated from some crosspoint queue at a single row of the crossbar we can use a bitmap that can contain up to C· N ACKs, generated by distinct such queues. Each crosspoint queue in a row of the crossbar will map into a single position in the respective bitmap, thus the size of the bitmap will must be C· N bits; the value "one" for some bit means that the respective queue has generated one ACK which is pending inside the crossbar. With this optimization, we eliminated the need to store the queue ids associated with the acknowledgments, and thus the read/write access rate to each credit-buffer is reduced, i.e., two bitmaps (i.e., 2· C· N bits) per MnPS time per input port. Using the same reasoning as in (a), it turns that the size of a credit buffer corresponding to some input port must be approximately $\frac{B}{MnPS}$ bitmaps i.e., $C \cdot N \cdot \frac{B}{MnPS}$ bits. Note that instead of a bitmap, we can employ an array containing ACK-counters, M bits wide each, to create a mix of strategies (i) & (iii).

3) Reducing Credit Storage: Under many of the aforementioned schemes, we can use another level of flow-control to limit the buffer/counter size required to store pending ACKs within the crossbar chip. With backpressure on the generated credits, a crosspoint queue will be eligible for service at the output of the crossbar only if it can "store" the credit/ack that will be generated. This backpressure can be made selective (i.e., in row of the crossbar only the crosspoint queues that have send "many" credits are blocked), in which case it will better serve overall scheduling, since no unjustified blocking will occur. Note that this flow-control on credits engages entities which are located in the same chip, and thus, it can be made fairly simple and effective.

Under (ii) or (iii) we can eliminate credits storage within the crossbar chip, if the backpressure communication bandwidth of the crossbar-chip with each ingress line-card can accommodate the transmission of one bitmap (C· N bits) every MnPS time. In this case, it is simpler to transmit directly the ACK bitmap to the input; this saves us the need for complex credit handling methods within the crossbar chip and also enhances performance, since the credits will be transmitted with no queuing delays. But this choice is acceptable only for low port counts or low datapath rates. We used this technique to build a 4×4 buffered crossbar, switching variable size packets in an FPGA.

4) Implementation Considerations: Considering an actual implementation of a medium scale switch, one would normally dedicate one twisted pair for the backpressure communication of the crossbar-chip with each input line-card in separate. Assuming that eight twisted pairs are dedicated in the reverse direction for the packets payload, in a MnPS time we can transfer approximately up to five bytes of credit information; this backpressure bandwidth translates into 5-6 ACKs under schemes (i) or (ii), when N equals 32 and C is equal to 2.

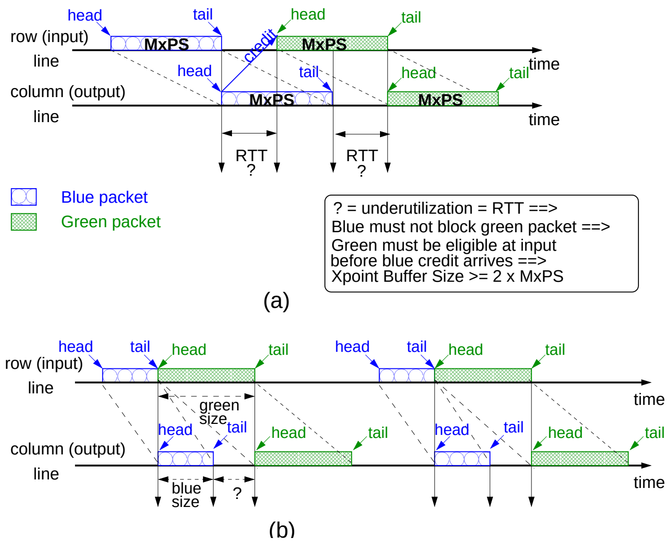

Fig. 8. Store & Forward

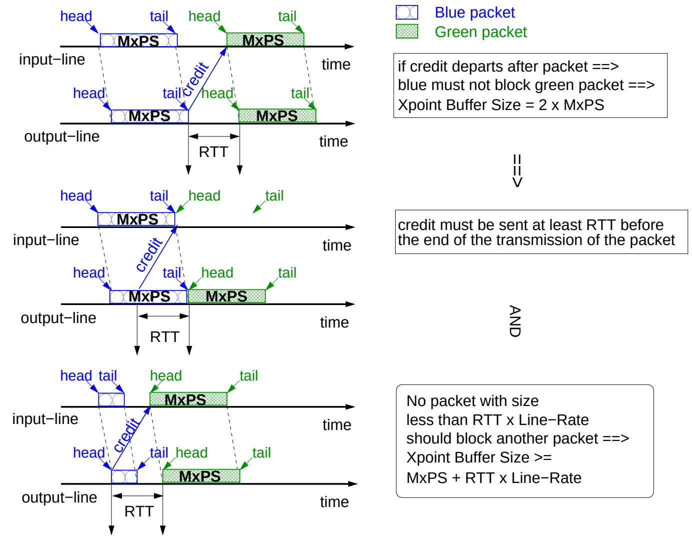

Fig. 9. Credit generation time

Thus, the details of the implementation will normally impose the most appropriate scheme for handling the credits.

An other implementation issue related to credits handling, is how to recover when a credit or a packet is lost due to line failure. We consider such issues in [16], where we design in hardware a single-chip, 32×32, variable-packet-size buffered crossbar.

3.7. Crosspoint Functions

We consider that each crosspoint queue, Q, resides in a private buffer space of a dual-ported SRAM [16] and that it is implemented using a *head* and a *tail* pointer that move circularly inside the buffer space dedicated to Q. Circular queues have the advantage that they can typically accommodate variable size packets. As we have mentioned, a packet switches clock domain inside the crosspoint queue; the read (head) pointer is synchronous with the output link's clock, whereas the write (tail) pointer is synchronous with the input link's clock.

The output schedulers within the crossbar chip perform cut-through, i.e., a packet becomes eligible for scheduling at the output side of the crossbar, at the time it's header reaches the crosspoint logic, that is even before it's entire body has been stored in the crosspoint buffer. An efficient implementation for this organization at the crosspoints is demonstrated in [16].

1) Store & Forward: If we implement store and forward at the crosspoints, then the buffer space per crosspoint queue required to sustain full output-link utilization even when a single flow, f, is active, is:

$$\mathrm{B}^{STORE} = 2 \times \mathrm{MxPS}$$

Assume that LR stands for the line rate and RTT stands for the delay from the generation of a credit till the first word of a packet, that utilized this credit at the input, starts being transmitted on the output lines of the crossbar assuming that the crosspoints perform cut-through. So, in RTT we include the propagation delay of a packet's, p, header, from the ingress line-card toward the crosspoint, but not the associated transfer delay of packet's p body; this latter delay is necessary, before the output considers p as eligible for service, when the crosspoints implement store and forward.

B^{STORE} buffer space at each crosspoint queue is necessary when f sends continuously pairs of MxPS packets, $p1$ and $p2$, at full line rate. If this buffer space is not available to f, then every $p2$ packet will be blocked by the corresponding $p1$ at the input; the credit for $p1$ will be generated when $p1$ is selected for service, and thus when $p1$ will be written in the crosspoint queue e.g. at time $t1$. The second packet $p2$, will be eligible at the output, at time $t1$ + RTT + $\frac{MxPS}{LR}$, i.e., when $p2$ will also be written in the crosspoint queue after receiving $p1$'s credit. Thus the output line will remain idle in the interval that starts from $t1$ + $\frac{MxPS}{LR}$ when the transmission of $p1$ ends, till $t1$ + $\frac{MxPS}{LR}$ + RTT, when $p2$ will start being transmitted on the output lines. –see fig. 8(a).

Additionally, store and forward has the drawback that it can increase considerably the delay of individual packets. Suppose, for instance, that two back-to-back packets of different size, arrive at a crosspoint queue –see fig. 8(b). Compared to cut-through, store and forward can increase the delay of the second packet, by time proportional to its size, while the output line remains idle –see fig. 8(b).

2) Cut-Through: Under cut-through, the minimum buffer space at each crosspoint queue, that is required to sustain full output-link utilization even when a single flow is active, is:

$$\mathrm{B}^{CUT} = \mathrm{RTT} \times \mathrm{LR} + \mathrm{MxPS} \quad .$$

To understand why, consider that a flow, f, sends pairs of back-to-back packets at full line rate; each first packet, $p1$, has size $s1$ equal to MxPS, and every second packet, $p2$, has size $s2$ equal to max [B - MxPS, MnPS], where B is the size of the crosspoint queue that is used by f; $p1$ and $p2$ have been selected so that (1) $s2$ is as small as possible, while (2) $p2$ is able to block $p1$ at the input. Condition (1) creates the necessary condition for underutilization, and condition

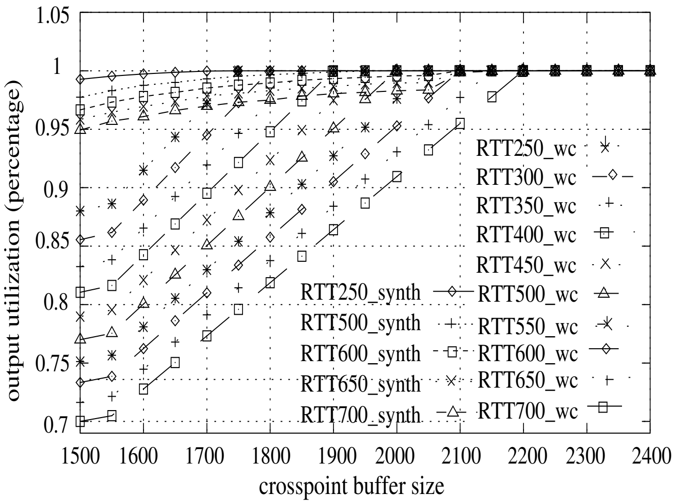

Fig. 10. RTT Experiment. For buffer size less than 1500 bytes, all measurements for output utilization are zero. Upper curves stand for SynthBackb traffic, lower curves for worst case.

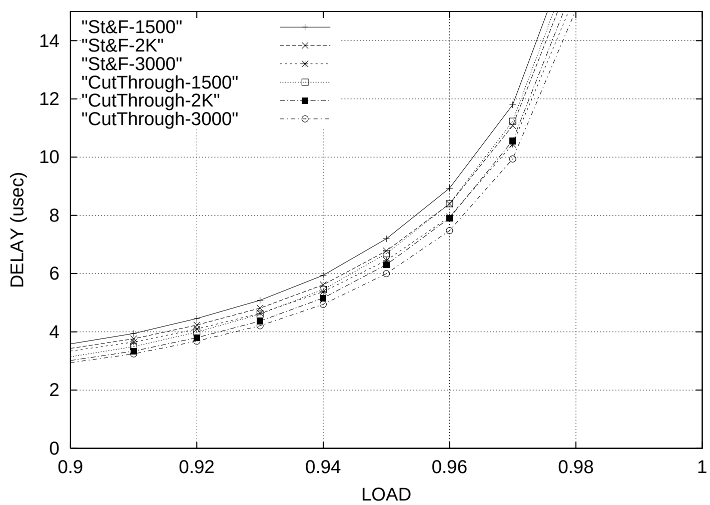

Fig. 11. Store&Forward Vs Cut-Through; 32×32, 10 Gb/s per port

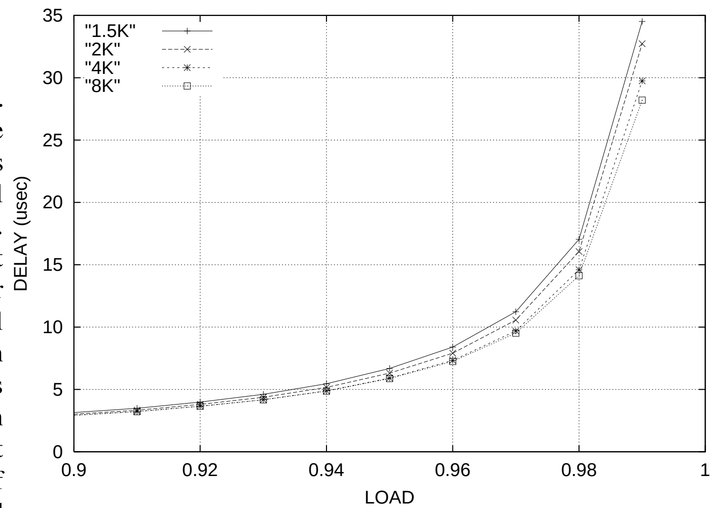

Fig. 12. 32×32, 10 Gb/s per port; uniform Poisson arrivals

(2) maximizes the duration of this possible underutilization. We claim that with buffer space at the crosspoint queue equal to B^{CUT}, no underutilization can occur. This happens because with B^{CUT} buffer space, we impose that $p1$ will be blocked at the input only if $s2$ is greater than RTT×LR. But in this case, when $p1$ will be ready for transmission at the output after receiving $p2$'s credit (i.e., RTT times after starting transmitting $p2$ on the output lines), the output will still be busy transmitting $p2$, because its size is greater than RTT×LR. So, full output utilization is guaranteed. In this example, we assumed that if the size of $p1$ is greater than RTT×LR, then the credit corresponding to $p1$ is generated at most RTT before the transmission of $p1$ ends; if the size of $p1$ is smaller than RTT×IR, then the credit must be generated immediately at the start of $p1$'s transmission; if we delay the generation of the credits, then the respective buffer space requirement will be 2 · MxPS –see fig. 9. In the system we simulate in sec. 7, cut-through is implemented at the crosspoints and credits are generated immediately when the transmission of packets start.

Figure 10, demonstrates the behavior of the switch regarding the relationship of the RTT with the buffer space required at each separate crosspoint queues. In this experiment we use a single, persistent flow, and MxPS and MnPS are set equal to 1500 bytes and 40 bytes respectively. For buffer size B, at the crosspoint queues, varying from 1400 to 2400 bytes, we measure the output utilization as a fraction of the line rate; we repeat the experiment for RTT values varying from 250 to 700 byte times. First, we let the packets of the flow being generated by a realistic traffic pattern, **SynthBackb**, which is explained briefly in sec. 7.3.2, and then, by the aforementioned, worst-case pattern. As the figure shows, output underutilization occurs for every B less than MxPS + RTT×LR. This knee in the utilization plots at crosspoint buffer size equal to MxPS + RTT×LR is more evident under the worst-case scenario, but appears also and under the **SynthBackb** arrivals.

In diagram 11, we measure the average packet delay of store and forward and of cut-through, with varying buffer space at the crosspoint under uniform Poisson packet arrivals. The different plots correspond to configurations with 1500 bytes (i.e. equal to MxPS), 2 KByte (i.e. equal to B^{CUT}) and 3000 bytes (i.e. equal to B^{STORE}) buffer space at each crosspoint. We can see that cut-through performs better than store and forward, under all buffers' size.

In fig. 12 we use cut-through and we demonstrate that under round-robin scheduling and uniform Poisson traffic, increasing the buffer space per crosspoint queue beyond B^{CUT} will not boost the performance considerably; this is not true when weighted round-robin scheduling is employed at the contention points [10], or when the packet arrivals are not uniformly distributed to outputs.

In diagr. 13, we employ an imbalanced (i.e. non-uniform) traffic scenario that we borrow from [4]; each input sends with

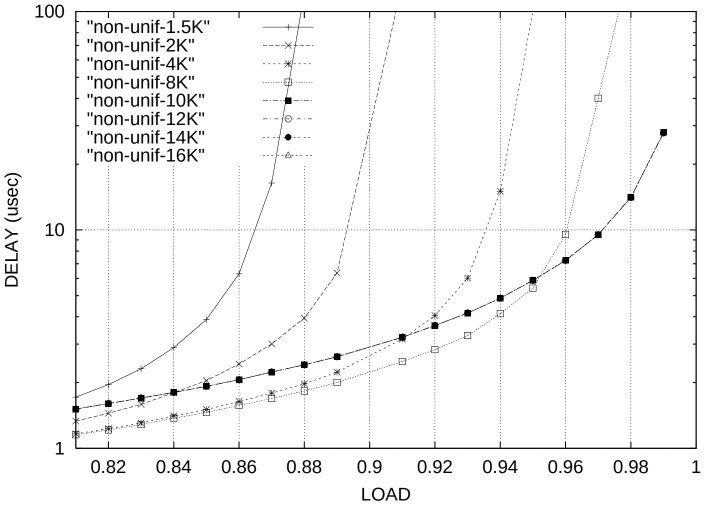

Fig. 13. 32×32, 10 Gb/s per port; non-uniform Poisson arrivals

probability 0.5 to a distinct "favored" output and distributes the remaining incoming traffic to the remaining outputs, uniformly; as in [4], this pattern is feasible. In fig. 13 we see that the 2K system saturates near input load 0.9; the throughput increases with increasing buffer space and becomes close to 1.0 when 10 KByte or more of buffer space are employed at each crosspoint. This traffic pattern is very interesting, since many of the switch architectures that are currently considered for high-speed switches, buffered or unbuffered, cannot sustain it well. One interesting approach to eliminate this effect in CICQ switches, has been presented in [24]. We have also been searching for similar scheduling methods, that take into account the occupancy of the crosspoint buffers –see method adaptive-WRR in [25], appendix **D**, page 145.

3.8. Scheduling Discipline at Contention Points

Currently we use symmetric scheduling at the input and output contention points. Remember that for a flow, f, to be eligible at the input, two conditions must hold: (a) a packet, p, must be present in VOQ $[f]$ and (b) the buffer space available to the crosspoint queue Q, where p will be sent, must be greater or equal to the size of p, i.e., in credit-based flow-control terms: CC [Q]$\geq$ sizeof (p); a crosspoint queue, Q, is eligible at the output, if it contains at least one packet –actually it suffices that the header of the packet has reached Q, since we perform cut-through at the crosspoints.

The scheduling discipline assumed at the input and the output links is a combination of non-preemptive priority scheduling and Fair-Queuing (FQ) implemented by the SFQ variant, with all flows having equal weight: when queues of different priorities are eligible for service, the highest eligible priority, l, is selected; if multiple queues with priority l are eligible, we use SFQ to select one of them. We use SFQ and not simple pointer-based RR, since the latter is oblivious of packets' size, and by so can produce unfairness, for instance, when it serves a MxPS packet from a flow, f, and a MnPS packet from a flow, g, in a round-robin fashion.

3.9. Scheduling the Operations at the Contention Points

Scheduling variable-size packets introduces some intricacies, since the time that it takes to transmit a packet on the crossbar lines varies due to the varying size of packets. One constrain is that the scheduling delay (in short, SD) must be less than the MnPS time, so that back-to-back MnPS packets can be forwarded at full line rate.

Name as t_{s_start} the time when a scheduling operation starts and as t_{s_end} the time when scheduling finishes, i.e., $t_{s\_end} = t_{s\_start}$ + SD. Let ST (t) denote the state of the variables affecting this scheduling decision at time t; ST includes flows/queues eligibility, credits state etc; obviously this state can change at any time. We assume that the scheduler picks and uses a snapsot of this state at time t_{s_start}, after all events that affect ST (t) at time t_{s_start} have been considered; any events that occurs in the interval $(t_{s\_start}, t_{s\_end}]$, cannot affect the scheduling outcome. We use this hypothesis, since we believe it is generic enough to accommodate any kind of scheduler.

1) Scheduling Timing: After the scheduling decision is taken at time t_{s_end}, the operations for the transmission of the selected packet $p1$, start. We regard that a new packet, $p2$, can start transmission at time t_{s_end} + $\frac{s}{LR}$, where s is the size of $p1$. If $p1$ and $p2$ are to be sent back-to-back, the latest moment that the next scheduling operation should start is t_{s_end} + $\frac{s}{LR}$ - SD, so that the next scheduling phase overlaps with the transmission of the last words of $p1$. We do not want to start scheduling earlier than this time, so that the scheduling-space (the number of eligible queues) is as "wide" as possible and scheduling sophistication can take place; we do not want to start scheduling later than t_{s_end} + $\frac{s}{LR}$ - SD, so that no bandwidth is lost, if at that time one or more eligible flow are waiting for service.

Before the start of a scheduling phase, a controller first checks to see if there exists at least one eligible queue –we assume that this operation, a huge OR gate on the queues eligibility flags, has essentially zero delay when compared to the delay of sophisticated scheduling algorithms. If no queue is found eligible at this time, the controller does not start a scheduling operation/phase, but deters scheduling for the first moment when some queue will become eligible. In this way, some bandwidth is saved in case that a flow, f, becomes eligible during a "null-result" scheduling phase, i.e., a scheduling operation, with duration SD, that finds no flow as eligible. If we allowed such "null-result" scheduling operations to take place, then f would have to wait for the end of the ongoing "null-result" phase, before the scheduler could select it at the next scheduling phase, and thus, f's delay would be increased unnecessarily.

2) Scheduler/Link Controller: Figure 14, shows graphically the finite-state machine of the controller that schedules the operations at an input or an output contention point (i.e. line) of the crossbar: IDLE-SCHEDULING means that scheduling operation is taking place while no packet is being transmitted on the corresponding link; BUSY-SCHEDULING

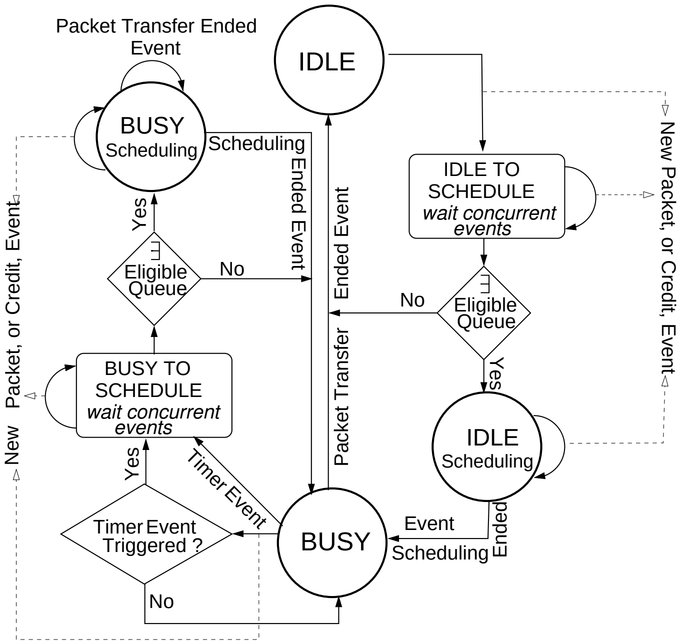

Fig. 14. Link/Scheduler Controller for Variable-Size-Packets

means that scheduling operation is concurrent to the transmission of a packet. When the controller enters the BUSY state, signaling the start of the transmission of a new packet, $p1$, at some time t, it also programs a Timer-Event (e.g., as generated by a counter) to occur at time t + $\frac{s}{LR}$ - SD, where s is the size of $p1$. As we stated above, changes in the eligibility flags of the queues before this Timer-Event triggers are neglected by the controller.

When the Timer-Event triggers, the controller checks the eligibility of the queues, and if it finds at least one eligible, (a) it enters into the BUSY-SCHEDULING state to select a new packet, $p2$, that will be transmitted at time t + $\frac{s}{LR}$; at this time the controller enters again into the BUSY phase, but now for the transmission of the new packet $p2$ [6].

If no eligible packet is found when the Timer-Event triggers, (b) the controller returns with zero delay into the BUSY state. But now, if a queue becomes eligible at some time t^* after t +$\frac{s}{LR}$ - SD while the controller is still BUSY transmitting $p1$, the controller will immediately enter the BUSY-SCHEDULING state, so that the new eligible packet can be served at at time t^* + SD, when the transmission of $p1$ will have certainly ended.

Other optimizations are also possible. For instance we can preempt the current scheduling phase, when a high priority queue becomes eligible while the scheduler is considering only flows with lower priority. Certainly the details of these optimizations depend strongly on the architecture and the actual algorithms used to implement the scheduler in hardware.

[6]Note that the Packet-Transfer-Ended event associated with the transmission of packet $p1$, which is also programmed to occur at t + $\frac{s}{LR}$, is masked-out in this case since, otherwise, it could switch the controller's state from BUSY to IDLE while the controller is BUSY transmitting $p2$ and not $p1$.

3.10. An Example of the Switch Operation: RTT

In this section, we present an operation scenario of the simulator that we created for the variable-packet-size buffer crossbar switching –see appendix *Simulator Architecture*. The scenario and the simulator, emulates the order of the basic operations in an actual switch. Suppose that a new maximum-size packet, p, arrives at the input line-card i at time t, and that there are adequate credits for this packet to travel toward it's corresponding crosspoint queue, i.e., no other packet is present in the system.

The input scheduler will start scheduling at time t and will finish at time t + SD; at this time, a memory access will be performed, to fetch the packet from the VOQs. So after memory access delay (in short, MemDel), that includes memory/bank activation, row and column accesses, plus the delay at the chips interfaces, the first word of the packet will be inside the ingress line-card chip, ready for transmission towards the crossbar chip. After a propagation delay (in short, PD), that includes the time-of-flight on the interconnect between the crossbar-chip and the ingress line-card, plus the delays at the chips interfaces and the delays that occur in extensively pipelined circuits, the packet header will reach the crosspoint buffer [7], and the packet will be eligible for scheduling at the output.

The output-scheduler will start scheduling at time t + SD + MemDel + PD, and will finish at time t + 2· SD + MemDel + PD; at this time an ACK will be generated, that must be sent to the ingress line-card i. For now we assume zero credit-scheduling delay (e.g. on-chip FIFO queuing), so this credit can start being propagated immediately, since we assumed that no other packet or credit is present in the system. After a propagation delay, and a transfer delay for the credit (in short, CD) equal to $\frac{credit\_message\_size}{credit\_line\_rate}$, the ACK will reach the ingress line-card i at time:

$$\text{RTT} = t + 2\cdot \text{SD} + \text{MemDel} + 2\cdot \text{PD} + \text{CD},$$

and the input scheduler will be able to make a new scheduling decision, that takes this credit under consideration.

So the Round-Trip Time, RTT, includes two scheduling and two propagation delays, one "first-word" delay in off-chip memory, and one credit transfer time (CD) on the backpressure interface that connects the crossbar-chip with an ingress line-card. As we have mentioned above, we want the scheduling delay and the credit transfer delay on the credit-lines to be slightly less than a MnPS time (i.e., $\frac{MnPS}{LR}$). Using these requirements we have that :

$$\text{RTT} = 2\cdot \text{PD} + \text{MemDel} + 3\cdot \frac{MnPS}{LR}.$$

The dominant factors on the RTT are the propagation delay and the MnPS time. Note though, that current trends show the MnSP time to decrease as line speed increases, and the propagation delay to increase, as switches are now oriented over multiple racks, distant from each other by tens of meters, basically for cooling purposes [15]. Also observe that when internal speed-up is employed, the LR in the RTT equation

[7]We currently consider that the internal header of the packets' internal header have zero size.

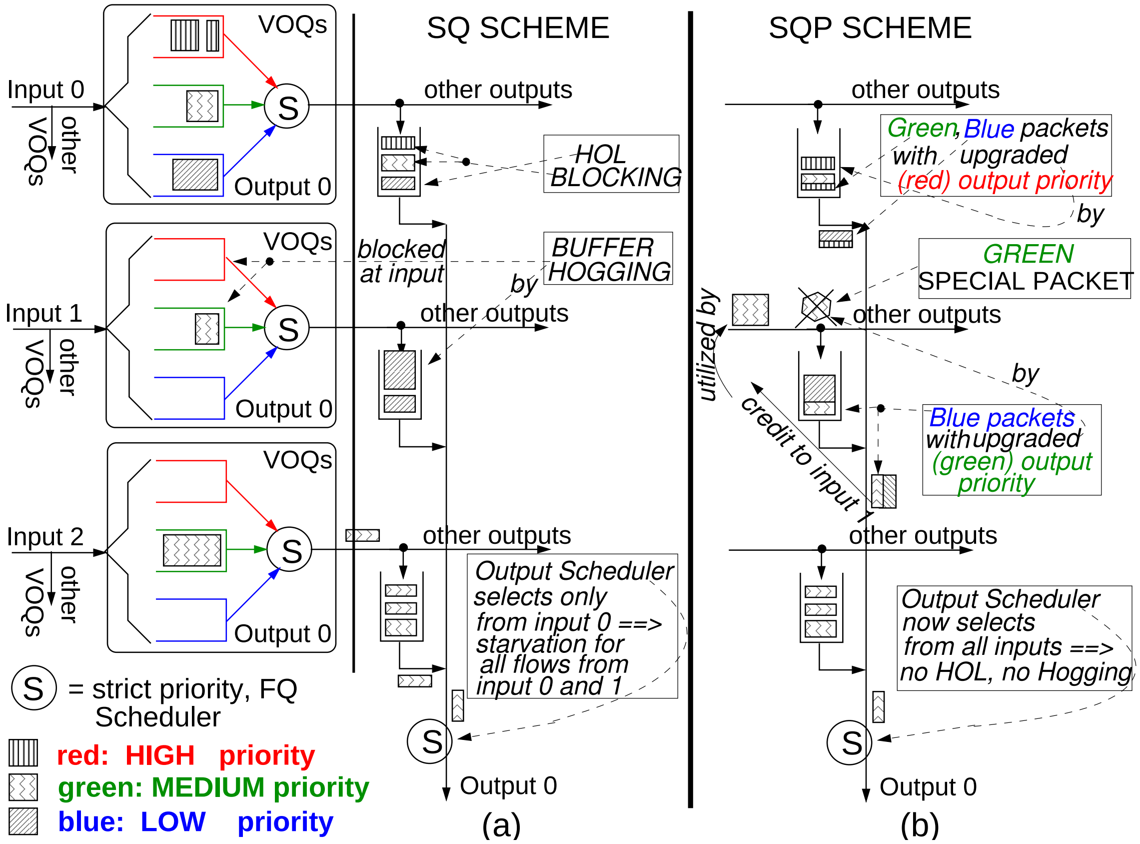

Fig. 15. **(a)** HOL blocking & Buffer Hogging in a SQ switch **(b)** SQP, see section 5.1.1

is multiplied by the speed-up factor, s, and thus, the RTT decreases; nevertheless the minimum required buffer space at each crosspoint increases by a term equal to $s\times$ [2· PD + MemDel] –see section 3.7.2 .

4 . Queuing Organization at the Crosspoints

Separate lanes per priority at each crosspoint are desired in order to provide flow isolation and protection, thus for reasons similar to why a buffered crossbar provides better performance compared to a shared-memory architecture. The incremental cost of each additional crosspoint lane is counted in buffer space that must typically be implemented within the crossbar chip: $N^2\cdot$ RTT·LR for a CICQ system switching cells, and $N^2\cdot$(RTT·LR + MxPS) for a CICQ switch directly transmitting variable-size packets.

4.1. Partitioned Crosspoint Buffer

In a CICQ system that deploys a separate crosspoint buffer/queue (or lane) per priority level, each flow is statically mapped to a separate crosspoint queue. Thus each ingress line-card maintains N·L credit counters, L for each crosspoint that it feeds; and a credit, c, that arrives at the input from a crosspoint queue, affects the eligibility status of one flow only. This simplifies the controller that is required at each ingress line-card, to determine which flows are eligible to be transmitted toward the crossbar (see section 3.5).

1) L Crosspoint Lanes: MQDB - The Ideal: We name the CICQ switch that employs a separate crosspoint lane for each priority level as multiple-queue-distinct-buffer (in short, MQDB). The drawback of the MQDB architecture is that it requires buffer memory in the crossbar chip that increases linearly with the number of the priority levels, i.e., O ($L\cdot N^2$). An instance of a MQDB switch supporting two priorities is presented in fig. 2.

4.2. Shared Crosspoint Buffer

A CICQ system that deploys less than L distinct buffers at each crosspoint, has to multiplex multiple priorities in a shared buffer. Two organization are possible; the first implements a single crosspoint queue, whereas, the second implements multiple logical queues in the shared crosspoint buffer space.

1) One Crosspoint Lane: SQ - HOL blocking: If a single crosspoint queue, Q, is shared among multiple priorities and low priority packets have been enqueued in Q first, a higher priority packet that is inserted after them can experience unacceptable long delay until it becomes the head of Q; this phenomenon is known as Head Of Line (HOL) blocking –fig. 15(a). We name the respective CICQ architecture as single-queue (in short, SQ). In SQ the number of priority levels supported does not affect the memory required within the crossbar chip –O(N^2)– and one credit counter at each input for each output suffices.

2) L Logical Crosspoint Lanes: MQSB - Buffer Hogging: Even if multiple queues, one for each priority, are maintained at each crosspoint but these queues share a common buffer space (i.e., logical queues) similar effects to HOL blocking can occur if low priority packets have exhausted the shared buffer space: higher priority packets, that are ineligible for service at the input due to flow-control, will have to wait until the low priority queues are served, so these can experience a delay comparable to that of low priority traffic; this effect is known as buffer hogging.

We name a system with multiple logical queues at each crosspoint, one for each priority level, as multiple-queue-shared-buffer (in short, MQSB), when all the queues at a crosspoint share a common buffer space. MQSB requires the same amount of buffer memory inside the crossbar chip, and the same structures for flow-control operation in the ingress line-cards, with SQ.

Note that buffer hogging can also appear in the SQ scheme, while MQSB by definition prohibits HOL blocking. Both HOL blocking and buffer hogging can have destructive effects on the higher priorities. Even a flow in the HIGHEST priority level, that normally receives preferential service, can experience starvation, which is indirectly induced by congestion of flows with intermediate priority –fig. 15(a)

A disadvantage of MQSB compared to SQ, is the control circuit, that is required at each crosspoint in order to maintain the logical queues. Besides the fragmentation that can occur if variable size packets are to be stored inside the queues, this scheme significantly increases the complexity of the already heavily loaded crossbar chip. Instead, circular queues, implemented in private buffer space, require much simpler control (e.g. a *head* and a *tail* counter) and do not incur fragmentation. Additionally, the implementation of a queue through a linked list, requires an *nxt* pointer for each memory block compromising the linked list, thus, consumes additional on-chip memory. A MQSB system, switching cells, is examined in [15].

3) RS optimization - Reducing Input Complexity: Under the SQ or the MQSB architecture, upon the arrival of a credit for a shared crosspoint queue, the controller associated with an ingress line-card may need to compute the eligibility of the L flows that use this queue i.e., flows that go through the same crosspoint and have different priorities (see section 3.5).

We can remove this intricacy if we employ the following discipline at the inputs: Denote by g the flow of the highest priority among all flows, G, with non-empty VOQs that go through the same crosspoint: if g is ineligible for service at the input, obviously due to missing credits for the shared crosspoint buffer space, then all flows in G will also be considered as ineligible. Under this discipline, a credit arrival event will induce the computation of the eligibility status of g only. We name the respective SQ and MQSB systems, as SQ-reserve-space and MQSB-RS-reserve-space (in short, SQ-RS and MQSB-RS), respectively.

We named this discipline as reserve-space, because, in addition to reducing input complexity, the RS tends to reserve the shared buffer at each crosspoint to the flow of the highest priority among all corresponding active flow. Thus, we expect that the HOL blocking and buffer hogging, that are present in SQ and MQSB, are normally reduced with the RS optimization: when variable-size packets are transmitted from the ingress in a SQ or a MQSB switch, then a low priority flow, f, that sends small packets, may win scheduling rounds at the input by taking advantage of that the higher priority levels going to the same output are currently ineligible, because they have large packets at the HOL of their VOQs. Consequently, f consumes additional credits for the buffer at the corresponding crosspoint, and thus, the same situation is renewed, or even worsens by reforming into buffer hogging. But even when a higher priority finally finds the credits necessary to travel to the crosspoint, it has to face the potential of HOL blocking caused by the packets of f. The RS optimization is expected to reduce this lingering of high priority packets behind "lucky", low priority packets. We confirm this behavior by simulations in section 7.3.1 .

5 . Methods

The methods that we propose use circular queues at the crosspoints. We do not consider schemes with multiple logical crosspoint queues, since we believe that this approach significantly increases complexity, especially in variable-packet-size CICQ switches. During the early stages of this work we tried to implement two circular queues storying variable-size-packets, inside a shared buffer and we verified how complex this is [26].

So, now, in order to reduce the memory required within the crossbar, we have to multiplex multiple priorities into a reduced number of lanes (queues/buffers). We first consider a scheme with only one such lane per crosspoint, and afterward we consider a more sophisticated method, which employs two such lanes at each crosspoint.

5.1. One Crosspoint Lane with Push-Forward: SQP

In the first scheme that we explore, like in SQ, all flows from the same input that go to the same output share a single crosspoint queue. So the system, as presented so far, is vulnerable to HOL blocking and buffer hogging.

1) SQP: In order to resolve HOL blocking we employ the following discipline at the crosspoints: the "effective priority" of a crosspoint queue Q, for output scheduling purposes, is the highest of the priorities of the packets currently enqueued in it; in this way, Q drains with the priority of the most "urgent" packet currently inside it. By doing that, low priority packets at the HOL can no any longer cause starvation to high priority packets behind them. We name this scheme single-queue-push-forward (in short, SQP) –see fig. 16(b).

To resolve buffer hogging, the input in SQP sends a special packet to signal this priority upgrade, in lieu of the actual packet which cannot be sent due to flow-control constraints. Special packets are not stored in crosspoint queues, neither

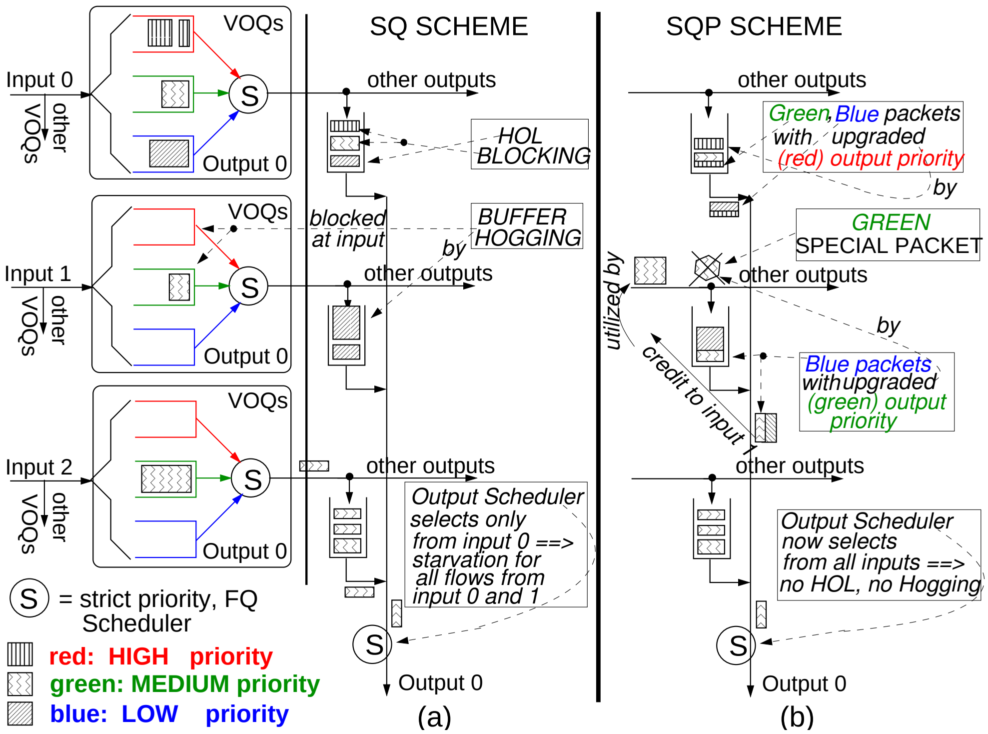

Fig. 16. **(a)** HOL blocking & Buffer Hogging in a SQ switch **(b)** SQP, see section 5.1.1

they are forwarded to outputs, but they are subject to scheduling at the input[8] and are transmitted like normal packets, through the same interface. Their size has been set equal to a MnPS, so that the scheduling rate is not affected.

2) SQP optimized: Consider now the case, where an input, i, has not yet received credit from a crosspoint queue, Q, for a packet of priority l^*, higher than the highest eligible priority, l, in the VOQs that correspond to Q.

This fact indicates probable heavy traffic at level l^* or higher in the crossbar. Under such conditions, if a priority l packet is enqueued in Q, this will need to wait until all packets in front of it are served and probably even more if output congestion actually occurs; this increases the probability that buffer hogging or HOL blocking will appear if a packet with priority higher than l arrives later at input i for the same output. So we performed the following optimization to SQP: when packets of priority l^* or higher are still pending (i.e., not yet acknowledged to the input by means of credits) at a crosspoint queue, Q, the input is not allowed to send a packet with priority lower than l^* toward Q.

We name this system as SQP-optimized (in short, SQP-opt). Although SQP-opt is not work-conserving, we show through simulations, that compared to SQP, it significantly reduces the delay of all high priorities, without considerably affecting the crossbar utilization. Another advantage of SQP-opt is that it simplifies the priority upgrade mechanism that must be implemented at the crosspoints.

[8]They inherit the priority of the packet blocked at the input

3) SQP-RS: The RS optimization has no meaning under the SQP-opt system: in SQP-opt, if a packet of priority l is not eligible at the input due to lack of credits and (1) the effective priority of the corresponding crosspoint queue, Q, is lower than l, then the respective flow, f, will be eligible to sent a special packet of priority l, and thus, the priority scheduler at the input can not select a lower priority packet. When (2) the effective priority of Q is equal or higher than l, then f will not be eligible at the input, but the same will also hold for all flows of lower priority flows than l (SQP-opt discipline). The latter statement does not hold under SQP, so we can define the respective SQP-RS system, similarly to SQ-RS.

Although with the aforementioned methods persistent buffer hogging and HOL blocking that may appear under worst-case scenarios are resolved, this comes at the cost of occasionally transmitting packets with higher priority than their specification, which increases the load at higher priorities, hence, potentially increases the average delay of high priorities. Considering that the crosspoint queues have relatively small capacity, we expect that this counter effect will not play a dramatic role.

5.2. Two Crosspoint Lanes with Adaptive Mapping: 2B-ADAPT

Our second method employs two separate lanes at each crosspoint. This is feasible with todays ASICs, at least for a 32×32 switch, with 10 Gb/s port rate. Nevertheless the complexity and the cost of the design increases, given that the area of the crossbar chip in a buffered crossbar is dominated by SRAMs modules –see [16].

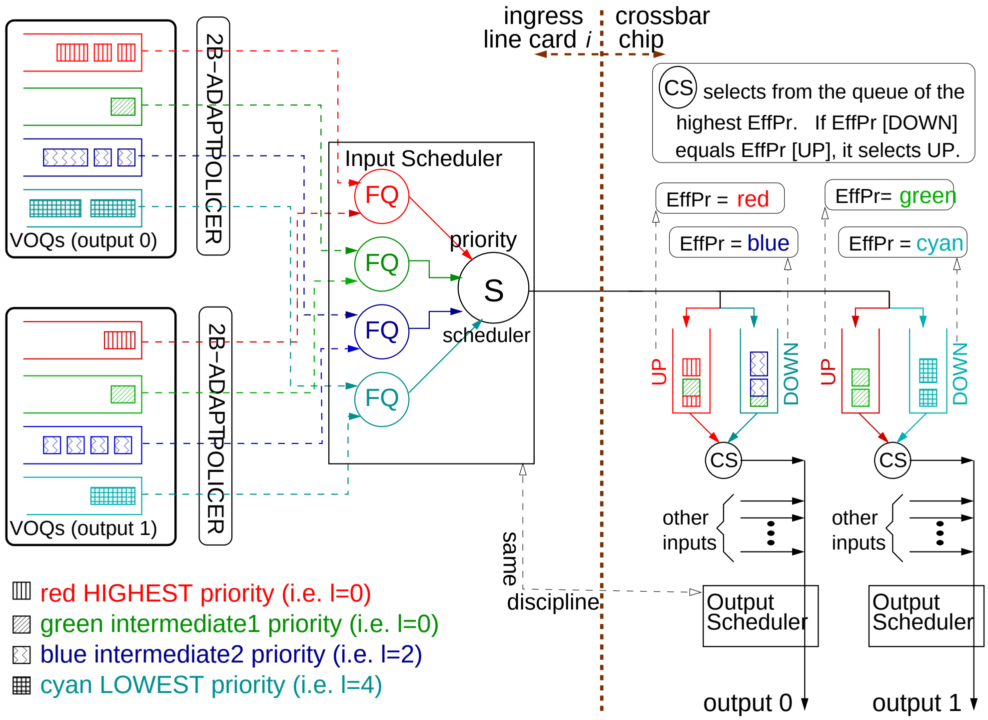

Fig. 17. Queue/scheduling organization under 2B-ADAPT.

Since there are two lanes that a packet may use to reach the scheduler at the output, a routing problem arises, i.e., to decide which such lane to use for a flow's packet. We favored for a dynamical routing performed on the packet level, and not for statical one, e.g., in the flow level; the entity that we employ to performs the routing is also a policer, since it can block a packet, in order to reserve the lane for higher priorities.

Our method, 2-buffers-with-adaptive-mapping (2B-ADAPT in short), uses SQP-opt ideas in a more sophisticated way. We assume two separate buffer/queues (lanes) per crosspoint; UP stands for one of these lanes and DOWN stands for the other –fig. 17.

1) Input Policy: An algorithm, Alg-2B-ADAPT, running independently at each input line-card, decides through which lane a packet can make it to the output. Alg-2B-ADAPT should not be confused with the input scheduler; essentially, Alg-2B-ADAPT monitors whether a flow f is eligible for service: it returns UP or DOWN when it finds f eligible –the packet at the HOL of f's VOQ can then use the respective lane– and NONE otherwise. The input scheduler, as described in sec 2.1, selects one of the eligible flows. Alg-2B-ADAPT is adaptive, since it processes packets based on the run-time state of the two lanes at the corresponding crosspoint.

2) Output Scheduler: The output scheduler in 2B-ADAPT considers all the non-empty UP and DOWN queues in a column of the crossbar and serves the one with the highest effective priority (l); when both the UP and DOWN queues in a crosspoint have the effective priority l, the scheduler selects UP. We can imagine that a crosspoint scheduler/arbiter selects UP or DOWN, according to the above discipline, and propagates it's decision to the corresponding output scheduler.

3) 2B-ADAPT Goal: The main idea in 2B-ADAPT is the following. Consider all flows from a given input to a given output. We try to use only the DOWN lane, but when multiple priorities are active, we allocate the UP lane to the highest priority among them. The packets of the other active flows, either use DOWN, or are kept at the VOQs. For all flows of priority l, lower than the HIGHEST supported, the algorithm is conservative when it uses UP: we want to keep the UP queue usually empty, so that a packet of priority higher than l, that arrives later at the input, is able to go through UP without experiencing the delay of traffic with priority l. Whenever HOL blocking appears in any of the two queues, or buffer hogging appears at UP, the mechanisms of SQP apply. Thus, Alg-2B-ADAPT can also return SpUP, meaning that the flow is eligible but only to send a special packet UP.

4) 2B-ADAPT Algorithm: We assume the following data structures at each input, for each output; the number of flows corresponding to this pair is L, the number of priority levels. Two bits for each flow f, FQ $[f]$, indicating whether the most recently sent and still pending packet of this flow was sent UP or DOWN, or NONE (if f has no pending packet). Similarly, PackUP $[f]$ counts the number of f's pending UP packets; and TimeUP $[f]$ keeps track of the time that has elapsed since the input started transmitting the oldest pending UP packet. Finally, two registers, EffPr [UP] and EffPr [DOWN], estimate the "effective" output-priority of UP and DOWN respectively, by keeping track of the highest pending priority in the corresponding queue; when no packet is pending UP or

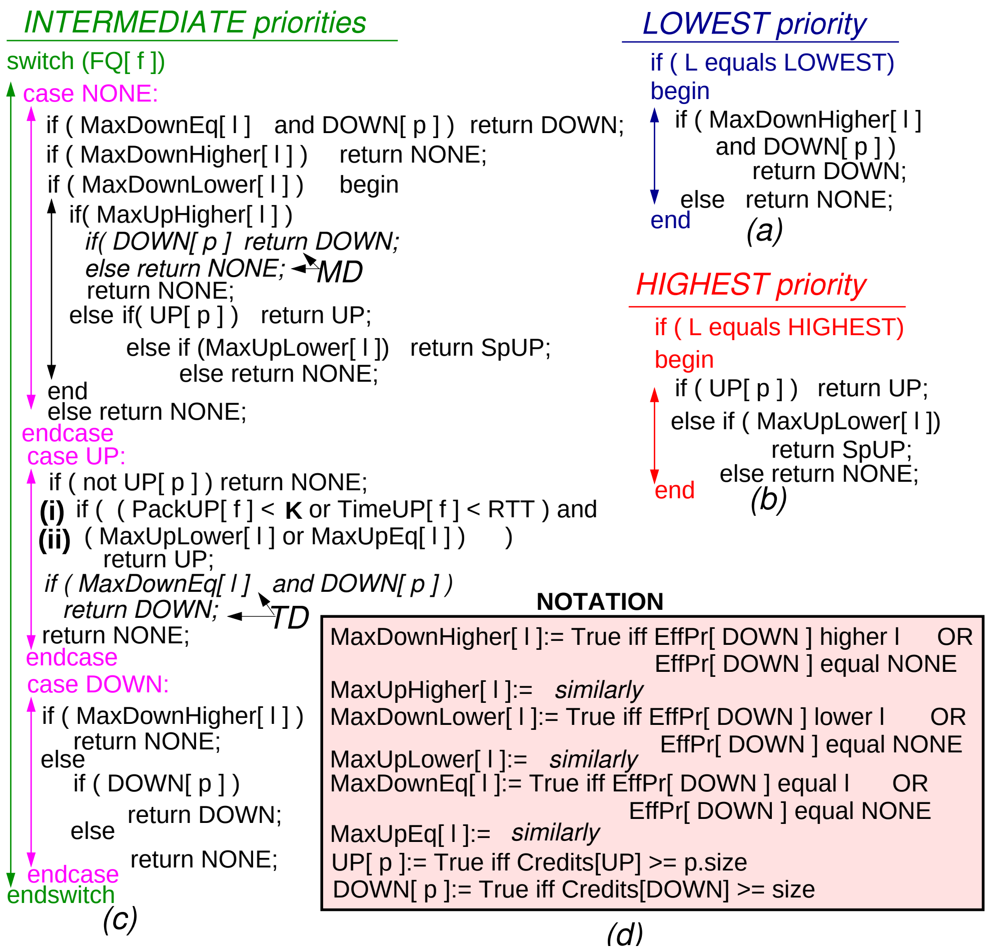

Fig. 18. Alg-2B-ADAPT policy for: **(a)** LOWEST priority; **(b)** HIGHEST priority; **(c)** Intermediate priorities. **(d)** Notation. (The lines in *italics* correspond to alternatives policies (i.e., TD and MD) and are *not* activated in 2B-ADAPT.)

DOWN, the respective register equals NONE[9].

Suppose that a packet p, from flow f with priority l, is processed by Alg-2B-ADAPT. As fig. 18(a,b) shows, the HIGHEST supported priority, l= 0, is mapped only UP whereas, the LOWEST, l= $L-1$, is mapped only DOWN. Intermediate priority levels can be mapped either UP or DOWN –fig. 18(c).

5) Justification: Like SQP-opt, 2B-ADAPT never maps a packet to a queue where packets of higher priority are pending. When this blocking is applied to a packet, p, with priority l, which otherwise would be mapped DOWN, it seems to be even more reasonable than the analogous blocking performed by SQP-opt: in 2B-ADAPT, when higher priority packets are pending DOWN, probably even higher priority levels are queuing UP; thus, contention among packets of priority higher than l at the output corresponding to p is even more probable, than it is when SQP-opt similarly blocks packet p, due to packets of priority higher than l that are pending at the single crosspoint queue.

Regarding intermediate priorities, when (a) FQ [f] equals NONE, i.e., f has no pending packet, 2B-ADAPT first tries to map p DOWN, but if it finds that EffPr [DOWN] is lower than l, it tries to map it UP, to save the delay of the packets pending DOWN. When (b) FQ [f] equals DOWN, 2B-ADAPT will use only the DOWN lane; this property, combined with the discipline at the output, ensures in-order transmission of packets within a flow –see appendix *2B-ADAPT & In-Order Transmission*.

Finally, (c) when FQ [f] equals UP and credits exist for p to go UP, p is found eligible as UP only if:

$$\text{PacketUP } [f] < K \text{ or TimeUP } [f] < \text{RTT (i); and}$$
$$\text{EffPr [UP]} \geq l \text{ (ii),}$$

i.e., no flow of priority l^*, higher than l, is using UP; otherwise p is considered as ineligible. We use criterion (i) in order to keep the UP queue relatively empty, and (ii) to reserve the UP lane for level l^*, assuming that the arrivals of packets within a flow exhibit temporal locality. In 2B-ADAPT, K is set equal to one (1).

Special packets are sent to resolve buffer hogging in the UP queue; it makes little sense to send special packets DOWN, since when a packet, p, is blocked from using DOWN due to lower priority packets that occupy the DOWN buffer space, the algorithm will try to sent p UP. Special packets are sent

[9]When we compare priorities, the NONE value for EffPr [{UP,DOWN}] translates either as HIGHEST, or as LOWEST –see fig. 18(d)

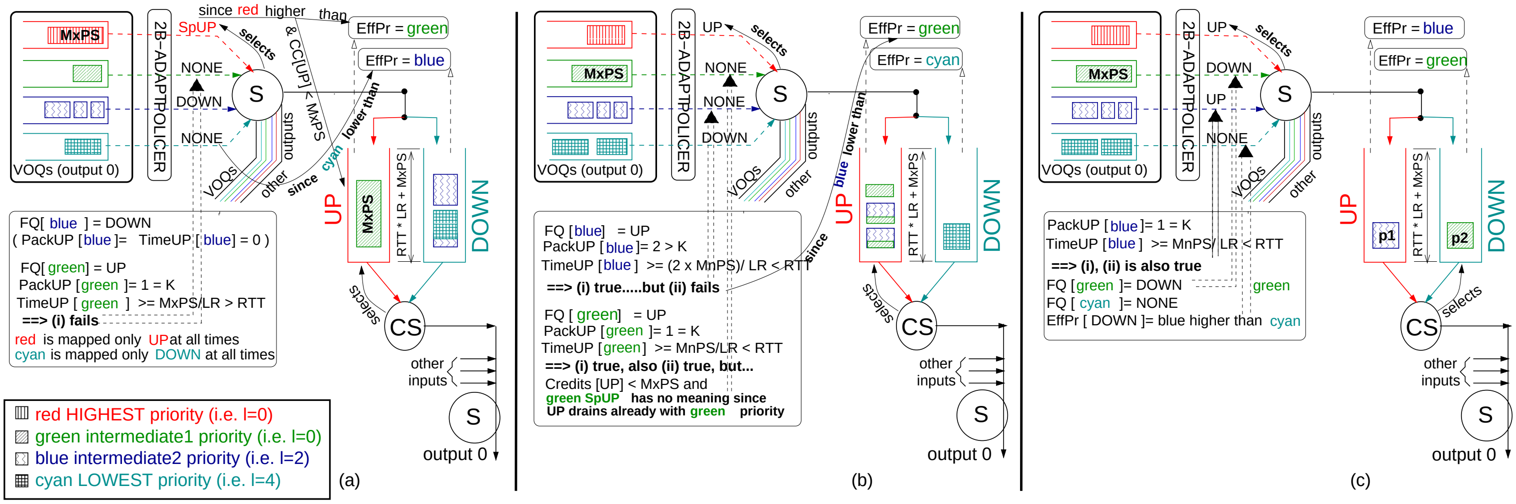

Fig. 19. Example of 2B-ADAPT method with four priority levels. Only the VOQs corresponding to a single crosspoint x are presented; only crosspoint x is presented and the corresponding output scheduler.

when (1) a packet p should be mapped UP but (2) the flow-control constraints the transmission of p and (3) the UP queue drains with lower priority than the priority of p, l.

Note that if we permitted packets to queue at the crosspoint behind higher priority packets, then it could happen that UP drains with priority l, but some packets, P, near the tail of UP, have lower priority than l. In this case it would be more appropriate to examine the priority of the packet at the tail of the UP queue (i.e. the packet that was more recently sent from the input side toward the corresponding UP queue), in order to decide if we must sent a special packet.

6) 2B-ADAPT Example: In fig. 19, we present some examples of the mapping function performed by Alg-2B-ADAPT. Consider the flows that arrive at input i and go to output j. Focus on fig. 19(a); the flow of the HIGHEST priority (i.e., red) is eligible to send a red special packet UP, since UP drains with lower priority (i.e. green) and the UP crosspoint queue cannot contain two MxPS packets. The LOWEST priority (i.e. cyan) is marked as ineligible, since it can use only the DOWN queue, but a higher priority packet is pending DOWN. The green intermediate priority is also considered ineligible. This happens because, FQ [green] equals UP, but criterion (i) fails, i.e. the time that has elapsed since the green packet pending UP started to be transmitted from the input exceeds the RTT. The blue intermediate priority is mapped DOWN, since FQ [blue] equals DOWN.

In fig. 19(b), the packet at the red VOQ will not cause UP to overflow, so the red priority is mapped UP; this does not hold for the green MxPS packet. The packet at the green VOQ, would be otherwise eligible as UP, since criteria (i) and (ii) hold. Since UP contains drains with green priority, it is pointless to send a green special packet UP; so green is considered ineligible. Regarding the blue priority, FQ [blue] equals UP, but (ii) fails since a higher priority packet (i.e., green) started using UP; so blue is marked ineligible. Finally, cyan is eligible to use the DOWN queue.

An irregular but legal behavior is exhibited in fig. 19(c). There, the blue packet (i.e., priority l=2) pending UP, $p1$, must have been sent from the input before the green packet (i.e., higher priority than blue) pending DOWN ($p2$) was: otherwise, $p1$ would be marked as ineligible. A possible scenario explaining how this can happen has as follows. A cyan packet, $p0$, was pending DOWN when $p1$ was sent from the input: otherwise, $p1$ would be either sent as DOWN (if no packet was pending DOWN when $p1$ was sent from the input), or would be considered ineligible (if $p0$ had higher priority than $p1$). The output scheduler selected $p0$ while the header of $p1$ was still in it's way traveling to the crosspoint: otherwise the output scheduler would select $p1$ before $p0$. Before $p2$ was sent DOWN from the input, the credit corresponding to $p0$ arrived, and thus the DOWN queue became empty. Hence, when Alg-2B-ADAPT considered packet $p2$, it found that no packet is pending DOWN and sent $p2$ as DOWN; and now, the DOWN queue contains a packet ($p2$) of higher priority than the packet contained UP ($p1$). The mapping of packets to crosspoint queues, as would be performed by 2B-ADAPT now, is shown in the respective figure.

5.3. 2B-ADAPT Alternative Input Policies

1) 2B-ADAPT-K=n: Observe that criterion (i) is somehow loose, since the size of packets varies considerably when the switch manipulates variable-size packets internally. It is interesting to examine how to adapt the criterion (i), when fixed-size cells are transmitted internally.

In 2B-ADAPT, we set K equal to 1, in order to prevent congestion from the UP lane, and thus reduce the delay of the higher priority levels. Other values for K are also possible; in the next section (2B-ADAPT-noRTT), we show that the distribution of the size of the packets and the RTT value of the switch should be taken under consideration when selecting K. We name the respective policies as 2B-ADAPT-K=n, where n is the value of K. In the simulations that we performed, we observed that by increasing K the higher priority levels receive higher average delay.

2) 2B-ADAPT-noRTT: If we do not include the "TimeUP $[f]$ < RTT" condition in criterion (i), (policy 2B-ADAPT-noRTT –in short, noRTT) then an intermediate priority flow, which uses the UP lane and sends small packets, may not be able to reach full link rate. For instance, if f sends MnPS packets and K is less than $\frac{RTT \cdot LR}{2 \cdot MnPS}$, then noRTT will bound f's rate to $\frac{LR}{2}$. The noRTT policy limits the number of packets that are pending UP, and thus, normally, reduces the delay of the higher priorities. Through simulations we observed that noRTT has actually better performance than 2B-ADAPT under the synthetic traffic patterns that we used, but we decided to include the RTT parameter in the default version of the algorithm, since we consider that this potential blocking can be harmful, especially, since in many cases, high priority flows consist of small packets, for example when high priorities are used for network or TCP control packets.

3) 2B-ADAPT-TD: In either case when criteria (i) or (ii) fail, instead of blocking p, we can try to send it DOWN (policy 2B-ADAPT-TD –in short, TD) –see *TD* marker in fig. 18. Similarly to policy noRTT, 2B-ADAPT-TD has the potential to reduce the delay of the higher priorities – remember that while FQ $[f]$ equals DOWN, f cannot use UP. However, the TD policy can be unjustifiably rushy and aggressive if it causes packets that were queued DOWN to be upgraded to priority l, because TD send p as DOWN while criteria (i) or (ii) had failed:

Normally, (1) the UP queue will drain before DOWN starts receiving service. (This hypothesis is based to that the effective priority UP will be equal to l^* or l, when (ii) or only (i) fail respectively, whereas, the effective priority of DOWN will be normally l after p is enqueued in it, since the effort of the algorithm is to keep the lower active priorities DOWN.) Hence, before p reaches the HOL of DOWN, (i) and (ii) will probably not fail any more. In this case, p would be able to go through the UP lane with similar or smaller delay and without upgrading any other packets. Furthermore, (2) the failure of either criterion indicates congestion at level l or higher within the crossbar, that can delay p in spite of the priority upgrades. In case (2) is true, sending p DOWN can also cause future buffer hogging or HOL blocking, if in the meanwhile that p is queuing DOWN, packets of priority higher to l appear at the input for the same output. Alg-2B-ADAPT simply blocks p, until no packet is pending UP, when (ii) fails, or until one credit for f is received, when only (i) fails.

4) 2B-ADAPT-MD: Another policy that we examined activates the code lines in fig. 18 marked *MD*. The medium-priority-down policy (or 2B-ADAPT-MD –in short, MD) maps a packet, p, DOWN when (1) FQ $[f]$ equals NONE, and (2) l is higher than EffPr [DOWN] and lower than EffPr [UP]. 2B-ADAPT simply blocks p in this case. Policy MD has similar advantages and disadvantages with policy 2B-ADAPT-TD.

Through simulations we verified that these alternatives can reduce the delay of the HIGHEST priority, but at the expense of intermediate priorities experiencing considerably higher delays.

6 . IMPLEMENTATION COST

Currently we are in the process of designing our methods in ASIC designs, in order to verify their feasibility and to discover possible patches in our algorithms, that may be imposed by the hardware implementation. We expect to have gate counts, area, timing and power estimates soon; these will be published in an separate appendix of this report.

6.1. Computing the Effective Priority

The only special circuits required by SQP and 2B-ADAPT inside the crossbar chip are these that compute the effective priority of crosspoint queues. If the inputs do no sent packet of priority l toward a queue, Q, with effective priority higher than l, like in SQP-opt and 2B-ADAPT, then one register maintaining the highest priority, under all packets currently inside Q , suffices. We name this register as priority-register, in short PR.

This register is updated every time a packet, p, is enqueued in Q: we compare the priority of p, l, with the value stored in the respective PR register, and if it is higher or equal we set PR equal to l; when Q becomes empty, PR is reseted to the lowest priority (i.e., $L - 1$ value). The output scheduler uses the PR registers to select one among the non-empty crosspoint queues.

If packets are allowed to queue behind packets of higher priority, like in SQP, then a FIFO-like structure is required at each crosspoint queue to properly compute it's effective priority: when the currently highest priority packet leaves the queue, we have to remember which is the next-highest priority packet, under all the packets that are inside the queue. Hence, either we have to traverse the queue that stores the actual packets, or we have to maintain the next-highest priorities of the packets inside the crosspoint queue, in a separate FIFO-queue.

6.2. Reducing 2B-ADAPT Complexity – 2B-ADAPT-RS

Regarding the complexity of Alg-2B-ADAPT, observe that it must run at packet arrivals or departures, and at credit arrivals, i.e., before or after scheduling at the inputs (see section 3.5). At a packet arrival or departure, Alg-2B-ADAPT has to process a single flow; at a credit, c, arrival, it must compute the eligibility of L flows, i.e., those that can utilize c; so the throughput of the circuit implementing Alg-2B-ADAPT must be higher than L + 2 computations per MnPS time.

Thus, in 2B-ADAPT, it is even more imperative to incorporate an RS like discipline, since the eligibility of a flow is not simply a flow-control check, like it is in SQ or SQP schemes, but requires the intervention of the Alg-2B-ADAPT algorithm. We are currently considering how exactly to match this optimization in the 2B-ADAPT context. With the RS optimization, only one circuit inside each ingress line-card implementing the Alg-2B-ADAPT policy will suffice, if its throughput is equal to three operations every MnPS time.

More generally, the idea is to keep the throughput of the circuit lower than $L + 2$ computations per MnPS time, and to neglect some flows considering them as ineligible for service at the input, until the Alg-2B-ADAPT circuit process them. The good think with RS, is that it neglects flows starting from the lower priority ones, and thus, reserves the lanes to the active flows of the highest priority.

Getting more into the details of the Alg-2B-ADAPT algorithm, note that it is structured by a switch-statement and few comparisons between small values, e.g. 5 bits each, assuming 32 priority levels; depending on the priority level of the packet processed (i.e., HIGHEST, INTERMEDIATE, LOWEST) and on the FQ value (i.e., UP, DOWN, NONE), only a small piece of the code in fig. 18 is executed. Note that the Alg-2B-ADAPT function is coded redundantly in fig. 18, in order to help the reader to understand the policy. In hardware it can be made much more efficient, especially if we exploit parallel execution or pre-computation techniques; such techniques will normally increase the silicon area, but will also increase the throughput of the circuit and will decrease its latency.

7 . Simulations

7.1. Simulation Environment & Models

We created an event-driven simulator to experiment with our methods –for information regarding the structure of the simulator, see also appendix *Simulator Architecture*. In this report, we simulate a 32×32 switch with port speed 10 Gb/s. For simplicity we assume no internal packet-header and thus no speed-up; a 4 byte internal-header, would necessitate to employ 1.1× speed-up at the interface between the ingress line-cards and the crossbar chip.

The VOQs and the crosspoint queues implement cut-through. The RTT is set equal to 400ns (i.e., 500 byte time at 10Gb/s), resulting as the sum of the following delays: (1) 100 ns propagation delay and 32 ns transfer delay (i.e., equal to a MnPS time) before a credit, c, which is generated inside the crossbar chip, reaches the input; (2) 30 ns scheduling delay at the input (i.e., just lower than a MnPS time) plus 76 ns memory access delay at the VOQs and 100 ns propagation delay, before the header of a packet, p, that used c at the input, reaches the crosspoint; (3) 30 ns scheduling delay at the crossbar output, before p starts being transmitted on the output lines.

A reasonable assumption regarding the 100 ns propagation delay is that it includes 60 ns time-of-flight on the interconnect between the crossbar-chip and the ingress line-cards (i.e., around to 18 meters distance), plus 40ns interfacing and pipeline delays within the chips.

We explicitly model variable-size packet arrivals. Unless otherwise commented, the size of each generated packet is always selected independently, using a Pareto distribution: average-packet-size (in short, AvPS) 400, MnPS 40 and MxPS 1500 (bytes). The output destinations of the packets are uniformly distributed to output ports, and all priorities arrive with equal probability. We use this uniform assignment of packets to priority levels, since it stresses the HOL blocking and the buffer hogging behavior.

Regarding the arrival patterns, we use **Poisson** packet arrivals, where packets' interarrival times are exponentially distributed, and also a bursty traffic model, **Bursts60**, where each burst consists of sixty (60) back-to-packet packets. Given that the size of each packet within a burst is selected independently through the aforementioned Pareto distribution, the resulting burst-size distribution is long-tailed with average close to 23 KByte. The idle periods in **Bursts60** are exponentially distributed and the packets within a burst have the same destination and the same priority. **Bursts60** is a synthetic pattern that tries to model worst-case, real traffic scenarios, i.e., situations that can occur under the erratic nature of real network traffic.

We plot packets' average delay just-before-output-service, or equivalently packets' average waiting-time, for each priority in separate, versus the aggregate input load; we present plots only for the most interesting priority levels (p0 stands for the HIGHEST). Note that the duration of a 64Byte cell/segment on 10 Gb/s link equals 0.0512 usec, and reversely 1 usec equals 19.531 cell durations.

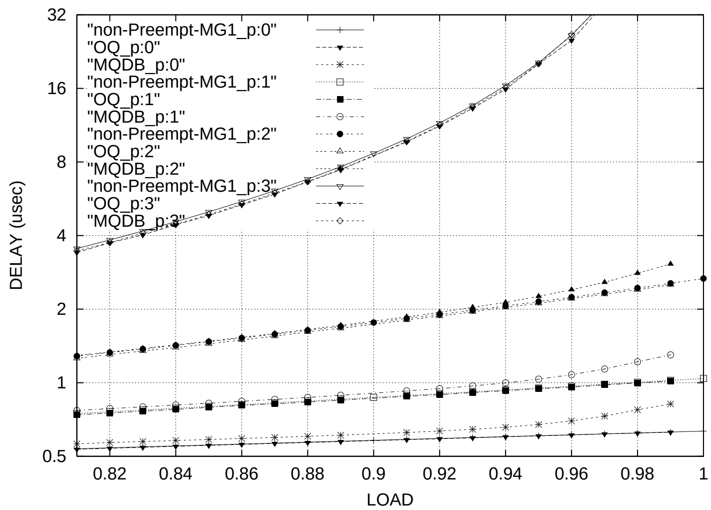

Fig. 20. 4 priority levels (p0 HIGHEST), **Poisson** arrivals. Y axis in logscale.

7.2. Comparisons with OQ and Cell Switching

1) OQ vs MQDB: We start by comparing the MQDB architecture with 2 KByte buffer space per crosspoint queue, to the pure output queuing system (in short, OQ). In the OQ model, the output queues are organized per input and per priority level, and they have infinite size. Note that we have set the minimum packet delay in the OQ system close to RTT/2 (i.e. 200 ns), assuming that packets are generated at the ingress of the switch and are transmitted using cut-through toward the output ports.

We also present the performance of OQ, computed using the theory of *non-preemptive*, priority scheduling, that can be found in [27]. This theory for M/G/1 queues (i.e., Poisson arrivals and arbitrary service time distribution) shows that the average delay of priority level l, D_l depends primarily on the input load of priority levels l^*, higher or equal to l. D_l loosely relates to the load (r_m) of a lower level, m, through the delay that a priority l packet, which just became eligible, can experience due to an ongoing transfer of a packet with priority m; the impact of such events on the delay of high priority flows depends on the aggregate input load and on the packets' service-time distribution, i.e., on the distribution of the size of the packets.

Applying the respective formulas in [27] for the OQ system and for the Pareto packets' size distribution that we use, we find that:

$$D_l = \frac{0.3051 \cdot (\sum_{l^*=0}^{l} r_{l^*} + \sum_{m=l+1}^{L} r_m)}{(1-\sum_{l^*=0}^{l}(1-r_{l^*})) \cdot (1-\sum_{l^*=0}^{l-1}(1-r_{l^*}))} \text{ usec.}$$

As diagram 20 shows for four priority levels, the analytical model perfectly matches the simulated OQ system. It also shows that at input loads up to 0.94, MQDB performs identically to OQ. At higher input loads, small discrepancies occur, that are more prominent for the higher priorities; these can be attributed to the *two* non-preemptive schedulers that

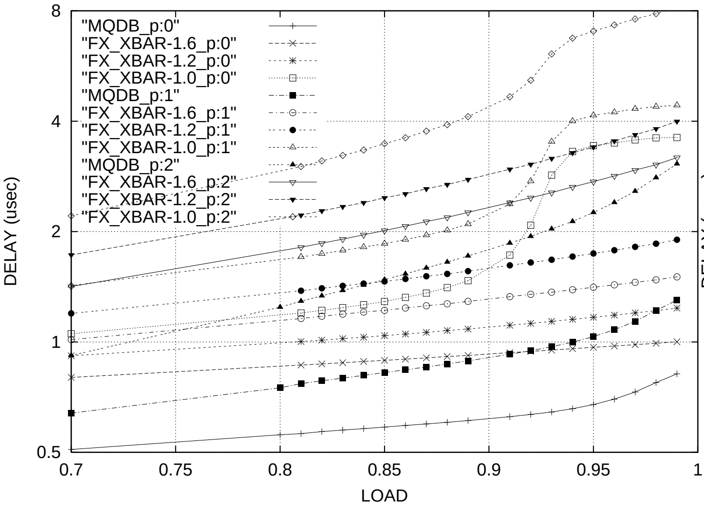

Fig. 21. 4 priority levels (p0 HIGHEST), **Poisson** arrivals. Y axis in logscale.

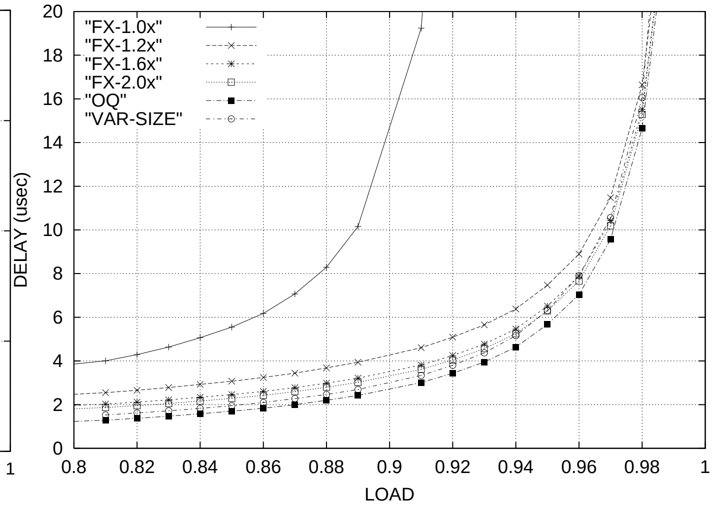

Fig. 22. 1 priority levels Uniform **Poisson** arrivals.

each packet has to pass in MQDB, and to small inefficiencies of MQDB relative to OQ (input queuing vs output queuing).

2) Packets vs Cells: In this experiment we compare the performance of the MQDB switch, with that of a CICQ system, with the same queuing organization as MQDB at the crosspoints, but which segments packets into fixed-size cells in the ingress and reassemblies in the egress line-cards; we name this system as FX-XBAR. In the FX-XBAR model we assume 64 byte cells, and we use speed-up factors (in short, SP) 1.0, 1.2, 1.6 and 2.0×. The buffer space per crosspoint queue is set equal to RTT×IR, where IR stands for the internal line rate, i.e., LR×SP. Finally, the scheduling delay is set equal to one cell time –in short CT, i.e., 64byte/IR–, and the credit rate per input port is set equal to one credit every CT. All other parameters are the same as with MQDB.

Diagram 21 compares MQDB with FX-XBAR under four priority levels and **Poisson** arrivals. As the diagram shows, FX-XBAR-1.0× saturates near 0.9 load where all priorities perform purely. This is primarily due to the segmentation overhead which increases the load that must be handled by the switch core. In FX-XBAR-1.6×, the average delay of p0 is not affected by the input load but is still higher than in MQDB: by contrast with MQDB, in the FX-XBAR, the delay of a p0 packet, p, includes one transfer delay of p (toward reassembly).

Normally, non-preemptive scheduling has lesser impact on the delay of high priorities with cell switching than with variable-packet-size switching: a high priority packet that now becomes eligible at the input or the crossbar-output scheduler, cannot not be delayed by more than one CT waiting for an ongoing, lower priority, transfer to end; in MQDB it may wait up to a MxPS time. Note, that this does not hold in the egress line-cards of FX-XBAR, where the reassembled packets are served non-preemptively.

Finally observe that by measuring packets' waiting time, we favored FX-XBAR: if two p0 packets arrive at time t from different inputs in an idle FX-XBAR, the smaller one will be reassembled firsts, due to smaller transmission delay, and thus will be served first; no such bias is evident in the MQDB system. It is well known though, that a smaller-packet-first server reduces packets' average waiting time.

In the experiment corresponding to diagram 22 we used uniform **Poisson** arrivals and a single priority level, i.e., no priority discrimination; it contains plots for the FX-XBAR switch with several speed-up factors denoted as FX-1.0, FX-1.4, FX-1.6 and FX-2.0×, for the OQ switch, and for the variable-packet-size CICQ switch, that we propose in this report, employing 2 KByte per crosspoint, denoted as VAR-SIZE in the diagram. We can see that the VAR-SIZE switch outperforms the FX-XBAR for any of the speed-up factor that we used, and is very close to the OQ system. This is a very interesting results, since much effort is paid nowadays in the CICQ switches transmitting cells, like the FX-XBAR model does.

In [16] we show through simulations, that the variable-packet-size crossbar presented in this report, with no speed-up, performs similarly to the iSLIP switch with speed-up 2.0×; it performs much better than the iSLIP switch, when the latter employs lower speed-up values.

7.3. Evaluating our Methods

1) MQDB vs Shared Buffer/Queue: Here, we examine the performance of MQSB, MQSB-RS, SQ, SQP, SQP-RS and SQP-opt, all with crosspoint buffer space equal to 2 KByte. To model MQSB (see sec. 2.2), we use a single credit counter at each input for each corresponding crosspoint, that counts for the available space (2 KByte) of all logical queues in that crosspoint. In an actual MQSB system, memory fragmentation could worsen performance.

In diagram 23 we compare MQDB, MQSB, SQ and SQP-opt under **Poisson** arrivals and four priority levels; we see that at 0.81 input load, SQP-opt and MQSB perform close to MQDB while SQ cannot discriminate well low from high priority traffic: this is due to HOL blocking. With increasing

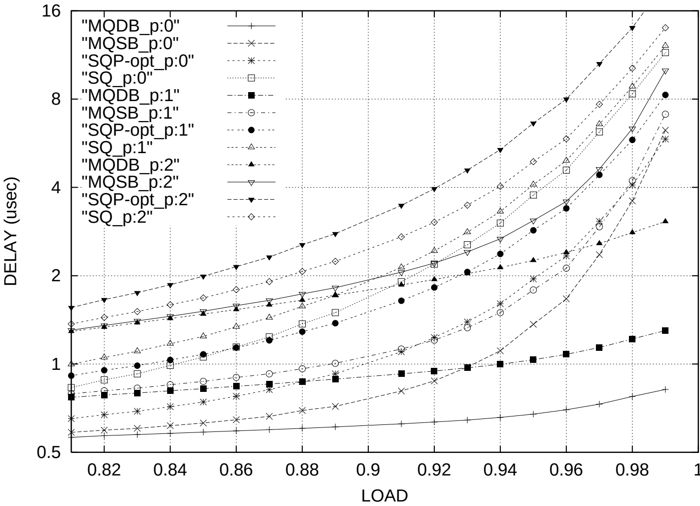

Fig. 23. 4 priority levels (p0 HIGHEST), **Poisson** arrivals. Y axis in logscale.

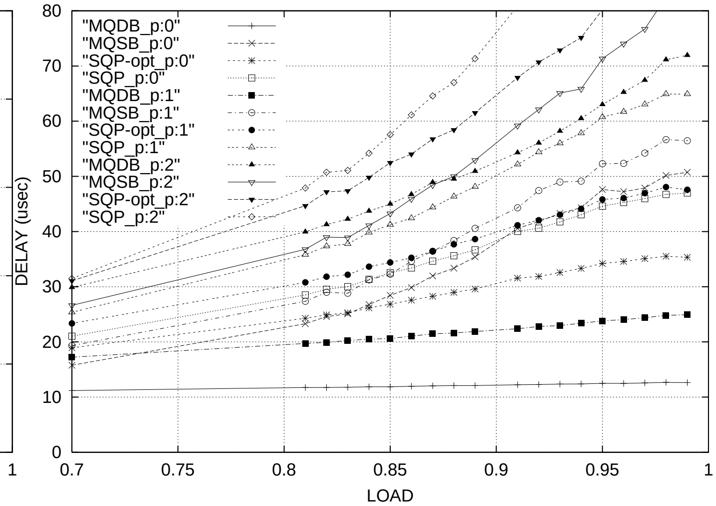

Fig. 25. 4 priority levels (p0 HIGHEST), **Bursts60** arrivals.

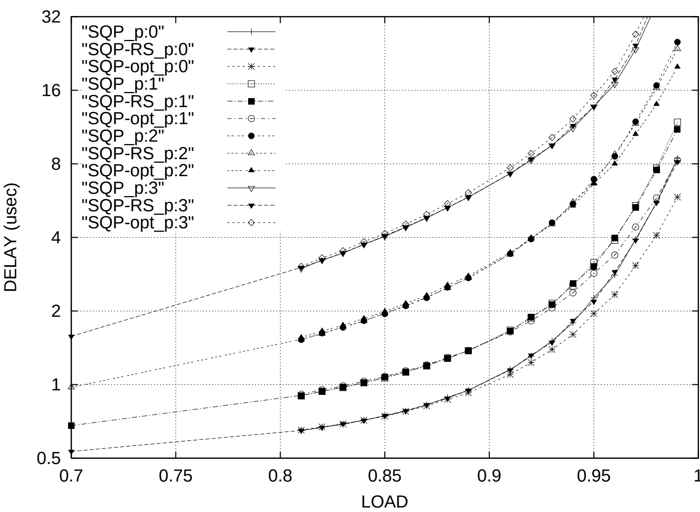

Fig. 24. 4 priority levels (p0 HIGHEST), **Poisson** arrivals. Y axis in logscale.

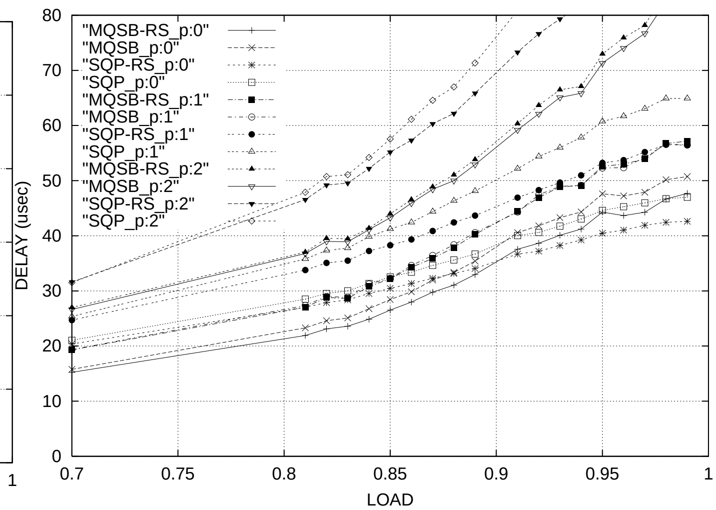

Fig. 26. 4 priority levels (p0 HIGHEST), **Bursts60** arrivals.

load, all systems except MQDB downgrade similarly with SQ; in SQP-opt this happens at lower load than in MQSB, but MQSB downgrades more sharply at higher input loads, i.e., near 0.97: this is due to buffer hogging.

Diagram 24 compares SQP-opt with SQP and SQP-RS under the same traffic as before. It shows that SQP-opt assigns lower delay to levels {p0, p1, p2} compared to SQP –{5, 8, 20} and {8, 12, 25} usec respectively, at 0.99 input load–, while p3 receives almost the same service. Under this traffic pattern, SQP-RS performs almost identically with SQP.

In diagram 25, where we compare MQDB, MQSB, SQP-opt and SQP under **Bursts60** arrivals, performance degradation in all shared-memory systems is apparent, and the superiority of SQP-opt compared to SQP is more evident. At low load (i.e., 0.5 to 0.7), MQSB performs better than SQP and SQP-opt since it employs multiple queues that eliminate HOL blocking, and buffer hogging is not harmful when the crosspoints' buffers are relative empty; it performs much worse than the SQP systems at higher input load, where the crosspoint buffers fill more frequently.

Finally, for diagram 26 we used the same bursty traffic model and we present plots for MQSB-RS, MQSB, SQP-RS and SQP. We can see that the RS discipline improves the performance in both the MQSB and the SQP system at high input load: SQP-RS assigns the lowest p0 average delay among all systems, followed by SQP and MQSB-RS. Regarding the p1 priority level, SQP-RS and the MQSB systems perform almost identically, while SQP performs worse. MQSB assigns lower average delay than SQP-RS to priority level p2.

2) MQDB vs 2B-ADAPT: Following, we examine the performance of 2B-ADAPT, with 2 KByte buffer space assigned to each crosspoint queue. Diagram 27 is with four priority levels and **Poisson** arrivals. It compares MQDB, with 2B-ADAPT and with MQSB-4K, i.e., the MQSB system with 4 KByte buffer space at each crosspoint. We used MQSB-

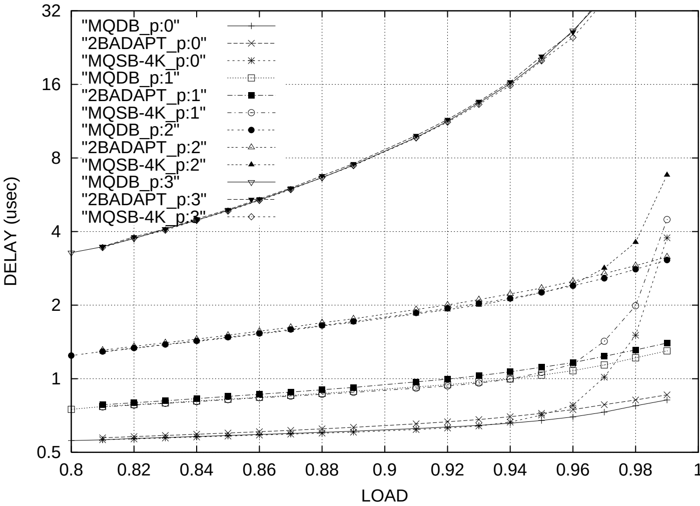

Fig. 27. 4 priority levels (p0 HIGHEST), **Poisson** arrivals. Y axis in logscale. MQSB-4K plots start at 0.8 load.

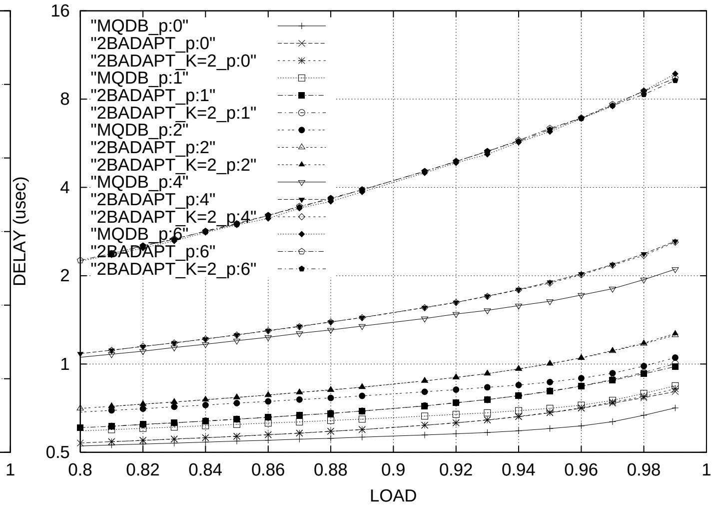

Fig. 29. 8 priority levels (p0 HIGHEST), **Poisson** arrivals. Y axis in logscale. 2B-ADAPT-K2 plots start at 0.8 load.

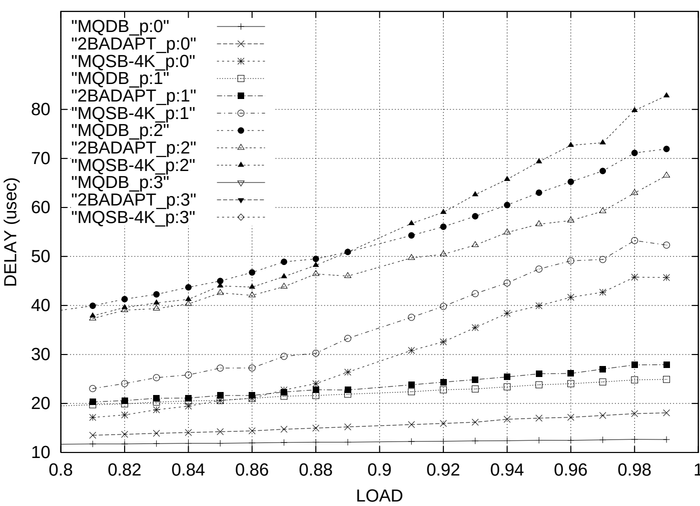

Fig. 28. 4 priority levels (p0 HIGHEST), **Bursts60** arrivals. MQSB-4K plot starts at 0.8 load

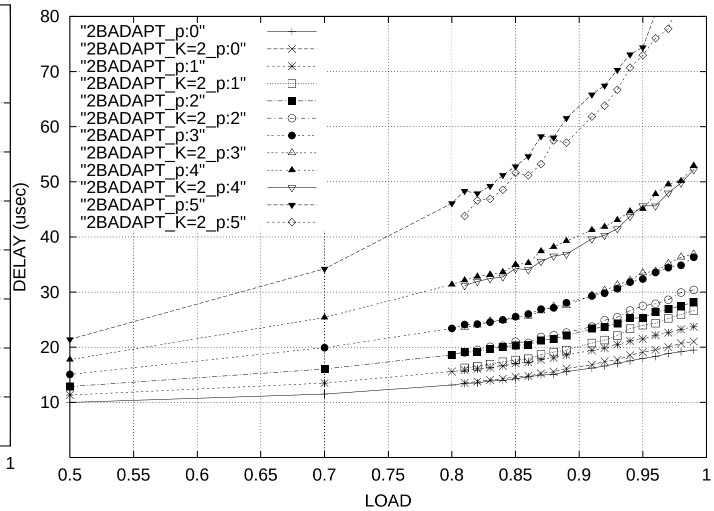

Fig. 30. 8 priority levels (p0 HIGHEST), **Bursts60** arrivals. 2B-ADAPT-TD plot starts at 0.8 load

4K instead of MQSB, since the MQSB-4K system requires the same amount of memory with 2B-ADAPT. As we can see in the diagram, 2B-ADAPT performs almost *identically* to MQDB, while MQSB-4K degrades at input load higher than 0.95.

Next we compare the same systems under **Bursts60** arrivals, again with four priority levels. As we can see in diagram 28, at input load higher than 0.8, the 2B-ADAPT plot has discrepancies relative to that of MQDB; these discrepancies grow with increasing load, but never exceed 50% –p0 average delay 18 vs 12 usec at 0.99 load. MQSB-4K again performs much worse at high loads.

Diagram 29 is for eight (**8**) priorities and **Poisson** arrivals. It compares 2B-ADAPT to MQDB and to 2B-ADAPT-K=2, i.e., an alternative 2B-ADAPT policy that sets the K parameter in criterion (i) equal to two –see section 5.3.1 . The diagram shows that 2B-ADAPT and 2B-ADAPT-K=2 perform similarly under this scenario. Compared to MQDB, 2B-ADAPT exhibits negligible discrepancies that are observable only at input load higher than 0.85: the maximum discrepancy under all priorities is near 15%, i.e., p0's average delay, 0.8 vs 0.7 usec, at 0.99 load. Under **Bursts60** arrivals, the 2B-ADAPT-K=2 policy assigns higher average delay to all high priority level, than the default 2B-ADAPT (diagram 30). In general, we observed that increasing K worsens performance, regarding the three or four highest priority levels.

Diagram 31 compares 2B-ADAPT to MQDB and to 2B-ADAPT-noRTT under eight priorities and **Bursts60** arrivals; it shows that the noRTT alternative of 2B-ADAPT policy reduces the delay of all high priorities compared to the default 2B-ADAPT. Compared to MQDB, the maximum discrepancies occur at the average delay of priority p0: 75% and 35% in 2B-ADAPT and in noRTT respectivelly, at 0.99 load. Re-

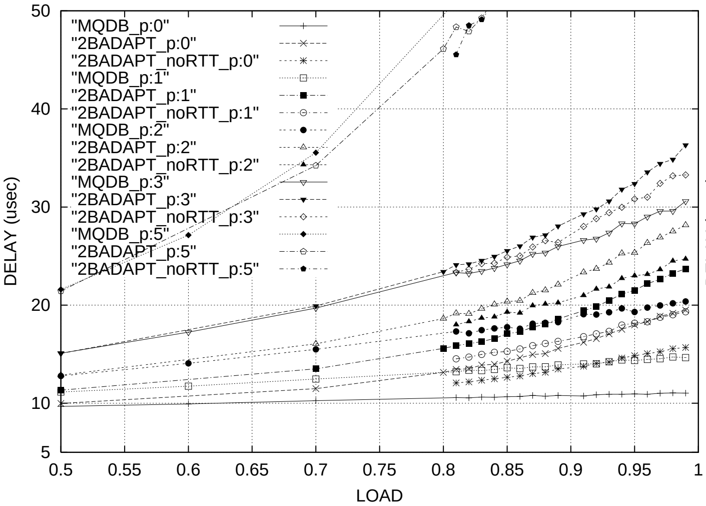

Fig. 31. 8 priority levels (p0 HIGHEST), **Bursts60** arrivals. 2B-ADAPT-noRTT plot starts at 0.8 load

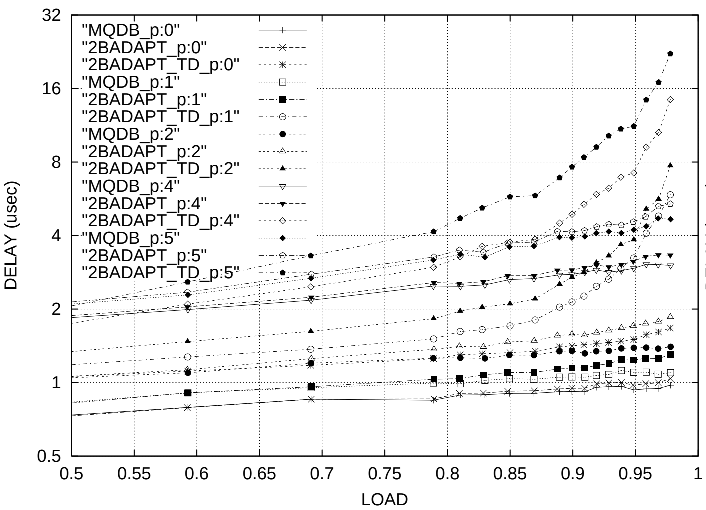

Fig. 32. 8 priority levels (p0 HIGHEST), **SynthBackb** arrivals. Y axis in logscale.

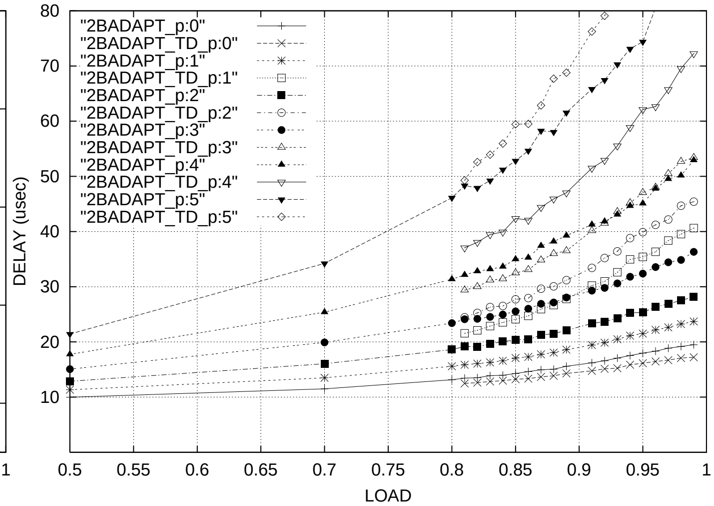

Fig. 33. 8 priority levels (p0 HIGHEST), **Bursts60** arrivals. 2B-ADAPT-TD plot starts at 0.8 load

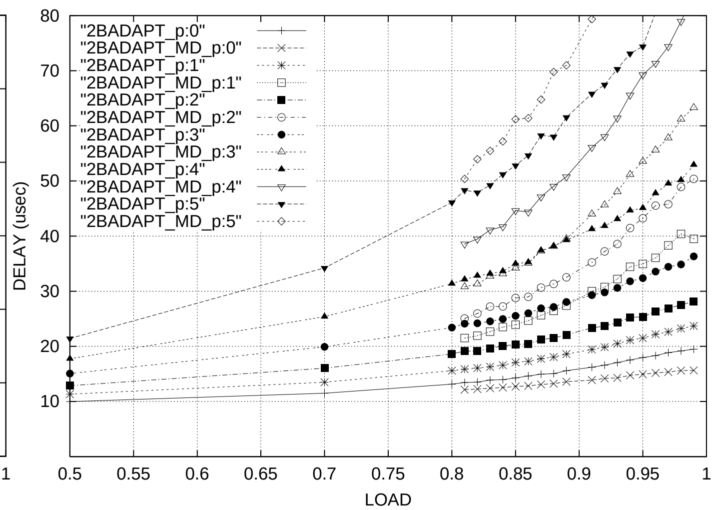

Fig. 34. 8 priority levels (p0 HIGHEST), **Bursts60** arrivals. 2B-ADAPT-MD plot starts at 0.8 load

garding lower priority levels, the corresponding discrepancies are considerably smaller (less than 33%). A noteworthy point, relative to section 5.3.2 , is that in the average case, the noRTT policy will not bound flows' rate, if $K\cdot$ AvPS $>$ RTT·LR.

Next we use a synthetic traffic pattern, **SynthBackb**, that tries to emulate as much as possible backbone, realistic IP traffic. In summary, under the **SynthBackb** pattern, the packet arrivals at an input line-card are generated by multiplexing thousands of interactive (IC) and bulk (BC) "conversations" in a FIFO queue; an IC is modeled as a Poisson process that sends 125 packets with size 40 to 44 bytes, while BC as a burst with average size 8 KByte [16].

Diagram 32 contains plots for MQDB, 2B-ADAPT and 2B-ADAPT-TD under **SynthBackb** arrivals and eight priority levels; each generated conversation is independently mapped to one level, using a uniform distribution. The diagram shows 2B-ADAPT to perform very close to MQDB. The TD alternative policy performs very poorly compared to 2B-ADAPT, for the reasons described in section 5.3.3 : TD dramatically increases the delay of all priorities except p7.

Under **Bursts60** and **Poisson** arrivals we found that 2B-ADAPT-TD and 2B-ADAPT-MD (see section 5.3.4) perform better than 2B-ADAPT regarding p0 and p7 and worse for all other priorities; diagrams 33 and 34 demonstrate this behavior under the bursty traffic model, concerning the TD and MD alternative policies respectively.

3) Imbalanced Destinations: Finally we consider the imbalanced traffic pattern, where each input port, i, has a distinct, "favored" output port, j, e.g. $j = i$; with probability 0.5 a packet that arrives from input port i is destined to output port j, whereas, the rest of the packets are uniformly distributed to the remaining outputs. Diagram 35 is for eight priority levels and the aforementioned imbalanced traffic model for

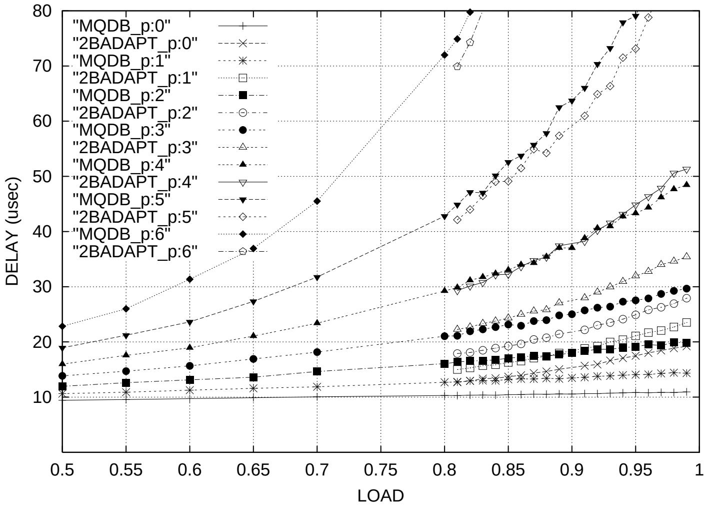

Fig. 35. 8 priority levels (p0 HIGHEST), **Bursts60**, *imbalanced* arrivals . 2B-ADAPT plot starts at 0.8 load

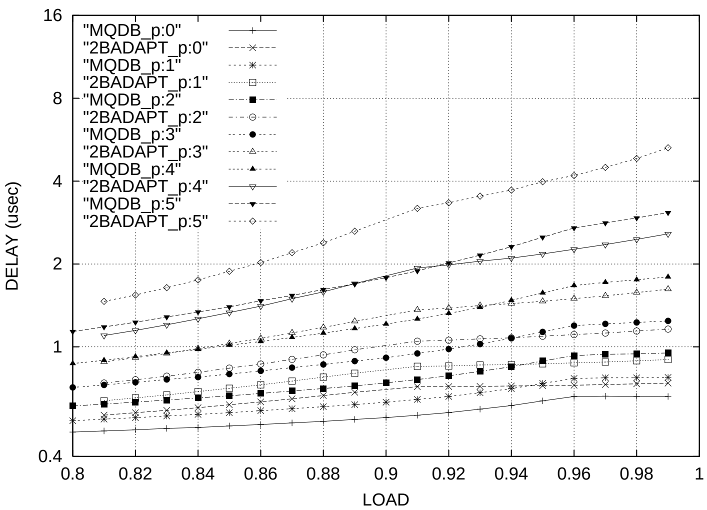

Fig. 36. 8 priority levels (p0 HIGHEST), **Poisson**, *imbalanced* arrivals . 2B-ADAPT plot starts at 0.8 load

Bursts60 arrivals (all packets within a burst have the same destination); we present plots for MQDB and 2B-ADAPT. As we can see, even under this non-uniform model, 2B-ADAPT does not degrade considerable. Diagram 36 is again for eight priorities, **Poisson** arrivals, and the imbalanced destinations distribution. It shows that the higher priority levels in 2B-ADAPT do not degrade considerably compared to MQDB; though, the lower ones do degrade, but this is due to that large crosspoint buffers are required to reach high throughput under this imbalanced traffic pattern –see section 3.7.2 –, and it is not very related to priorities handling. By employing a more sophisticated scheduler at the inputs or at the crossbar outputs in place of the simplistic FQ that we currently employ, or by increasing the buffer space of the crosspoint buffers/queues –remember that MQDB has 4 times more buffering–, we consider that these discrepancies will be reduced.

CONCLUSIONS

We presented a novel variable-packet-size, multiple-priority CICQ switch. We demonstrated that this switch architecture, with no internal speed-up, can provide very good performance comparable only to that of practical input-queued systems that employ speed-up higher than two. Although we compared it only with a fixed-size cell crossbar, the reader can examine the literature regarding buffered crossbars, to make the respective comparisons with unbuffered input-queuing architectures. We also proposed methods employing a minimal number of lanes at each crosspoint, (i.e., one or two) which effectively support multiple priority levels. Our results indicate, that implementing more than two crosspoint lanes does not worth the associated, high incremental cost, since our system with two such lanes performs very close to the ideal system which employs a separate crosspoint lane for each priority: under a worst case scenario the maximum relative discrepancy is 75% whereas, in the typical case it is only 15%. Finally, we demonstrated that the only considerable incremental cost of our methods is located in the ingress line-cards.

ACKNOWLEDGMENTS

I would like to thank my supervisor Manolis Katevenis for his proposal to consider *variable-packet-size* buffered crossbar, and for his guidance and contribution in the development of the ideas presented in this report. I would also like to thank Georgios Passas for participating in the development of the simulator, Kostas Harteros, Georgios Sapunjis, Giannhs Papaeystathiou and Dionyshs Pnevmatikatos for discussing the subject and for their useful comments. Finally, I would also like to thank Dimitris Simos for transforming our abstract/software models into hardware, and Mixalis Ligerakis for his technical support.

REFERENCES

[1] T. Anderson, S. Owicki, J. Saxe, C. Thacker: "High-Speed Switch Scheduling for Local-Area Networks", *ACM Trans. on Computer Systems*, vol. 11, no. 4, Nov. 1993, pp. 319-352.

[2] R. LaMaire, D. Serpanos: "Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues", *IEEE/ACM Trans. on Networking*, vol. 2, no. 5, Oct. 1994, pp. 471-482.

[3] N. McKeown: "The iSLIP Scheduling Algorithm for Input-Queued Switches", *IEEE/ACM Trans. on Networking*, vol. 7, no. 2, April 1999, pp. 188-201; http://tiny-tera.stanford.edu/∼nickm/papers/ToN_April_99.pdf

[4] Y. Li, S. Panwar and H.J. Chao, "On the performance of a dual round-robin switch," Proc. IEEE INFOCOM'01, pp. 1688-1697, 2001; http://citeseer.nj.nec.com/li01performance.html

[5] N. Ni, L. N. Bhuyan: "Fair scheduling for Input Buffered Switches", citeseer.nj.nec.com/482342.html

[6] P. Krishna, N. Patel, A. Charny, R. Simcoe: "On the Speedup Required for Work-Conserving Crossbar Switches", *IEEE J. Sel. Areas in Communications*, vol. 17, no. 6, June 1999, pp. 1057-1066.

[7] D. Stephens, H. Zhang: "Implementing Distributed Packet Fair Queueing in a scalable switch architecture", *Proc. INFOCOM'98 Conf.*, San Francisco, CA, March 1998, pp. 282-290.

[8] T. Javidi, R. Magill, and T. Hrabik: "A High-Throughput Scheduling Algorithm for a Buffered Crossbar Switch Fabric" *Proc. IEEE Int. Conf. on Communications (ICC'2001)*, Helsinki, Finland, June 2001, vol. 5, pp. 1586-1591.

[9] R. Rojas-Cessa, E. Oki, and H. Jonathan Chao: "CIXOB-k: Combined Input-Crosspoint-Output Buffered Switch", *Proc. IEEE GLOBECOM*, 2001, vol. 4, pp. 2654-2660.

[10] N. Chrysos, M. Katevenis: "Weighted Fairness in Buffered Crossbar Scheduling", *Proc. IEEE Workshop High Perf. Switching & Routing (HPSR 2003)*, Torino, Italy, June 2003, pp. 17-22; http://archvlsi .ics.forth.gr/bufxbar/
[11] G. Georgakopoulos: "Few buffers suffice: Explaining why and how crossbars with weighted fair queuing converge to weighted max-min fairness", *Submitted to ICC'04*; http://archvlsi.ics.forth.gr/bufxbar/
[12] N. Chrysos, M. Katevenis: "Transient Behavior of a Buffered Crossbar Converging to Weighted Max-Min Fairness", Inst. of Computer Science, FORTH, Heraklion, Crete, Greece, August 2002, 13 pages. http://archvlsi.ics.forth.gr/bufxbar/
[13] Ferdinand Gramsamer e.a.: "Flow Control Scheduling" , revised and extended version of *ICCCN 2002*, Oct. 14-16, Miami, FL, pp. 438-443
[14] Taiwan Semiconductor Manufactoring Company Ltd; http://wwww.tsmc.com
[15] F. Abel, C. Minkenberg, R. Luijten, M. Gusat, I. Iliadis: "A Four-Terabit Packet Switch Supporting Long Round-Trip Times", *IEEE Micro Magazine*, vol. 23, no. 1, Jan./Feb. 2003, pp. 10-24.
[16] Manolis Katevenis, Giorgos Passas, Dimitris Simos, Giannhs Papaefstathiou and Nikos Chrysos "Variable Packet Size Buffered Crossbar (CICQ) Switches" http://archvlsi.ics.forth.gr/bufxbar
[17] K. Yoshigoe, K. Christensen: "A Parallel-Polled Virtual Output Queued Switch with a Buffered Crossbar", *Proc. IEEE Workshop High Perf. Switching & Routing (HPSR 2001)*, Dallas, TX, USA, May 2001, pp. 271-275; http://www.csee.usf.edu/∼christen/hpsr01.pdf
[18] Wael Noureddine, Fouad Tobagi: "Improving the Performance of Interactive TCP applications Using Service Differentiation", *Proc. Infocom'02 Conf.*, New York, June 2002.
[19] Wu-chang Feng et. al. "Adaptive Packet Marking for Providing Differentiated Services in the Internet", *Proc. of Int. Conf. on Network Protocols* Oct. 1998; http://citeseer.nj.nec.com/feng98adaptive.html
[20] P. Goal, H. M. Vin and H. Cheng: "Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks", *Proc. of ACM-SIGCOM '96, Palo Alto, CA, August 1996, pp 157-168.*
[21] S. Nojima, E. Tsutio, H. Fukuda, M. Hashimoto: "Integrated Services Packet Network Using Bus Matrix Switch", *IEEE J. Sel. Areas in Communications*, vol. 5, no. 8, October 1987, pp. 1284-1292.
[22] M. Katevenis: "Fast Switching and Fair Control of Congested Flow in Broad-Band Networks", *IEEE J. Sel. Areas in Communications*, vol. 5, no. 8, October 1987, pp. 1315-1326.
[23] K. Yoshigoe, K. Christensen, A. Jacob: "The RR/RR CICQ Switch: Hardware Design for 10-Gbps Link Speed", *Proc. IEEE Int. Performance, Computing, and Communications Conf.*, April 2003, pp. 481-485; http://www.csee.usf.edu/∼christen/ipccc03.pdf
[24] Lotfi Mhamdi, Mounir Hamdi: "A High-Performance Scheduling Algorithm for Internally Buffered Crossbar Switches", *Proc. IEEE Workshop High Perf. Switching & Routing (HPSR 2003)*, Torino, Italy, June 2003
[25] N. Chrysos: "Weighted Max-Min Fairness in a Buffered Crossbar Switch with Distributed WFQ Schedulers: a First Report", *Technical Report FORTH-ICS/TR-309, Inst. of Computer Science, FORTH, Heraklio, Crete, Greece; M.Sc. Thesis, Univ. of Crete (advisor: M. Katevenis); April 2002, 150 pages.* http://archvlsi.ics.forth.gr/bufxbar (Also a Greek shorter version appears in the same site)
[26] N. Chrysos: "Implementing Two Circular Queues in a Shared Buffer", *Inst. of Computer Science, FORTH, Heraklio, Crete, Greece November 2003.* http://archvlsi.ics.forth.gr/bufxbar
[27] L. Kleinrock, "Queueing Systems, vol. 2", *Wiley, New York, 1975*

APPENDIX

Simulator Architecture

1) Events: We have created an event-driven simulator to evaluate the performance of the proposed architecture under variable-size packets load. The simulator was built from the scratch in C++ code. We decided to create our own code for the creation/handling of events, instead of using some standard event-driven simulator (e.g. C-SIM) so that we can optimize it for the purposes of our project. In addition, building upon the events handling and ordering methods of a black box simulator can sometimes turn to be frustrating, especially when you lack the necessary experience.

The resulting simulator uses a single event-heap to dispatch and schedule events; the event-heap is implemented as binary tree from the standard C++ template. In order to reduce execution time, we tried to minimize the number of events that are generated by the transmission of a single packet, without compromising the accuracy of the simulator. The execution time is proportional to:

#PKTS$\cdot$ *max* [log (E [#PKTS_IN_SYSTEM]), N^2]

The most important types of events that we have used are:

- PACKET_ARRIVAL: signals the arrival of a new packet at an ingress line card
- CREDIT_ARRIVAL: signals the arrival of a credit in the ingress line card
- PACKET_ARRIVAL_AT_CROSSPOINT: notifies that the header of a packet arrived at a crosspoint
- PACKET_TRANSMISSION_ENDED: notifies that the transmission of a packet on a crossbar line ended
- SCHEDULING_START: marks the start of a new scheduling operation/phase at a contention point
- SCHEDULING_ENDS: marks the end of a scheduling operation, and also the time when the operations associated with the transmission of the selected packet start.

2) Time Ordering & Events Handling: The event-structure, contains a type-id, a module-id signifying the modules (i.e., the input and/or output port) that are associated with the event, and of-course a time-field that holds the time when the event must occur. Events are fetched from the event-heap in non-decreasing order with respect to their time-field. All concurrent events are dispatched and are directed, in random order, to the appropriate module for a first-hand processing.

Modules' state variables, which are not affected by the order that events are handled, are updated in this phase. State variables that are subject to race conditions are not updated during this phase, since the order that concurrent events are first-hand processed is actually arbitrary. In the presence of such an event, the associated modules switch into an intermediate state; after all concurrent events are first-hand processed the simulator checks which modules might need to change critical state and calls the appropriate routines.

This two phase handling of events maintains causality and helps debugging. Assume for instance that a credit, c, arrives at the ingress line card i, at simulation time t; before t, the only flow with non-empty VOQs was flow f, which, also, is interested for credit c. Additionally, suppose that the link controller is idle at time t, i.e., no packet is being transmitted toward the crossbar from input i. This CREDIT_ARRIVAL event triggers the controller at input i to increment the credit counter of the respective flow during the first phase, since this variable is not subject to race conditions.

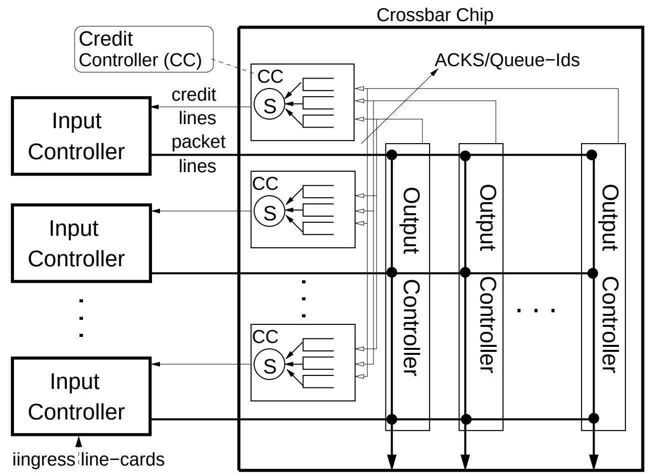

Fig. 37. The system controller; focus on the structure of the credit controller used in the simulation.

But scheduling in input i is not performed during this phase, even though the c credit might be responsible for f becoming eligible. If scheduling was performed and f was selected, then the link controller at input i would switch into BUSY state during this first phase. But if a PACKET_ARRIVAL event, concurrent to the credit arrival, had not been yet processed when the scheduler switched into the BUSY state, then this event (i.e., a new packet arrival at input i) would not have been taken under consideration by the scheduler. At any case the scheduler's decision would depend on the order that these concurrent events were processed.

In order to maintain causality, when the CREDIT_ARRIVAL event is first-hand processed by the controller at the input link, the controller enters the TO_SCHEDULE state and only after all concurrent events are processed, the simulator re-awakes the controller to start scheduling. So the simulator handles events in two phases, in the same fashion that many hardware description languages do (see fig. 14).

3) Simulator Components: The simulator for the buffered crossbar contains the following core modules:

- *Input Controller*: performs the operations of an ingress line card
- *Output Controller*: performs the operations that occur at a column (output) of the crossbar-chip
- *Credit Controller*: manages the credits that are pending inside the crossbar

There are N –equal to the port number– instances of input, output and credit controllers that operate independently of each

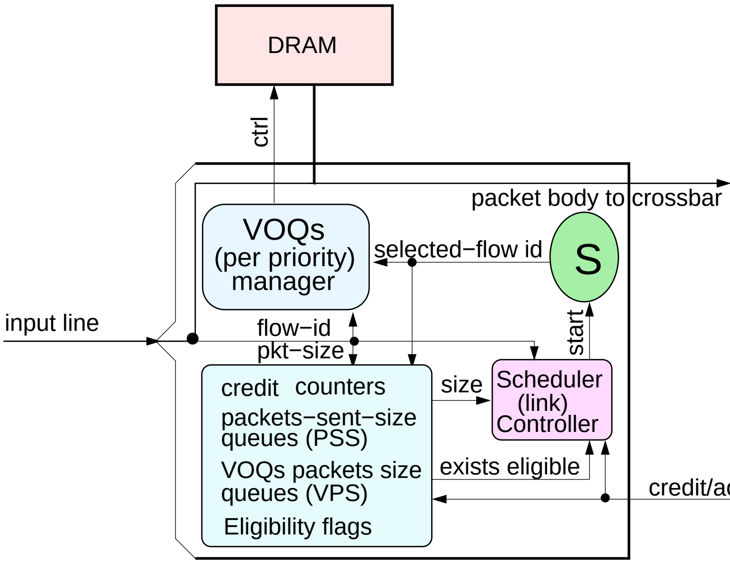

Fig. 38. The data structures in an input controller.

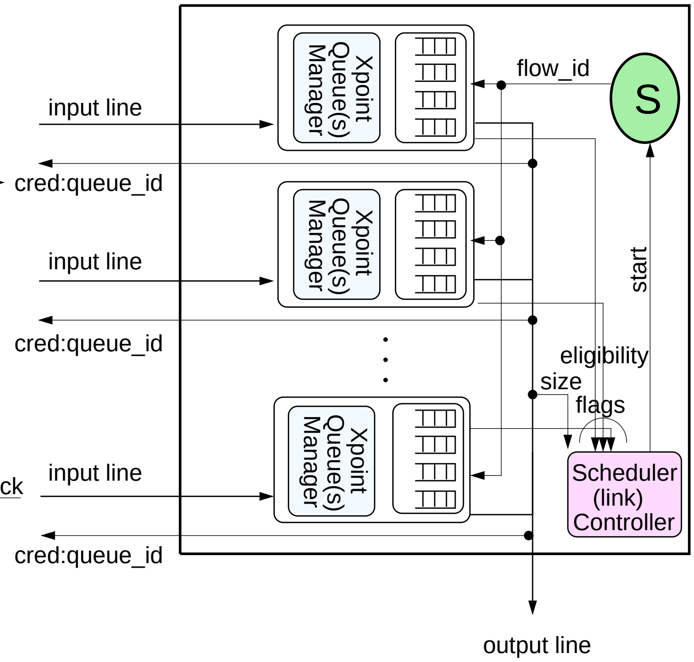

Fig. 39. The data-structures in an output controller.

other.

An input controller maintains the models for the VOQs and the VPS FIFOs; also, it contains the structures required for flow control operation, i.e., the credit counters and the PSS FIFOs. Statistics regarding the state of the crosspoint buffers that are fed by an input (packets no yet acknowledged, their size and priority etc.) are also maintained by the associated input controller, in accordance to the particular scheme. The input controller is responsible for queuing and scheduling packets at the VOQs, subject to flow control. We assume that it is an autonomous part of the ingress line card, responsible for the transmission of packets toward the crossbar.

An output controller maintains the crosspoint buffers and queues in a column of the crossbar [10]. Different schemes that we explore in this paper have different number of buffers and different queuing organization at the crosspoints. Scheduling and forwarding toward the output lines are performed by the output controller. In addition, the output controller informs the credit controllers for the credits generated by the transmission of packets on the corresponding crossbar's output.

A credit controller maintains the queues that store credits that are pending inside the crossbar chip; it is also responsible for the scheduling and the transmission of the credits toward the input controller. Separate credit queues per input, output port and per priority level have been modeled, but we have mostly used only the per input port partitioning. Unless otherwise commented, we assume a FIFO with unlimited capacity inside the crossbar chip, at the interface of the chip with every ingress line card, which stores the credits generated from the queues at the corresponding crosspoints. In addition, we assume that the write/read rate to each separate credit queue is N+1 credits-messages per MnPS time. Credits are send from the crossbar chip toward each ingress line cards, by a separate interface (i.e., lines/cables in an actual system), than packets are; the bandwidth of this interface is a parameter in our simulations.

In configurations where speed-up is employed, a separate output controller is needed to emulate the operation of the egress line cards. Operations of this controller include per flow queuing, reassembling and scheduling/forwarding of packets on the external output lines. Currently we use internal speed-up only at configurations where fixed-size-cells are transmitted internally into the switch. In these configurations we assume that the reassembly buffers have infinite capacity –no flow control between the crossbar chip and the egress line cards is implemented–, and we use non-preemptive strict priority scheduling combined with simple pointer-based RR; the latter discipline is used to brake priority ties, i.e., to select one flow, when multiple flows of the same priority are eligible (i.e., have at least one complete packet at their reassembly queue) for service.

Additional system modules are the following:

- *Traffic Controller*: generates the packet arrivals at an input port
- *Events Controller*: responsible for dispatching the events from the events-heap and for routing them to the appropriate modules
- *System* instantiates and integrates the aforementioned controllers

There are N traffic controllers –one for each input port of the crossbar–, one events controller and one system controller.

[10]For a hardware implementation, it is probably more functional to distinguish the crosspoint entity and build a separate controller for the crosspoints, instead of incorporating them inside the output controller [16].

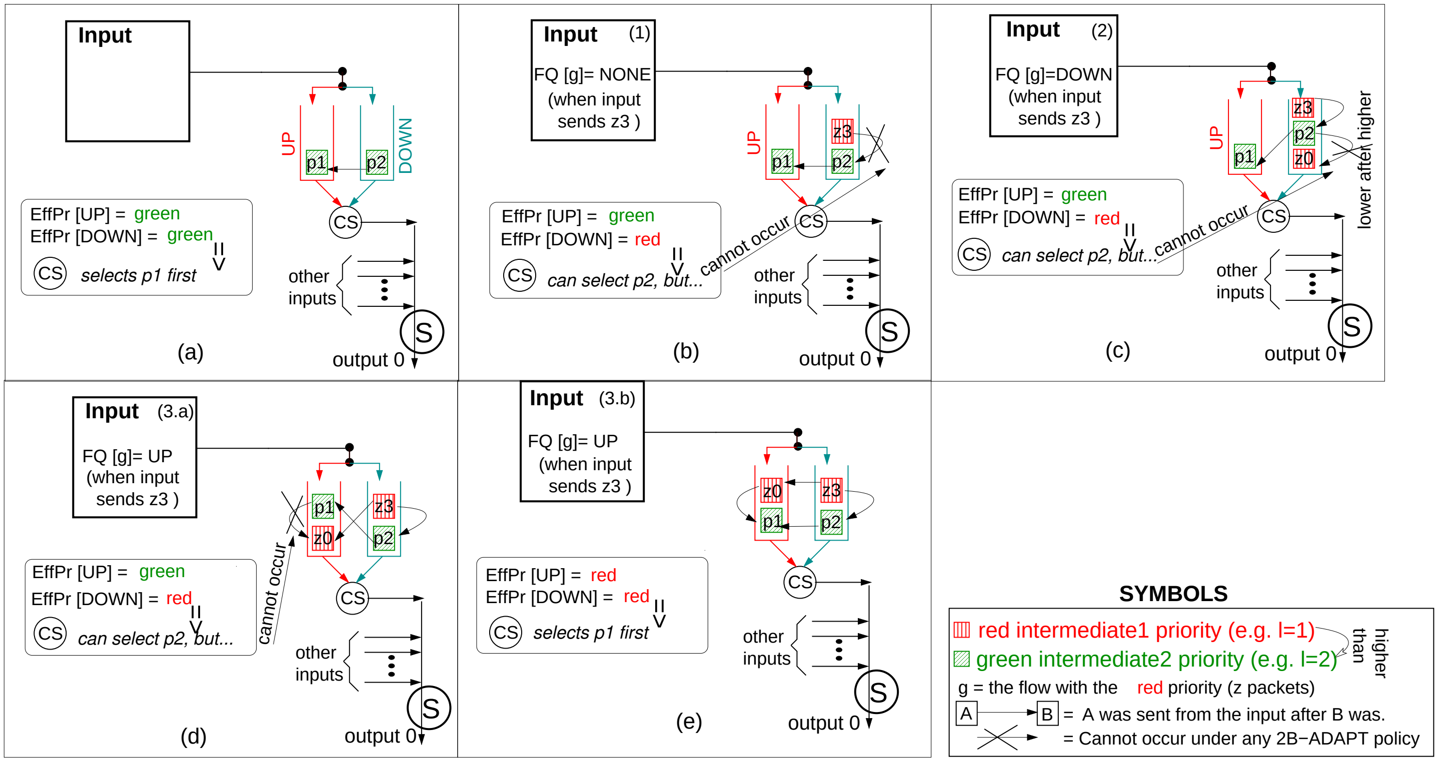

Fig. 40.

Appendix

.1. 2B-ADAPT & In Order Transmission

With adaptive assignment of packets to crosspoint queues, packets belonging to a single flow can potentially departure from the switch out of order. Alg-2B-ADAPT and all of its derivatives never sent as UP a packet of a flow that has packets pending DOWN. Thus, the only case that reordering can occur is when a packet, $p1$, is sent UP and afterward, before $p1$ is transmitted on the output lines, a packet $p2$ from the same flow is sent DOWN; this is permitted only under the 2B-ADAPT-TD policy. 2B-ADAPT-TD may sent $p2$ as DOWN when $p1$ is pending UP, only if the DOWN queue has equal or lower effective priority than the priority l, of $p1$ and $p2$. We have to prove that $p2$ cannot depart from the DOWN queue before $p1$ does, and we will do so by contradiction.

Suppose that $p2$ leaves from the DOWN queue before $p1$ leaves from the UP. Since the output scheduler selects from the UP when both queues have equal effective priority –fig. 40(a)–, the only case that this can occur is if the effective priority of the DOWN queue changed while $p2$ was still in it. This means, that a packet, $z3$, with priority l^* higher than l, was sent as DOWN from the input after $p2$ was, and before $p1$ started leaving the UP crosspoint queue. Let g denote the flow that $z3$ belongs to.

If at the time when $z3$ was sent from the input, (1) FQ $[g]$ equaled NONE, then the algorithm would try to sent $z3$ only as UP, since at this time, a lower priority packet, $p2$ was pending DOWN; so this cannot have been the case –fig. 40(b). If (2) FQ $[g]$ equaled DOWN, then, at the time when $z3$ was selected, an other packet of flow g was pending DOWN. The interesting case is if this packet $z0$, was sent DOWN before $p2$, otherwise $z0$ takes the place of $z3$ in the proof. In this case, $p2$ could not have been sent as DOWN, since packet $z0$ would still be pending in that queue, and thus the effective priority of DOWN would be higher than l –fig. 40(c).

Finally if (3) FQ $[g]$ equaled UP when $z3$ was sent DOWN, then this would mean that at that time, a packet, $z0$, belonging to flow g, was pending UP. One of the following conditions must be true; (a) $z0$ was sent UP before $p1$. In this case, our hypothesis says that $z0$ was still pending UP when the $p1$ and $p2$ reached the crosspoints; so $p1$ could not have be sent as UP at the first place, with any of the policies that we examine –fig. 40(d); (b) packet $z0$ was sent UP after $p1$; again, our inductive hypothesis tells us that $z0$ must have reached the UP queue before $p1$ started to departure from UP; but then, the effective priority of UP would have been changed into l^* before $p2$ departed from DOWN, so $p2$ could not be have been sent before $p1$, since the output scheduler would first select $p1$ from UP and afterward $p2$ from DOWN because both queues would have equal effective priority, l^* –fig. 40(e). This completes the contradiction since we found no way that $p2$ can departure from the switch before $p1$ does $\triangleleft$.