Scheduling Multicast Traffic in Internally Buffered Crossbar Switches*

Lotfi Mhamdi
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
e-mail: lotfi@cs.ust.hk

Mounir Hamdi
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
e-mail: hamdi@cs.ust.hk

Abstract —**Scheduling multicast traffic has been an active research topic due to the tremendous growth of multicast traffic (audio, video, teleconferencing, etc.) on the Internet. Considerable research work has been done on Input Queued (IQ) switches to handle the multicast traffic. Unfortunately, all the proposed solutions were of no practical value because they either lack performance or were simply not practical. Internally Buffered Crossbar (IBC) switches, on the other hand, have been considered as a robust alternative to buffer-less crossbar switches to improve the switching performance. However, no work has ever been done on multicasting in IBC switches. In this paper, we fill this gap and study, for the first time, the multicasting problem in IBC switches. In particular, we propose a simple scheduling scheme named Multicast cross-points Round Robin (MXRR) for the IBC switch architecture. Our scheme was shown to handle multicast traffic more efficiently and far better than all previous schemes. Yet, MXRR is both practical and achieves high performance.**

Index terms-- multicast, scheduling, buffered crossbar fabric

I. INTRODUCTION

The Internet's continued tremendous growth, coupled with the variety of services it is expected to provide, creates many challenges for the design and implementation of packet switches. As a result, there has been extensive research work in this area and many switching architectures have been proposed. The output-queued (OQ) switch is highly desirable for its optimal performance and QoS guarantees [3]. Unfortunately, the lack of fast enough memories along with the high internal speedup at which the OQ must operate prohibited it to scale to even a medium sized switch. Input-queued (IQ) switches, however, have gained more interest because of their low cost and high scalability [13]. While, the well known HoL blocking problem limits the achievable throughput of an IQ switch [10], the virtual output queuing (VOQ) architecture [13, 14, 15] was proposed instead and has improved the switching performance of FIFO-based IQ by several orders of magnitude, hence making the IQ switches even more attractive.

In addition to point-to-point (unicast) switching, the Internet is undergoing a remarkable transformation with increasing point-to-multipoint (multicast) traffic. This is mainly driven by the widespread of audio and video material exchanged by users on the Internet. Unlike unicast, multicasting means that an incoming cell is destined for more than one output. Traditionally, handling multicast traffic was not a simple or cost effective operation and different architectures have been proposed [10]. The fabric-based switch architecture was considered an attractive solution due to the capability it offers in transferring multicast cell at low cost. When a cell is transferred through a crossbar fabric, it can reach as many output ports as needed during the same time slot. This can be done by setting the crossbar fabric configuration (closing the corresponding cross-points) so that the cell reaches the desired outputs simultaneously.

As a result, different crossbar-based architectures have been proposed to handle multicast traffic. Most of these architectures were based on multicast FIFO queues at the ingress ports [1]. Those FIFO queues are used to accommodate arriving multicast cells before getting switched to their output ports. However, as with unicast traffic, the HoL blocking problem limits the achievable throughput and degrades the performance of the switch. One of the straightforward approaches proposed to overcome the HoL problem was the adoption of multicast VOQ architecture [9]. For an N x M switch using the multicast VOQ architecture, each input has to maintain up to 2^M-1 queue to match the number of fanout configurations. For large switches, this architecture becomes impractical and no arbitration scheme has ever been proposed. As for the multicast FIFO architecture, a plethora of algorithms has been proposed [1, 2, 4]. Unfortunately, for high-bandwidth switches, none of these algorithms has been considered as an efficient solution because they either lack performance or fairness or implementation complexity, or all of the above. This is, mostly, attributed to the centralized design of these schedulers and to the nature of the crossbar-based architecture.

In this paper, we propose a novel architecture to handle multicast traffic. Our architecture is based on multicast

* This work was supported in part by the Hong Kong Research Grant Council (Grant Number: RGC HKUST6181-01E).

FIFO queues at the ingress ports with an internally buffered crossbar (IBC) fabric. Through the rest of this paper, we will refer to this architecture as *Multicast Internally Buffered Crossbar* (MIBC) switch. In fact, the IBC switch architecture has been shown to have great potential in solving the complexity and scalability issues faced by their buffer-less predecessors [7, 8, 12]. In particular, we propose a scheduling scheme named *Multicast cross-points Round Robin* (MXRR) algorithm. We show that the MXRR scheduler achieves high throughput and outperform all previously proposed schemes. Yet, MXRR meets all the requirements for a good choice because of its simple hardware implementation and ability to achieve fairness.

The rest of the paper is organized as follows: section II contains background knowledge and introduces the multicast problem in more details. Section III introduces the MIBC architecture along with our proposed MXRR scheduling scheme. In section IV, we present a simulation study and compare the MXRR to some state-of-the art schemes. Finally, section V gives some concluding remarks.

II. BACKGROUND KNOWLEDGE

A. Architectural Model

A multicast switching architecture customarily consists of N input ports and M output ports. Each input port contains a multicast FIFO queue that accommodates cells coming to that input and destined to any of the output (or set of output) ports. It is assumed that each arriving cell contains a vector that indicates the set of output ports to which this cell has to be switched to. For an N x M switch, the destination vector of a multicast packet can be any of 2^M-1 possible vectors. This vector is known as the fanout of the input cell. Traditionally, researchers have tackled the multicasting problem by mapping it to unicast scheduling. The idea was to replicate every multicast input packet over multiple packet times, generating one output packet per packet time. Then each generated packet is equivalent to a unicast packet and any unicast arbitration method can be applied. However, this approach has two disadvantages [1]. First, each input must be copied multiple times, increasing the required memory bandwidth. Second, the input packets contend for access to the switch multiple times, reducing the bandwidth available to other traffic at the same input.

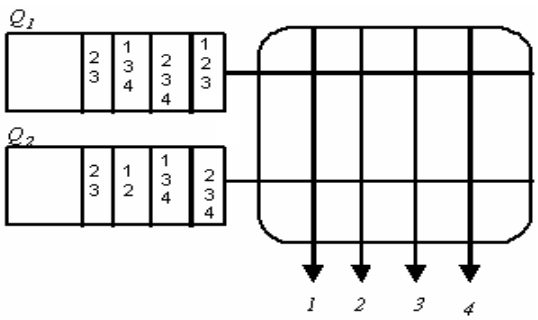

Fig. 1: a *2* x *4* multicast crossbar switch

Depending on the output availability, a cell may or may not reach all its outputs indicated by its fanout vector. With this situation occurring, two different service disciplines can be used for multicast packets [5]. The first is known as *no fanout-splitting*, in which all of the packet copies must be sent in the same packet time. If any of the output packets loses contention for an output port, none of the output packets are transmitted and the packet must try again in the next time slot. If we consider the no fanout-splitting discipline, only one cell from Fig. 1 can be transferred. This is because both HoL cells in Q_1 and Q_2 have packets destined to outputs 2 and 3 and at most one cell could be sent to an output per time slot. In contrast, the second discipline is known as *fanout-splitting*, in which output packets may be delivered to output ports over any number of packet times. Only those output packets that are unsuccessful in one packet time continue to contend for output ports in the next time slot. Fanout-splitting provides a higher switch throughput [6] for little increase in implementation complexity. Higher throughput is achieved because fanout-splitting is work conserving. In this paper, we only consider fanout-splitting. If we consider again the example in Fig. 1 and the fanout-splitting discipline, since Q_1 and Q_2 both have cells destined to output ports 2 and 3, we know that at the end of the scheduling phase we will still find cells destined to those outputs. Then, depending on the policy used, these two cells can be both queued (*concentrated*) at either the HoL of Q_1 or Q_2. Or they can be *distributed* over the two input queues, one cell in each queue. For the sake of clarity, we introduce some terminology, similar to that of [1], which is used throughout the rest of this paper:

- Residue: the residue is defined as the number of cells left at the HoL of the input queues after loosing contention for the outputs at the end of each time slot. In the case of Fig.1, the residue is equal to {2,3}.
- Concentrating policy: a multicast scheduling policy is said to be concentrating if, at the end of each time slot, the residue is left on the smallest number of input ports. If we refer to the example of Fig. 1, such policy will leave the residue, {2,3}, either on the input port number 1 or on the input port number 2, but not on both.
- Distributing policy: a multicast scheduling policy is said to be distributing if, at the end of each time slot, the residue is left on the largest number of input ports. If we refer to the example of Fig. 1, this policy will leave the residue, {2,3}, on both input ports 1 and 2, and not on one port only.

Recently, several scheduling algorithms have been proposed for the multicast FIFO crossbar switching architecture. The following section describes some of these algorithms.

B. Existing Scheduling Algorithms

As with the unicast scheduling, multicast scheduling has been of high interest in the research community. As a result, many algorithms have been proposed. These schemes can

be divided into two classes: weighted and non weighted algorithms. An algorithm named Concentrate was presented in [1]. The Concentrate algorithm always concentrates the residue on the smallest number of input ports. In the case of ties, Concentrate chooses the input having the HoL cell with the shortest waiting time. This algorithm was used as a basis to compare the performance of other buffer-less algorithms, since it achieves the highest throughput for the multicast FIFO crossbar switching architecture. Unfortunately, the Concentrate algorithm is not considered practical, since it requires up to N iterations per time slot to complete. This makes it very hard to run at high-speed or on a large sized switch. Moreover, Concentrate is not considered a fair scheme, since it leads to the permanent starvation of the input queues. Because the performance of Concentrate can be used as upper bound on throughput for comparing other algorithms, an algorithm named TATRA was proposed in [1] and aimed to approximate the throughput of the Concentrate policy. The TATRA scheduling algorithm was motivated by Tetris, the popular block-packing game. TATRA has the properties of guaranteeing at least one input packet is discharged each packet time, and also residue concentration. Unfortunately, TATRA has two major disadvantages: first, it is difficult to implement since the process cannot be parallelized. It is simpler than the Concentrate algorithm, but still requires N iterations per time slot. Second, treating all inputs uniformly is of no benefit when the inputs are non-uniformly loaded or when some inputs request a higher priority.

A scheme named the *Multicast Round-Robin Matching* (mRRM) was proposed in [2]. Designed to be simple to implement in hardware, mRRM tends to concentrate the selection onto a small number of inputs, while maintaining fairness. However, unlike Concentrate, mRRM does not perform well and cannot achieve high throughput and therefore was not considered practical for high-speed networks. Similar to mRRM, a scheme named the *Centralized Multicast Contention Resolution* (CMCR) was proposed in [4]. The only difference is that, instead of maintaining a single output pointer, each output maintains an independent pointer. This slight difference makes CMCR unable to guarantee to completely serve a packet in one time slot. Also, CMCR is a non-concentrating scheme, which makes it perform worse than mRRM even though the time complexity of the two is identical.

As a summary, we argue that each of the above presented schemes tries to address some issues but fails to meet other vital requirements. So far, none of these algorithms proved simultaneously efficient in terms of high throughput, practical in terms of implementation complexity or fair with respect to the input FIFO queues. In the following section, we propose our new architecture along with a scheduling scheme that meets all these requirements.

III. The Multicast Internally Buffered Crossbar Switch architecture (MIBC)

A. Motivation

Our choice of the multicast internally buffered crossbar switch architecture is motivated by the fact this architecture has key advantages that can serve to ensure that the scheduling algorithm can be simple and efficient at the same time. The presence of internal buffers drastically improves the overall performance of the switch due to the advantages it offers. First, the adoption of internal buffers makes the scheduling totally distributed, hence reducing dramatically the arbitration complexity and making it linear. Second, and most importantly, these internal buffers reduce (or avoid) the output contention. Meaning, they allow the inputs to send cells to an output irrespective of simultaneous cells transfer to the same output. If an output is not ready to receive a cell from an input, the input still can send it to the internal buffer provided that this internal buffer has enough room for that cell.

B. Switch Model

We consider the switch model defined in Fig 2. Fixed size packets, or cells, are considered. Upon their arrival to the switch, variable length packets are segmented into cells for internal processing and re-assembled before they leave the switch. A processing cycle has a fixed length, called a cell or time slot. There are N input cards, each one contains a FIFO multicast queue. The internal fabric consists of NM buffered crosspoints (XP). When an arriving cell, to an input port i, $\forall\, 1 \le i \le N$, has its fanout vector containing the output j, $\forall\, 1 \le j \le M$, it must go through the crosspoint $XP_{i,j}$, before continuing its journey to the output buffer.

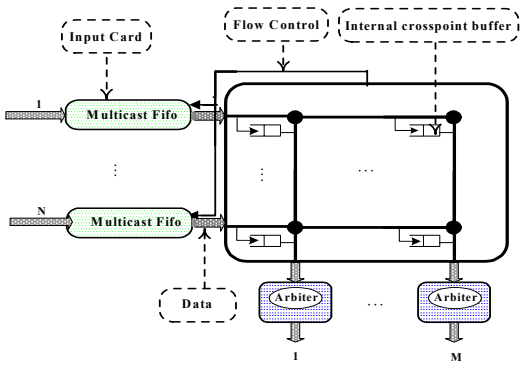

Fig. 2: The N x M multicast internally buffered crossbar (MIBC) switch architecture

As with unicast scheduling, a multicast scheduling cycle consists of the following three steps: input scheduling, output scheduling and delivery notifying. During the input scheduling phase, each input, i, selects, in an independent and parallel way, the HoL cell of its multicast FIFO queue and sends it to the internal buffer corresponding to its fanout set. Likewise, each output, j, selects, independently and in parallel, a non empty crosspoint buffer, $XP_{i,j}$, and sends its cell to the output queue. Then, the delivery notifying is performed to carry the flow control between the internal buffers and the input queues. For each

delivered cell, the flow control mechanism "informs" the corresponding input of the internal buffer status.

C. *The Multicast Cross-point Round Robin (MXRR) Algorithm*

The description of each scheduling phase of the Multicast cross-point Round Robin algorithm is as follows:

- *Input scheduling*

  Each input sends its HoL multicast cell to the set of internal buffers corresponding to its fanout vector. If one or more internal buffers are not free, the cell stays at the HoL of that input and waits for the next input scheduling phase to send to its remaining internal buffers.

- *Output scheduling*

  All the output pointers are, artificially, set to the same initial position and incremented, each time slot, by one mod (*N*).
  Starting from its pointer's index, each output selects the first non empty cross point buffer and sends its queued cell to the output buffer.

D. *Example*

Consider the following *2* x *4 MIBC* switch. Let us assume that the output pointers are all pointing to input 1 and all the internal buffers are empty.

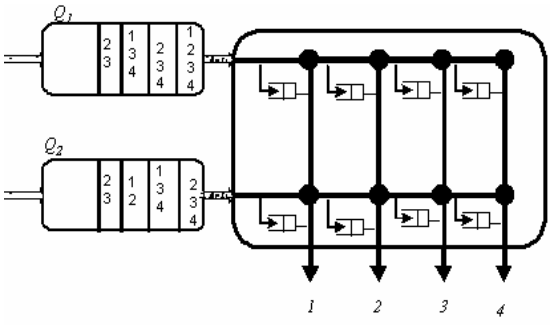

Fig. 3: *2 x 4* MIBC switch

During the input scheduling phase, both HoL cells of Q_1 and Q_2 will be completely transferred to the internal buffers. During the output scheduling phase, since the output pointers have index 1 each, therefore every output *j*, will select the internal buffer $XP_{1,j}$, $\forall\, 1 \le j \le 4$. This means that, during this time slot, the HoL cell of Q_1 is completely served. At the beginning of the second time slot, $XP_{2,2}$, $XP_{2,3}$, and $XP_{2,4}$ are occupied, therefore the second cell of Q_2 (which becomes the HoL cell) cannot send its cell to all the outputs {1,3,4}. It only can send to $XP_{2,1}$ which leaves a residue of {3,4}. Q_1, however, can send its HoL cell completely to the internal buffers. During the output scheduling phase, since the output pointer's indexes are incremented to 2, therefore each output *j*, will select the internal buffer $XP_{2,j}$, $\forall\, 1 \le j \le 4$. This means that, during this time slot, the HoL cell of Q_2 is completely served and part of the second cell as well. From this example, we draw the following properties and advantages of the MXRR scheduling scheme:

- The MXRR scheme guarantees the total service of at least one packet each time due to the output pointers setting (which point to the same internal buffer and advance synchronously). Moreover, the time a packet waits at the HoL is bounded by number of input ports, *N*.
- Fair and starvation free: Since the output pointers move artificially and in a synchronous fashion irrespective of the chosen packet, the starvation problem will never occur. The chance of service for any two cells from two different input ports is exactly the same due to the round robin pointer movement.
- Simple in hardware implementation: Each input does FIFO arbitration. The outputs, on the other hand, work in a totally distributed and parallel manner. No computation and comparison of weights is needed to make an arbitration decision. Each output arbiter just performs simple round-robin arbitration.
- Enhanced performance: Achieves high throughput and has lower packet latency than all previously proposed buffer less algorithms. We will examine this property in the following section, which contains the simulation results.

IV. PERFORMANCE STUDY

The simulation results are gathered from an 8x8 and a 16x16 switch respectively. Each point in the figures below runs for 1,000,000 time slots, and the statistics are gathered from the $100{,}000^{th}$ time slot. We evaluated the different algorithms under two uniform traffic models: Bernoulli and bursty. The average burst size we use is B =16. For both traffic models, the multicast vectors are uniformly distributed over all possible multicast vectors. Since the average fanout is (8+1)/2=4.5 for the 8x8 switch, and (16+1)/2 = 8.5 for the 16x16 switch, and each output has the same chance of being a destination, the average output load is 4.5 times as much as the input load in the case of the 8x8 switch and 8.5 in the case of 16x16 switch.

For comparison, we show the performance of Concentrate, TATRA and our MXRR algorithm. We chose Concentrate because it provides the highest throughput for the single FIFO structure but is not considered a practical algorithm. The choice of TATRA was because it is considered as one of the practical algorithms that achieve high performance.

Fig. 4 shows the average cell latency of an 8x8 FIFO multicast switch employing Concentrate, TATRA and MXRR algorithm. MXRR has a smaller average delay than both other schemes when the traffic is Bernoulli (B = 1). This result is achieved even when the traffic is bursty (B = 16) and with MXRR using a simpler scheduling scheme (i.e., no iterations or weight are used).

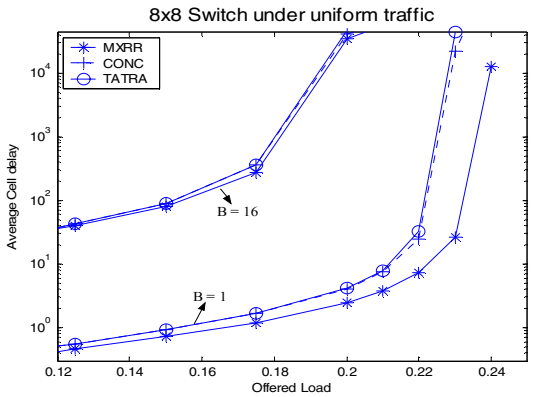

Fig. 4: Delay performance of an 8x8 switch under uniform arrivals

As for the 16x16 switch's performance, Fig. 5, the MXRR shows the best performance again. TATRA has a worse average delay than Concentrate.

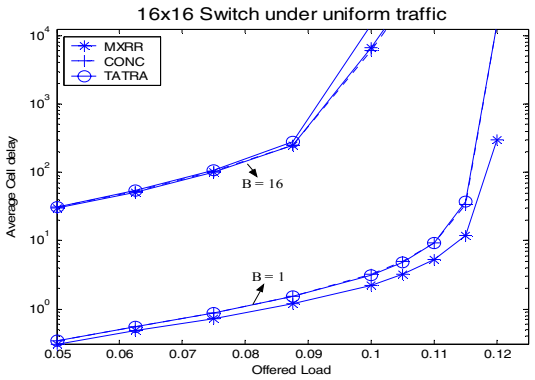

Fig. 5: Delay performance of a 16x16 switch under uniform arrivals

It is of note that the performance of the MXRR scheme is expected to improve as the internal buffer size increases. To this end, we studied the effect of the internal buffer size on the performance of MXRR. Fig. 6 depicts the average delay performance of MXRR scheme under Bernoulli I.I.D uniform arrivals and different internal buffers sizes of 2, 4 and 8 cells respectively.

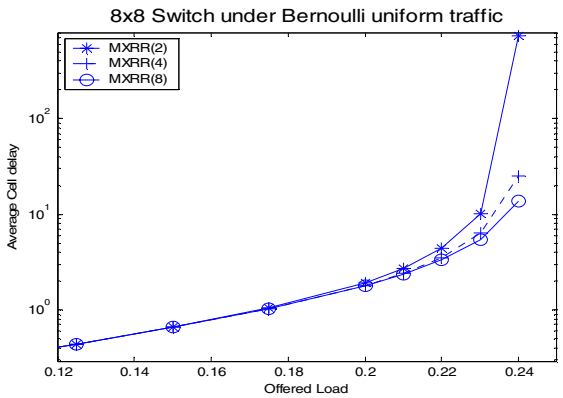

Fig. 6: Average delay performance of MXRR scheme under Bernoulli I.I.D uniform arrivals and different switch sizes

We can see that between a size of 2-cell internally buffered crossbar switch and 4, there is a big improvement in performance. However, the performance becomes almost steady for internal buffers sizes beyond 4 cells.

V. CONCLUSION

This paper proposes a new architecture to address the multicast traffic handling problem. Our new architecture, based on the internally buffered crossbar fabric, along with a simple scheduling scheme was shown to exhibit much better performance than the buffer less architecture. In particular, our proposed MXRR scheme was shown to outperform all state-of-the-art algorithms. Yet, it is simple to implement in hardware and able to run at very high speed. We expect that, with more carefully designed scheduling schemes for the MIBC switching architecture, much higher performance can be achieved.

REFERENCES

[1] B. Prabhakar, N. McKeown, and R. Ahuja, "Multicast Scheduling for Input-Queued Switches," *IEEE JSAC*, June 1997.

[2] B. Prabhakar and N. McKeown, "Designing a Multicast Switch Scheduler", *Proceedings of the 33rd Annual Allerton Conference on Communication, Control, and Computing*, pp. 984-993. Montipacketo, Illinois, October 1995.

[3] Chuang, S.-T.; Goel A.; McKeown, N.; Prabhakar, B.; "Matching output queueing with a combined input output queued switch," *Computer Systems Technical Report CSL-TR-98-758*, March 1998.

[4] H. J. Chao and J.-S. Park, "Centralized Contention Resolution Schemes for A Large-Capacity Optical ATM Switch", *Proc. IEEE ATM Workshop*, Fairfax, VA, May 1998.

[5] J. F. Hayes, R. Breault, and M. Mehmet-Ali, "Performance Analysis of a Multicast Switch", *IEEE Trans. Commun.*, vol. 39, No. 4, pp. 581-587, April, 1991.

[6] J. Y. Hui, and T. Renner, "Queuing Analysis for Multicast Packet Switching," *IEEE Trans. Commun.*, vol. 42, no. 2/3/4, pp.723-731, Feb. 1994.

[7] L. Mhamdi and M. Hamdi, "MCBF: A High-Performance Scheduling Algorithm for Buffered Crossbar Switches," *IEEE Communications Letters*, Vol. 7, No. 9, Sept. 2003.

[8] L. Mhamdi and M. Hamdi, "Practical Scheduling Algorithms for High-Performance Packet Switches", *IEEE ICC'03*, Vol. 3, pp. 1659-1663. Alaska, May 2003.

[9] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Optimal Multicast Scheduling in Input-Queued Switches," *Proc. ICC 2001*, pp. 2021-2027.

[10] M. Guo, R. Chang, "Multicast ATM switches: survey and performance evaluation", *Comp. Commun. Rev.*, vol. 28, pp. 198-131, Apr. 1998.

[11] M. Karol, M. Hluchyj, and S. Morgan, "Input Versus Output Queuing on a Space-Division Packet Switch," *IEEE Trans. on Commun.*, Vol. COM-35, Dec. 1987, pp.1337 – 1356

[12] M. Nabeshima, "Performance Evaluation of Combined Input-and Crosspoint-Queued Switch," *IEICE Trans. Commun.*, Vol. E83-B, No.3 March 2000.

[13] N. McKeown, "Scheduling algorithms for input-queued cell switches," Ph.D. Thesis, University of California at Berkeley, 1995.

[14] T. Anderson, Owicki, S. Saxe, J. Thacker, C. "High speed switch scheduling for local area networks," *ACM Transaction. on Computer Systems*, Nov.1993, pp. 319-352.

[15] Tamir, Y.; Chi, H.C.; "Symmetric crossbar arbiters for VLSI communication switches" *IEEE Transactions on Parallel and Distributed Systems*, Jan 1993. Vol.4, no.1, pp. 13-27.