Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

SYMMETRIC CROSSBAR ARBITERS FOR VLSI COMMUNICATION SWITCHES

Yuval Tamir
Hsin-Chou Chi

October 1990
CSD-900028

Symmetric Crossbar Arbiters for VLSI Communication Switches †

Yuval Tamir and Hsin-Chou Chi
Computer Science Department
4731 Boelter Hall
University of California
Los Angeles, California 90024-1596
U.S.A.
Phone: (213)825-4033 E-mail: tamir@cs.ucla.edu

Abstract

Small crossbars are key components of communication switches used in multicomputer interconnection networks. The traffic through the switches is often delayed due to conflicting demands for resources, such as buffer space or output ports. Achieving the maximum possible performance of the communication network is dependent on *efficient, fair* scheduling of the available resources within each switch. Hence, switches must include *arbiters* that resolve conflicting resource demands. Efficient design and implementation of these arbiters is critical for maximizing network performance.

In order to maximize performance, recent communication switch designs allow packets at an input port, destined to different output ports, to be transmitted through the switch in any order. Each input port contends for *multiple* output ports but needs only one for full utilization. Similarly, each output port contends for multiple input ports and needs one for full utilization. The arbitration task is thus *symmetrical* with respect to inputs and outputs.

This paper focuses on the design and implementation of *symmetric crossbar arbiters*. Several arbiter designs are compared based on simulations of a multistage interconnection network. These simulations demonstrate the influence of the switch arbitration policy on network throughput, average latency, and worst-case latency. It is shown that some ''natural'' designs result in poor system performance and/or slow implementations. Two efficient arbiter implementations are proposed. Based on network simulations, VLSI implementation, and circuit simulation, it is shown that these arbiters achieve nearly optimal system performance without becoming the critical path that limits the system clock.

Index Terms: Arbiters, communication coprocessors, communication switches, crossbars, interconnection networks, multiprocessors, multicomputers, VLSI systems.

† This research is supported by the SDIO Innovative Science and Technology Office, managed by the Office of Naval Research, contracted to the Jet Propulsion Laboratory under task plan #80-2984.

October 1990

I. Introduction

The ability of multiprocessors and multicomputers to achieve high performance is dependent on interconnection networks that provide high-bandwidth low-latency interprocessor communication. The key components of interconnection networks for large multiprocessors and multicomputers are small $n \times n$ switches [8, 2, 18, 20, 4, 25, 3]. An efficient internal micro architecture for these switches is essential for achieving high-performance communication with cost-effective implementations.

The function of a communication switch is to receive packets arriving at its input ports and route them to the appropriate output ports. The bandwidth of the input ports is typically equal to the bandwidth of the output ports. If two packets destined to the same output port arrive at the input ports of the switch simultaneously, they cannot both be forwarded. In an *unbuffered* network, one of the two packets must be discarded or misrouted. In a *buffered* network, where buffer storage is associated with each switch, one of the packets is stored in the switch until its destination output port is free [6]. By adding buffer storage to each switch, the maximum throughput of the network is increased [6].

A typical communication switch consists of input ports, output ports, an $n \times n$ crossbar, and some buffer memory [6, 11, 19]. Depending on the traffic patterns, there may be conflicting demands for these resources. *Arbiters* that resolve the conflicts and provide efficient and fair scheduling of these resources are critical for achieving the maximum possible performance from a given network. This paper focuses on the design and VLSI implementation of these arbiters.

In communication switches where packets are processed at each input in first-in-first-out (FIFO) order, arbitration of the internal crossbar is relatively simple. Following arbitration, each input port may be either active (transmitting) or idle (blocked). For each output port, there is a choice of which of several contending input ports will connect to it. Hence, for each output port, there is a simple independent arbiter that selects one of the inputs requesting that output and blocks the others [3, 1].

Recent communication switch designs maximize performance by allowing packets at an input port to be processed in *non-FIFO* order [15, 11, 22, 5]. Specifically, packets at an input port, destined to different output ports, may be transmitted through the switch in any order. Each input port contends for *multiple* output ports but needs only one for full utilization. Similarly, each output port contends for multiple input ports and needs one for full utilization. The arbitration task is thus *symmetrical* with respect to inputs and outputs. Furthermore, since the arbitration result for each port is dependent on the arbitration for other ports, the arbitration task is more complex than for switches with FIFO input ports and the task cannot be performed by separate independent arbiters at the input ports or output ports.

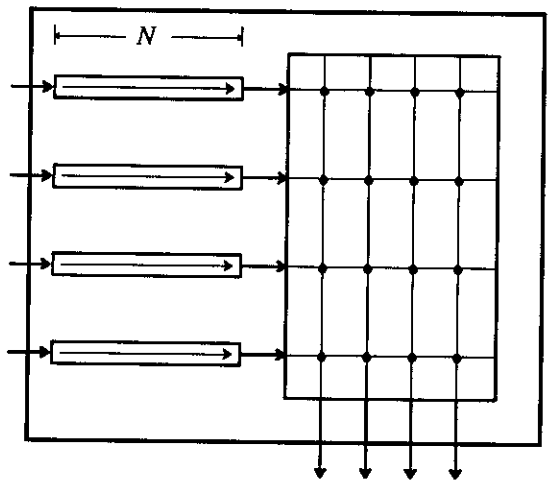
In order to realize the potential for high performance of non-FIFO processing of packets, the crossbar arbitration should result in switch configurations that allow for the maximum number of packets to be transmitted simultaneously. The arbitration process itself must be fast relative to the rate at which packets are received and relative to the latency of packet transmission through the switch. This paper discusses the design and VLSI implementation of *symmetric crossbar arbiters* that meet these requirements. This work is part of the UCLA ComCoBB (Communication Coprocessor Building-Block) project, whose focus is the design and implementation of a high-performance communication coprocessor for VLSI multicomputers. The investigation of symmetric crossbar arbiters was motivated by the fact that our design of a communication coprocessor involves the use of non-FIFO input buffers [22].

In the next section, we discuss the crossbar arbitration problem and summarize previous work on arbiter design. Symmetric crossbar arbitration is defined and its use in modern communication switches is explained. Section III presents several possible symmetric crossbar arbitration schemes. Basic implementation considerations are discussed, and the operation of three practical symmetric crossbar arbiter designs is described. The performance advantage of using an efficient arbitration policy is demonstrated using simple probabilistic analysis of 2×2 switches. The performance of several symmetric crossbar arbitration schemes is evaluated in Section IV. The evaluation is based on event-driven simulations of individual switches of several sizes, and event-driven simulations of multi-stage interconnection networks. The VLSI implementation of the best practical arbitration schemes is described in Section V. Circuit simulation is used to determine the performance of the proposed arbiter. The abbreviations used to denote the various arbitration schemes discussed are summarized in the Appendix.

II. Crossbar Arbitration

While many different internal organizations of buffered communication switches are possible, critical implementation considerations lead to the use of buffers at the input ports rather than central buffer pools or buffers at the output ports [22]. In Figure 1 we show two of the possible switch organizations with input port buffers. A key component in these switches is a crossbar, which allows arbitrary permutations in connecting the input buffers to output ports. The crossbar consists of n horizontal buses (rows) and n vertical buses (columns). Each horizontal bus is connected to an input buffer, while each vertical bus is connected to an output port. A horizontal bus intersects a vertical bus at a *crosspoint*. At each crosspoint there is a switch, which may be closed to form a connection between the corresponding input buffer and output port.

With FIFO buffers (Figure 1-a), there is at most one packet at each buffer that is ready for transmission through the crossbar. Based on the destination of the packet at the head of its FIFO queue,



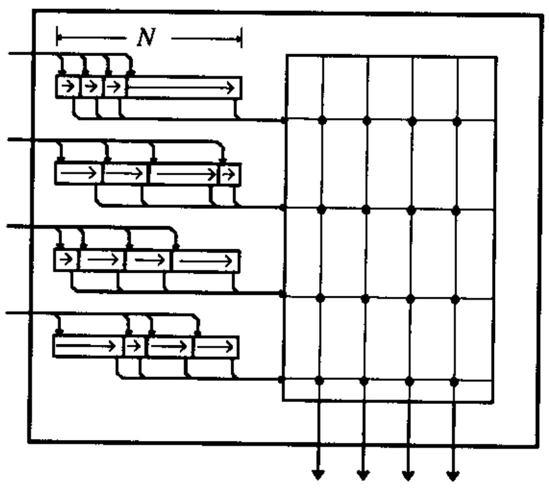


Figure 1: Switches with FIFO buffers and DAMQ buffers

each non-empty input buffer contends for the internal bus connected to one of the output ports. Access to each internal output port bus can thus be arbitrated independently of arbitration of access to other internal buses. Hence, a crossbar arbiter for an $n \times n$ FIFO switch (i.e., a switch with FIFO input port buffers) can be constructed out of n independent conventional bus arbiters [1].

A conventional bus arbiter receives up to n requests for use of the bus and grants one of these requests. For equal priority requests, the goal of a bus arbitration scheme is to provide *fair* access to the different ''users'' and to lend itself to reliable, low cost, fast implementation. The design of such n-user 1-server (or 1-of-n) arbiters has been investigated by many researchers and system developers, and is well understood [9, 23, 24]. Crossbar arbitration with n 1-of-n arbiters is used in the Torus Routing Chip [3], where there is a fixed priority order of the inputs to each of the arbiters. Bhuyan [1] describes crossbar arbitration where there is an attempt to maintain fairness by ''rotating'' the priority order in each 1-of-n arbiter such that the top priority is given to the user following the one who was last serviced.

In order to increase the bandwidth between processors and memory modules, systems with multiple buses have been proposed [12, 17]. In these systems n processors are connected to m memory modules through b buses. In order to complete, a request from a processor must win the arbitration for two types of resources: a memory module and a bus. Since each processor requests one particular memory module and a memory module can handle only one request at a time, this arbitration can be performed by m independent conventional 1-of-n arbiters [13, 1]. All the memory modules for which there are requests must then contend for the available b buses. Since a memory module can use any of the b buses, a different type of arbiter must be used. Lang and Valero [13] describe the design and implementation of such an m-user b-server (b-of-m) arbiter.

In the Concert multiprocessor system [10], a segmented ring bus is used to interconnect processor clusters. Each one of the n processor clusters (users) can request up to $n/2$ bus segments (servers) in order to complete an access. Each cluster request is translated into a request for a particular set of bus segments. If any one of the requested bus segments is busy, the cluster request cannot be serviced. If a cluster request is denied because one of its required bus segments is busy, any other bus segments already "given" to the cluster become free for use by another cluster. Hence, for maximum utilization of the ring bus, the arbitration for each bus segment cannot be performed independently of the arbitration of other bus segments. Thus, conventional 1-of-n arbiters cannot be used. The required arbitration is also different from the b-of-m arbitration used for multi bus systems since the resources (bus segments) are not interchangeable and a request may require more than one resource.

An $n \times n$ communication switch with FIFO buffers (Figure 1-a) does not utilize the resources of the switch efficiently. Specifically, the problem with such a switch is that packets may be blocked unnecessarily — if the packet at the head of the queue is blocked, all other packets in the buffer, even those destined to idle output ports, are also blocked [22]. In order to improve output port utilization, and thus increase the throughput of the switch, packets can be segregated according to the output port to which they have been routed. This can be done using separate FIFO buffers for each of the output ports at each of the input ports [15,22]. In order to better utilize the available buffer space, the input buffers can be *dynamically* partitioned between the different queues. Such a *dynamically-allocated, multi-queue* (DAMQ) buffer has been designed at UCLA (Figure 1-b), and has been shown to significantly increase network throughput [22,7]. In a lightly different context, Dally [5] has shown that multi-queue input buffers are also useful when there is a dedicated queue for each virtual channel instead of each output port.

An $n \times n$ crossbar with its associated arbiter are shown in Figure 2. The inputs to the arbiter consist of n^2 *request* lines, one per crosspoint, and n *output port blocked* (*OPB*) lines, one per output port. A *request* line is asserted when the use ("service") of the particular crosspoint is needed. In order to provide flow control, it is sometimes necessary to block a switch from forwarding packets to one of its outputs. An *OPB* line is asserted in order to prevent use of a particular output port by inhibiting granting of any crosspoint in the corresponding column. In the rest of this paper we consider a crosspoint to be requested only if the *request* line is asserted while the *block* line is negated. The outputs from the arbiter consist of n^2 *grant* lines and n^2 *control* lines. The *grant* lines indicate which crosspoint requests have been granted, while the *control* lines connect or disconnect the individual crosspoints in the crossbar.

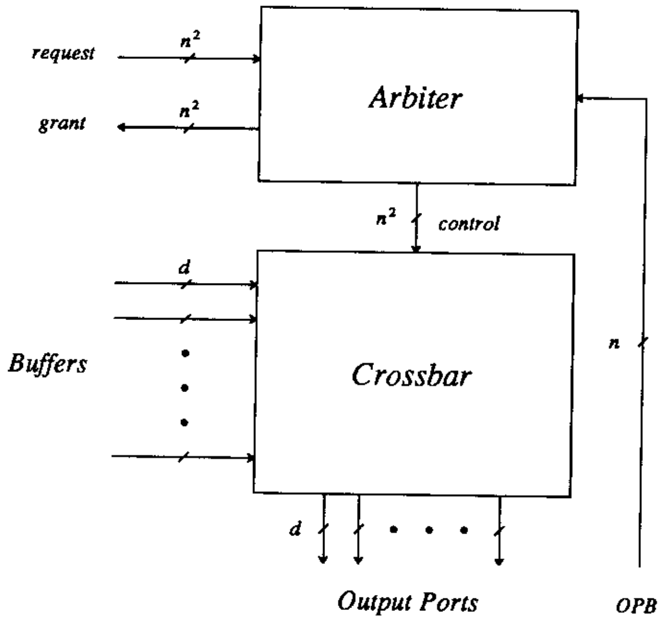


Figure 2: A crossbar arbiter in context. The crossbar connects n d-bit buses from the input buffers to the n output ports. Crosspoints (row/column connections) are requested using the n^2 *request* lines, and granted using the *grant* lines. The OPB (output port blocked) lines indicate which output ports are blocked and should not participate in the arbitration.

At any point in time, a multi-queue input buffer can transmit through the crossbar the packet at the head of *any* of its queues. If any one of these packets is transmitted, the input buffer output bandwidth is fully utilized. We consider each *queue* to be a "user" and each internal crossbar bus to be a "server" (resource). Hence, the switch consists of n^2 users and $2n$ resources. A request from a queue can be granted only when the required input bus *and* the required output bus are available. Hence, as with the Concert segmented ring bus, each user requests more than one resource. Furthermore, as with the segmented bus, arbitration for the different resources cannot be performed independently. In order to maximize resource utilization and performance, the result of the arbitration of one resource (e.g. an input bus) may have to be modified if the winner loses the arbitration for another resource (an output bus). Unlike the segmented bus arbitration problem, if a user has a request, it is *always* for the same two resources. Hence, only n out of the n^2 users may contend for any resource. Furthermore, *at most n* out of the n^2 users may win each arbitration.

A user-resource model of a 2×2 crossbar with multi-queue input buffers is shown in Figure 3. For any crossbar, the arbitration problem can be described as a matrix of requests, each one for a crosspoint of the crossbar. The goal is to arbitrate among the requests so that at most one grant is given to a row of the matrix, and at most one grant is given to a column of the matrix. With conventional FIFO buffers, there

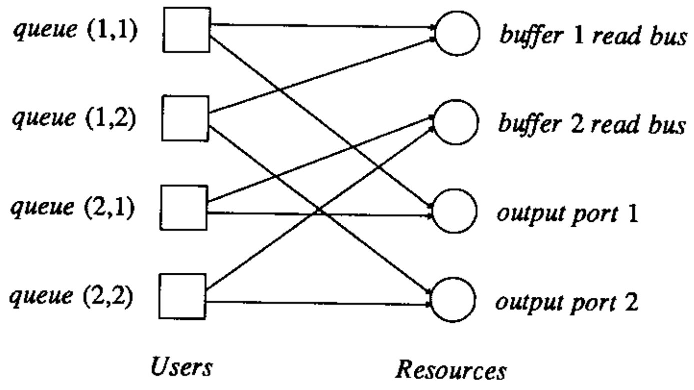


Figure 3: A user-resource model of symmetric arbitration of a 2×2 crossbar.

are never conflicting requests for crosspoints on the same row. With multi-queue input buffers, conflicting requests on the same row can occur. Due to this symmetry between rows (input buses and input ports) and columns (output buses and output ports), the function of the required arbiter is called *symmetric crossbar arbitration.* Symmetric crossbar arbitration is a generalization of the conventional crossbar arbitration, since the latter is a special case of the former.

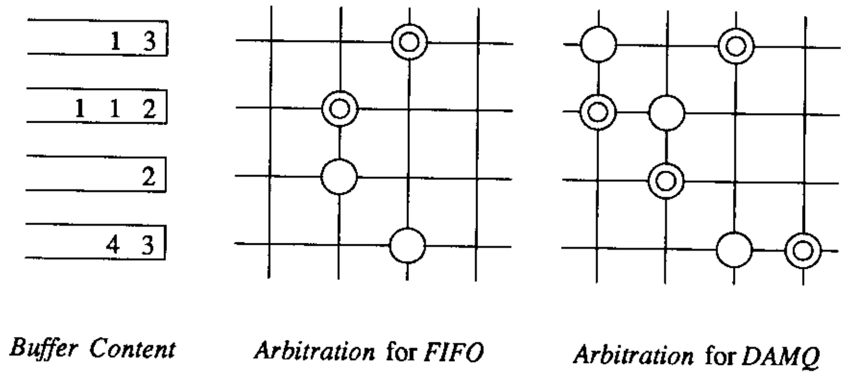


Figure 4: Example arbitrations for switches with FIFO and DAMQ buffers. Double circles indicate granted requests. Single circles indicate denied requests.

Since DAMQ buffers (and other multi-queue buffers) allow more than one request to the crossbar arbiter, there is the opportunity to connect more crosspoints of the crossbar than with FIFO buffers, thus leading to higher throughput and lower latency. Figure 4 shows an example of buffer contents and how requests can be arbitrated for FIFO and DAMQ buffers. The numbers in the buffers represent the destination output ports of the packets. The crosspoints with single circles indicate denied requests, while those with double circles indicate granted requests.

III. Symmetric Crossbar Arbiters

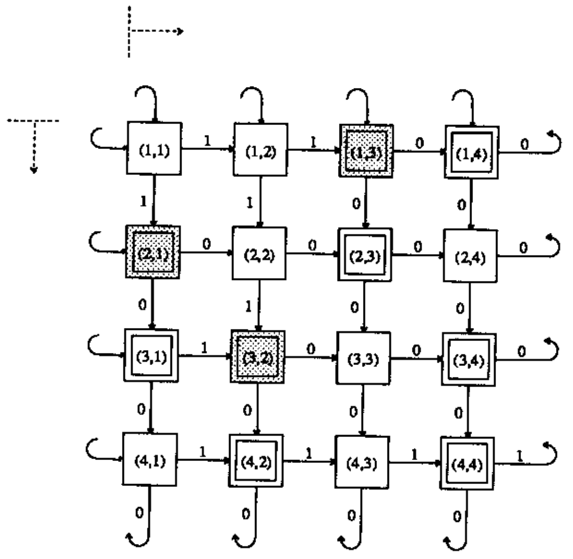
With multi-queue input buffers there is the potential for achieving significantly higher network throughput than with conventional FIFO buffers [22]. In order to realize this potential, it is necessary to use symmetric crossbar arbiters which, on the average, connect more crosspoints following every arbitration cycle than conventional crossbar arbiters. Since the arbitration task is more complex than with FIFO buffers, there is a question of whether "good" arbiters will be too slow and/or use excessive chip area. In this section we demonstrate the importance of the arbitration scheme for achieving high performance and discuss several possible schemes. We focus on three schemes that appear particularly promising in terms of the potential for high performance and practical implementation. More extensive discussion of the performance and implementation of the proposed schemes are found in sections IV and V, respectively.

A. The Importance of the Arbitration Scheme

There may be a question of whether there is a significant performance difference between "good" and "poor" symmetric crossbar arbitration schemes. In this subsection we demonstrate that there is such a difference by using static probabilistic analysis to compare two arbitration schemes for a 2×2 switch: a practical scheme that follows naturally from previous work on arbiter design (Section II), and a theoretical scheme which is not practical but can achieve nearly optimal performance. In order to calibrate the performance of the proposed schemes, they are compared to conventional crossbar arbitration for a switch with FIFO buffers.

As described earlier, with FIFO arbitration (FIFOA), there is at most one request for each row. Multiple requests for each column are arbitrated independently, using round-robin arbiters [1, 24]. For a switch with multi-queue buffers, efficient symmetric crossbar arbitration maximizes the number of crosspoints utilized. We consider the performance of a *static optimal arbiter* (SOA), which examines all the requests and searches for the arbitration result that maximizes the throughput for the next cycle. If there are several configurations with equal throughput, one is selected randomly. The SOA uses exhaustive search of all legal configurations and is clearly not a practical arbiter design. We use it for comparison with other, more practical, arbiter designs.

A straightforward extension of FIFO arbitration to symmetric crossbar arbitration is to decompose the arbitration process into two steps: arbitration among conflicting column requests followed by arbitration among conflicting row requests that have won the first step. Such a *two step arbiter* (TSA) is amenable to VLSI implementation as an $n \times n$ array of arbitration cells, one cell per crosspoint (Figure 5). For each crosspoint (i,j), in addition to the *request* $(R_{i,j})$ input and *grant* $(G_{i,j})$ output, each cell has two inputs, *north* ($N_{i,j}$) and *west* ($W_{i,j}$), and two outputs, *south* ($S_{i,j}$) and *east* ($E_{i,j}$). Note that $N_{i,j} = S_{i-1,j}$ and $W_{i,j} = E_{i,j-1}$. The $N_{i,j}$ signal indicates that the rows above did not request column j. The $W_{i,j}$ signal indicates that there are no granted requests for the crosspoints to the left. The G output is asserted if, and only if, the crosspoint is requested and both the N and the W inputs are asserted. Thus, $G_{i,j} = R_{i,j} \wedge N_{i,j} \wedge W_{i,j}$, $S_{i,j} = N_{i,j} \wedge \overline{R_{i,j}}$, and $E_{i,j} = W_{i,j} \wedge \overline{G_{i,j}}$. For the highest priority row and column, all the N and W inputs, respectively, are set to 1. If the arbitration cells are combinational circuits with propagation delay T, then, for an $n \times n$ crossbar, the arbiter reaches a valid arbitration configuration in $(2n-1)T$ time units.



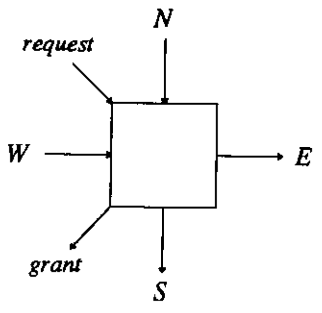


Figure 5: A two-step symmetric crossbar arbiter. An example arbitration is shown. Double squares indicate that the crosspoint has been requested. Shaded squares indicate that the crosspoint has been granted.

In order to ensure *fairness* with the two-step arbiter, the highest-priority row and column must not be fixed. Hence, while in the discussion above we assumed that crosspoint (1,1) has the highest priority, fairness requires that each crosspoint has an equal opportunity to have the highest priority. With top priority at a crosspoint other than (1,1), the scheme requires that the S output from the bottom row is connected to the N input of the top row and the E output from the right-most column is connected to the W input of the left-most column. Two n-stage token rings (circular shift registers) are used to keep track of the highest priority row and the highest priority column. The column shift register is advanced following each arbitration cycle while the row shift register is advanced following every n arbitration cycles. Figure 5 shows an example of the operation of the 4×4 two-step arbiter, in which the top left crosspoint has the highest priority. In the figure, double squares indicate that the crosspoint has been requested while shaded squares indicate that the crosspoint has been granted.

Simple probabilistic analysis can be used to evaluate the throughput of switches with the arbitration schemes discussed above. We assume that requests for various crosspoints are independent, and that the probability that there is at least one packet in buffer i destined for output j is p, for all $1 \le i,j \le n$. For a switch with multi-queue buffers, this implies that the request probability for any crosspoint is p. For a FIFO switch, there can be only one request on each row, so the request probability for any crosspoint is $\frac{1}{n}[1-(1-p)^n]$. The probability of a *grant* for crosspoint (i,j) will be denoted by $g_{i,j}$. The normalized throughput of the switch (the throughput per column) can be calculated using: $(\sum_{i=1}^{n}\sum_{j=1}^{n} g_{i,j})/n$. As an example, we will compare the three arbitration schemes for a 2×2 switch.

Probabilistic analysis of crossbars in FIFO switches is discussed in [16]. For FIFOA, if the crosspoint request probability is q, the normalized throughput is $1-(1-q)^2$. Using $q = ½[1-(1-p)^2]$, the normalized throughput is $2p-2p^2+p^3-¼p^4$.

For SOA, six cases must be considered. *(a) no requests*: The probability is $(1-p)^4$, and the throughput is 0. *(b) 1 request*: The probability is $\begin{bmatrix}4\\1\end{bmatrix} p(1-p)^3$ and the total throughput is 1. *(c) 2 requests in the same row or column*: The probability is $4p^2(1-p)^2$ and the total throughput is 1. *(d) 2 requests in different rows and in different columns*: The probability is $2p^2(1-p)^2$ and the total throughput is 2. *(e) 3 requests*: The probability is $\begin{bmatrix}4\\3\end{bmatrix} p^3(1-p)$ and the total throughput is 2. *(f) 4 requests*: The probability is p^4 and the total throughput is 2. The normalized throughput is 50% of the expected value of the total throughput, calculated from the analysis above, i.e., $2p-2p^2+2p^3-p^4$.

For the TSA, without loss of generality, we assume that crosspoint (1,1) has the highest priority. Let $v_{i,j}$ denote the probability that crosspoint (i,j) wins the (vertical) arbitration for column j in the first arbitration step. Clearly, $v_{1,1}=v_{1,2}=p$, and $v_{2,1}=v_{2,2}=(1-p)p$. Based on this, we can calculate the grant probability for the second arbitration step: $g_{1,1}=v_{1,1}=p$, $g_{1,2}=(1-v_{1,1})v_{1,2}=(1-p)p$, $g_{2,1}=v_{2,1}=(1-p)p$, and $g_{2,2}=(1-v_{2,1})v_{2,2}=[1-(1-p)p](1-p)p$. The normalized throughput is 50% of the sum of the grant probabilities, yielding $2p-2p^2+p^3-½p^4$.

The results of the above analysis are shown in Figure 6. As expected, SOA outperforms FIFOA. TSA, however, is even worse than FIFOA, especially when the request probability is high. In the extreme case where $p = 1$, the normalized throughput with TSA is 1/2, because the requests granted after the first arbitration step are all in the same row. This result indicates that a poor symmetric crossbar arbiter can negate the potential performance advantage of multi-queue buffers.

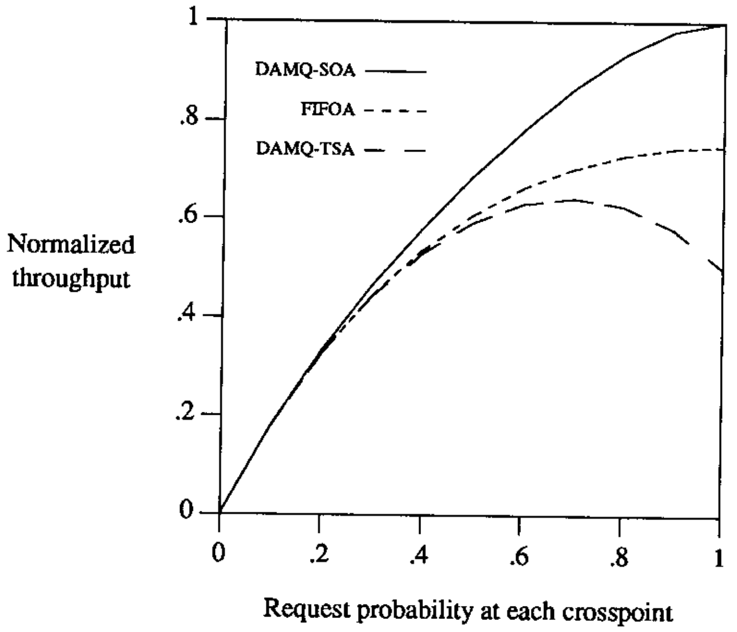


Figure 6: Normalized throughput based on static probabilistic analysis. A 2×2 switch with FIFOA, SOA, and TSA arbitration schemes.

B. The Skewed Two-Step Arbiter

A key problem with the *two step arbiter* (TSA) is that, in the first step, the top priority is given to the same row for all the column arbitrations. This increases the probability that multiple crosspoints in one row will win the first step, even though only one of them can be used at a time. A possible solution to this problem is to give the top priority to different rows for the different column arbitration. The "skew" in the column arbitration starting points is provided in the *skewed two-step arbiter* (STSA).

The operation of the STSA is identical to the TSA, except for the mechanism used to indicate the top priority cells. Specifically, instead of the two n bit circular shift registers which point to the top priority row and column, the STSA uses a single n bit circular shift register, which points to a "wrapped diagonal" of top priority crosspoints. For example, Figure 7 shows a STSA where the "diagonal" of high priority cells consists of cells (1,1), (2,4), (3,3), and (4,2). The result is that four connections are made by the STSA, while with the TSA, there is no way for four connections to be made for the specific pattern of requests shown.

The arbitration cells of the STSA are identical to those used in the TSA. The periphery of the array of arbitration cells, which changes the top priority cells to maintain fairness, is simpler since there is only one shift register instead of two. With the STSA, the horizontal (row) arbitration is performed in parallel with the vertical (column) arbitration. Hence, the STSA is faster (has smaller worst case delay) than the

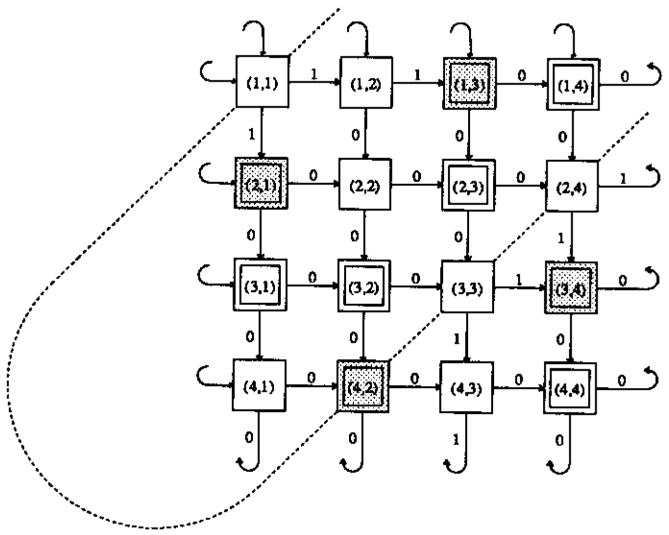


Figure 7: A skewed two-step symmetric crossbar arbiter. For the example arbitration shown, the top priority cells are (1,1), (2,4), (3,3), and (4,2).

TSA. Specifically, if the delay per cell is T time units, the worst case arbitration time for the STSA is only nT time units.

C. The Wave Front Arbiter

The low performance of the TSA is a direct result of the partitioning of the arbitration process into the two separate steps of column and row arbitration. As discussed earlier, this results in low throughput since multiple crosspoints on high priority rows are likely to win the first step and prevent other crosspoints on their columns from being used. With the *wave front arbiter* (WFA) the entire arbitration process is performed in one step, so higher throughput is expected.

Rather than starting with a top priority row, the arbitration process with WFA begins with one top priority arbitration cell. The arbitration cells reach their final configuration in a ''wave front'' that moves diagonally from the top left to the bottom right corner of the arbiter. Figure 8 shows the operation of the WFA with the top priority in arbitration cell (1,1). In order to maintain fairness, two n bit circular shift registers, vertical and horizontal, as with TSA, are used to select the current top priority cell. The horizontal shift register is shifted every cycle, while the vertical shift register is shifted every n cycles.

The critical difference between WFA and TSA as well as STSA is the function of the arbitration cell. Specifically, the definition of the $N_{i,j}$ signal is different from the one discussed in Subsection III.A. With TSA and STSA, $N_{i,j}$ indicates that none of the rows above have requested column j. With WFA, $N_{i,j}$ indicates that there are no granted requests for the crosspoints above (i,j). Hence, using the notation of Subsection III.A, $G_{i,j} = R_{i,j} \wedge N_{i,j} \wedge W_{i,j}$, $S_{i,j} = N_{i,j} \wedge \overline{G_{i,j}}$, and $E_{i,j} = W_{i,j} \wedge \overline{G_{i,j}}$. As with TSA and

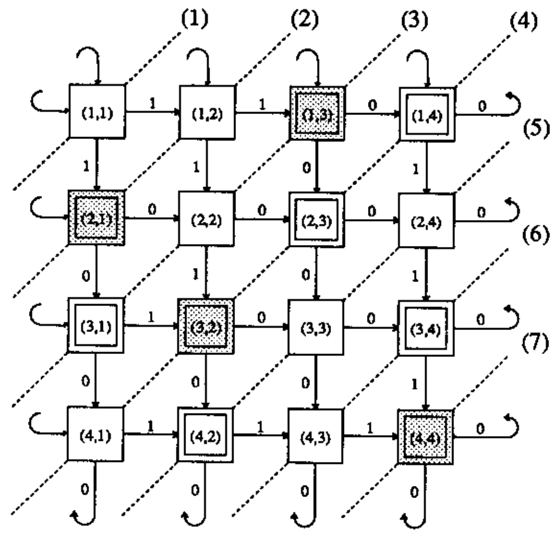


Figure 8: Wave front symmetric crossbar arbitration. Cell (1,1) has the top priority. The numbered diagonals indicate the progression of the arbitration wave front.

STSA, the cells are simple combinational circuits. If cell (1,1) has the top priority and a cell performs its operation in T time units, the outputs of cell (i,j) are stable in their final values after $(i+j-1)T$ time units. Hence, the arbitration completes after $(2n-1)T$ time units.

D. The Wrapped Wave Front Arbiter

With WFA, in the first stage of arbitration (in the first T time units), only one crosspoint is "processed" so at most one final *grant* signal is generated. This appears wasteful since even in the first stage it is possible to process n crosspoints which are guaranteed not to conflict. The n crosspoints of any "wrapped diagonal" of the arbitration array (see Figure 9) are guaranteed not to conflict since they are all on different rows and different columns. If the arbitration "wave front" begins with all n crosspoints of such a diagonal, the arbitration of n crosspoints instead of 1 crosspoint is completed after the first T time units. This basic idea is the basis of the *wrapped wavefront arbiter* (WWFA).

The arbitration cells used in the WWFA are identical to those used with the WFA. The difference between the schemes is only in the mechanism used to indicate the top priority cells when the arbitration process begins. Specifically, with WFA two circular n bit shift registers are used to indicate the top priority cell, as with TSA. With WWFA, a single n bit circular shift register is used to indicate the wrapped diagonal of top priority crosspoints, as with STSA.

Figure 9 shows the arbitration process where the diagonal of high priority cells consists of cells (1,1), (2,4), (3,3), and (4,2). If a cell performs its operation in T time units, the outputs of cell (i,j) are stable in their final values after $[((i+j-2) \textit{ mod } n)+1]T$ time units. Hence, the WWFA is faster than the

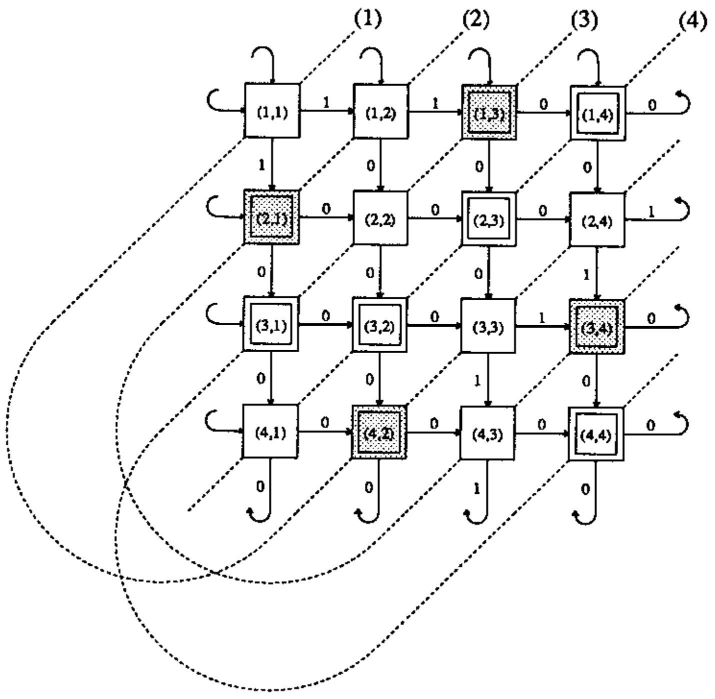


Figure 9: A wrapped wave front symmetric crossbar arbiter. The ''wrapped diagonal,'' (1,1), (2,4), (3,3), and (4,2) has the top priority. The numbered diagonals indicate the progression of the arbitration wave front.

WFA, completing an arbitration in nT time units.

E. Probabilistic Analysis of Arbitration of a 2×2 *Switch*

In this subsection we use simple probabilistic analysis, as in Subsection III.A, to evaluate the throughput of a 2×2 switch using the symmetric arbitration schemes discussed in the previous three subsections. The notation used follows the notation of Subsection III.A. This simple analysis provides an initial indication of the expected throughput. Realistic performance evaluation, using simulations which take into account the queueing effects, are discussed in Section IV.

The analysis of the STSA is similar to the analysis of TSA. Let $v_{i,j}$ denote the probability that crosspoint (i,j) wins the (vertical) arbitration for column j in the first arbitration step. Without loss of generality, we assume that the diagonal of top priority cells consists of cells (1,2) and (2,1). Clearly, $v_{2,1} = v_{1,2} = p$, and $v_{1,1} = v_{2,2} = (1-p)p$. Based on this, we can calculate the grant probability for the second arbitration step:

$$g_{1,2} = v_{1,2} = p, \quad g_{2,1} = v_{2,1} = p,$$

$$g_{1,1} = (1-v_{1,2})\,v_{1,1} = p(1-p)^2, \quad g_{2,2} = (1-v_{2,1})\,v_{2,2} = p(1-p)^2.$$

The normalized throughput is 50% of the sum of the grant probabilities, yielding $2p - 2p^2 + p^3$.

For WFA, we assume, without loss of generality, that cell (1,1) has the top priority. This implies that if cell (1,1) is requested, it is granted, i.e., $g_{1,1} = p$. Cells (1,2) and (2,1) are granted if requested and cell (1,1) has not been granted, i.e., $g_{1,2} = g_{2,1} = p(1-p)$. Cell (2,2) is granted if, and only if, either cell (2,2) is the *only* cell requested, or cell (1,1) is also requested, thus blocking possible grants for cells (1,2) and (2,1). Hence, $g_{2,2} = p(1-p)^3 + p^2$. The normalized throughput is 50% of the sum of the grant probabilities, yielding $2p - 2p^2 + \frac{3}{2}p^3 - ½p^4$.

It is easy to show that, for a 2×2 switch, WWFA always results in the same arbitration result as STSA. It should be noted that these two arbitration schemes are not identical with larger switches (see Section IV). For WWFA, we assume, without loss of generality, that the diagonal of top priority cell consists of cells (1,1) and (2,2). For both of these cells the implication is that if they are requested, they are granted, i.e., $g_{1,1} = g_{2,2} = p$. Cells (1,2) and (2,1) can be granted only if they are requested and neither cells (1,1) or (2,2) have been requested, i.e., $g_{1,2} = g_{2,1} = p(1-p)^2$. The normalized throughput is 50% of the sum of the grant probabilities, yielding $2p - 2p^2 + p^3$.

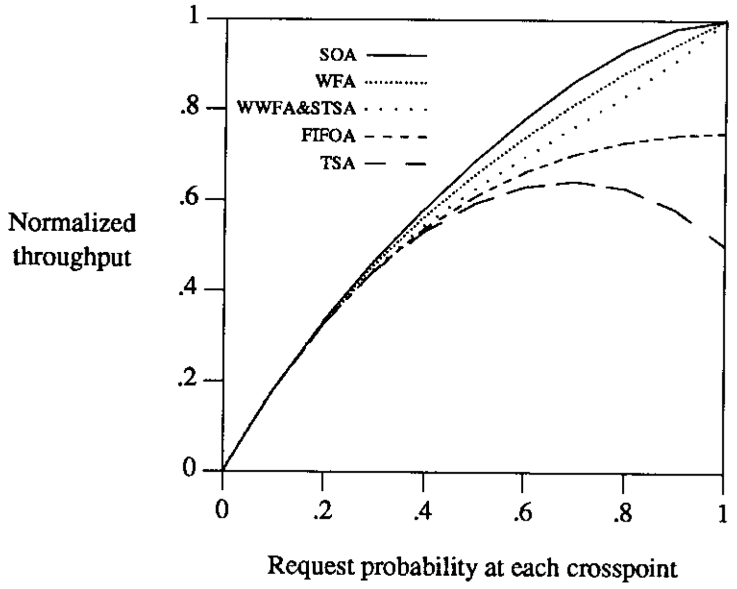


Figure 10: Normalized throughput based on static probabilistic analysis. A 2×2 switch with SOA, WFA, WWFA, STSA, FIFOA, and TSA arbitration schemes.

Figure 10 shows the results of the probabilistic analysis of a 2×2 switch with all the arbitration schemes discussed in this section. It should be noted that three of the practical arbitration schemes proposed in this section (STSA, WFA, and WWFA) achieve nearly that same performance as the statically optimal arbitration scheme. This important result, that the benefits of multi-queue buffers can be fully realized with practical arbitration schemes, is confirmed in the next section using simulations.

IV. Performance Evaluation

The probabilistic analysis used in Section III does not take into account the queueing effects and cannot be easily extended to evaluating the performance of a complete network rather than a single switch. In order to perform more realistic performance evaluation of the proposed arbitration schemes, we used an event-driven simulator [21], that allows detailed simulation of arbitrary objects interacting via messages.

Our simulations focus on a 64×64 Omega network [14] which uses *blocking* switches. This multistage buffered network operates synchronously. The minimum delay per stage is called a *stage cycle* [26]. A stage cycle is the latency of a packet per stage in an empty network, where there is no contention for buffer space or output ports. A packet may be forwarded to a switch in the next stage only if there is a free buffer slot in the input buffer of the switch.

The simulations are based on a traffic model with the following properties: 1) packets are of a fixed size, 2) during each stage cycle there is an equal probability of generating a packet at each of the source node, 3) packet destinations are uniformly distributed over all the network outputs. The performance measures used are the average latency, measured in stage cycles, and the normalized network throughput, which is the average number of packets received by each output of the network per stage cycle. As a measure of *fairness*, we use the "99th percentile latency," which is the minimum of the latencies of the 1% of the packets that received the poorest service (longest latencies) from the network. Ideally, all packets receive the same "quality" of service, and the 99th percentile latency will be the same as the average latency. In reality, even if all conflicts local to the switches are resolved in a fair way, the 99th percentile latency may be several times larger than the average latency (see Subsection IV.B). Any lack of fairness in resolving conflicts local to the switches will result in increasing the 99th latency and increasing the <u>difference</u> between the 99th percentile and average latencies.

We report the results from simulating a single switch as well as from simulating the entire 64×64 network. The single switch simulations were run with each sender generating approximately 3,000 packets. The network simulations were run with each sender generating approximately 1,500 packets. In order to eliminate start-up effects, statistics were gathered only after the first one third of the packets were received. Simulation was terminated once there was one sender that had completed sending all its allotted packets. In order to verify the validity of the results, for each rate of generating messages at the sources, multiple runs were performed with different random number seeds. At least four runs were performed for each data point reported in this section. The throughput and latency numbers used are the averages of results from these multiple runs. The throughputs for individual simulation runs were all within 3% of the throughput numbers reported. The average latencies reported are within 6% of the average latencies obtained in simulations (3% percent for all the network simulations). For the 99th percentile latencies, the result from each simulation run is an integer number of stage cycles. Hence, for low throughputs (and latencies), percentage variations in some 99th percentile latency results were larger than the percentage variations in average latencies. However, the results of the simulation runs were either within one stage cycle or 6% of the 99th percentile latencies reported.

In addition to the six arbitration schemes discussed in Section III (FIFOA, TSA, STSA, WFA, WWFA, and SOA), we consider two other possible schemes: 1) *fixed priority wave front arbitration* (FPWFA), which is identical to WFA except that the top priority always remains in the same arbitration cell, and 2) *longest queue first arbitration* (LQFA), which assigns priorities to inputs in direct proportion to the number of packets in their buffers and arbitrates between different queues in an input buffer by assigning queue priorities in direct proportion to the number of packets in each queue [11]. Since there is no need to rotate priorities with FPWFA, its implementation is expected to be simpler. LQFA would be difficult to implement efficiently, but it may be expected to perform better than other practical arbitration schemes. DAMQ buffers [22] are assumed in the evaluation of all seven symmetric crossbar arbitration schemes.

In the following two subsections we consider the effect of the crossbar arbitration scheme on the performance of 4×4 switches with input buffers that can accommodate up to four packets. We consider a single switch first and then the 64×64 network, consisting of three stages of 4×4 switches. The effects of switch size (number of inputs and outputs) and buffer size (number of packet slots per input buffer) are considered in Subsection C.

A. *Performance Evaluation of a Single* 4×4 *Switch*

Figure 11 shows the average latency versus throughput of a single 4×4 switch with four packets slots per input buffer. In general, the results corroborate the probabilistic analysis of Section III. The maximum throughput achieved, as well as the average latencies for high throughputs, are dependent on the arbitration policy. In particular, poor policies, such as FIFOA and TSA, achieve significantly lower maximum throughput than efficient policies, such as WFA or LQFA. WFA and WWFA provide approximately the same performance as the much more expensive LQFA, and only slightly worse performance than the theoretical SOA. While the probabilistic analysis of a 2×2 switch showed identical results for STSA and WWFA, similar analysis of 4×4 and larger switches shows that WWFA results in significantly better performance. This mediocre performance of STSA relative to WFA and WWFA is also shown in Figure 11.

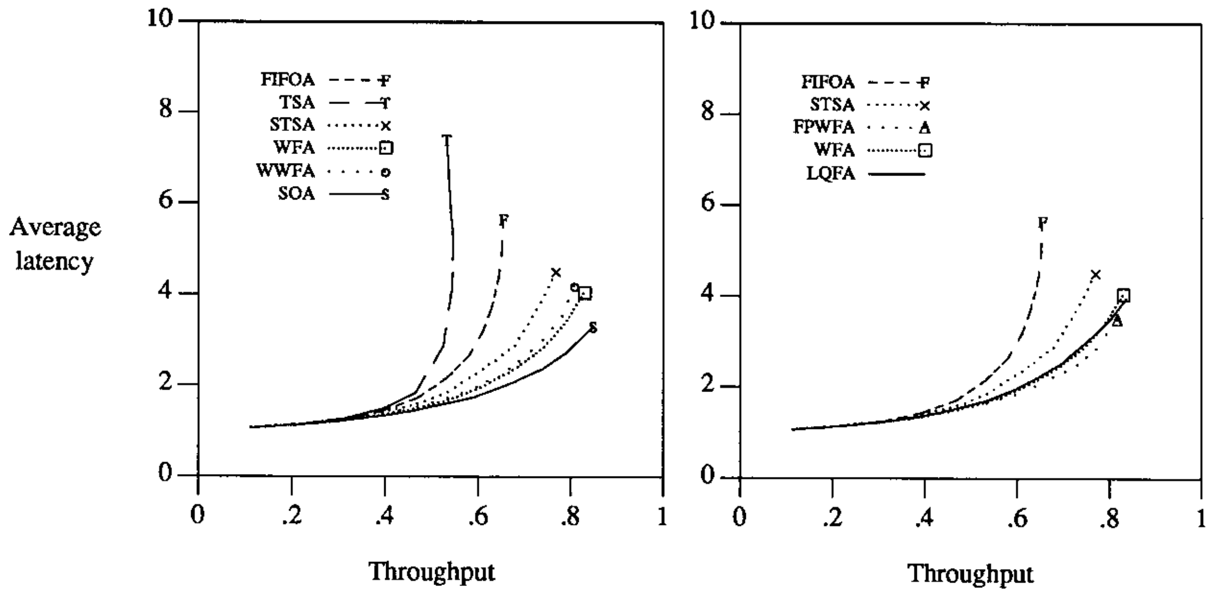


Figure 11: Average latency vs. normalized throughput of a single 4×4 switch with four packet slots per input buffer.

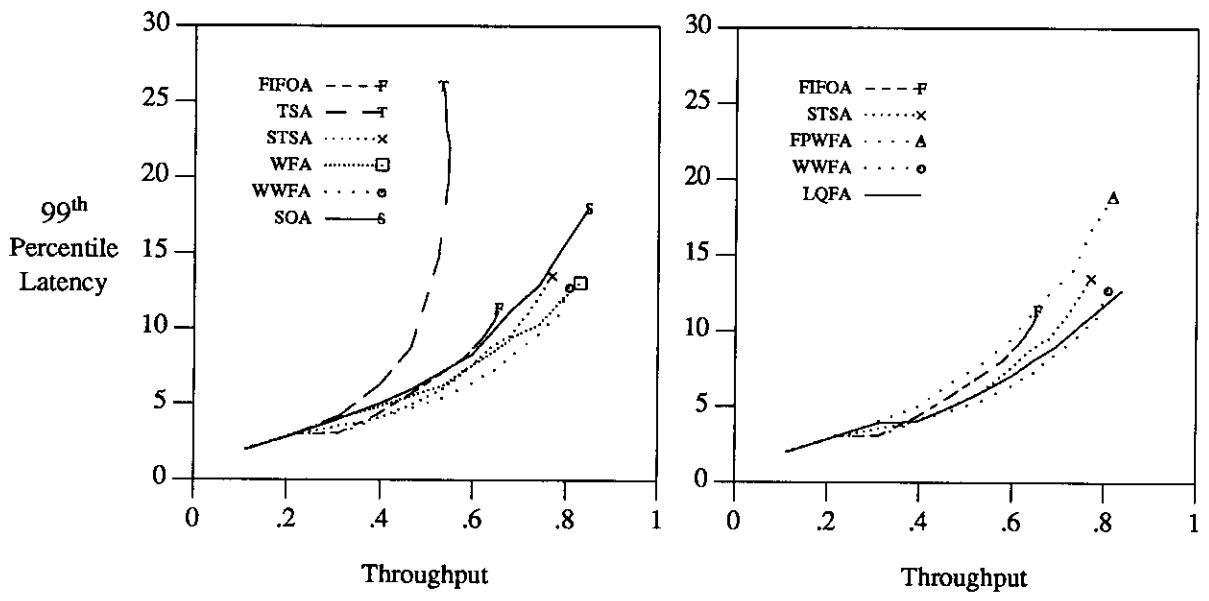


Figure 12: 99th percentile latency vs. throughput of a single 4×4 switch with four packet slots per input buffer.

The simulation results for 99th percentile latency are shown in Figure 12. TSA results in the worst performance with respect to 99th percentile latency as well as with respect to average latency. SOA results in relatively poor (high) 99th percentile latency. Thus, it appears that the optimization for maximum throughput is at the expense of fairness. As might be expected, fixed priority results in unfair arbitration — FPWFA results in the second highest 99th percentile latencies even though it achieves low average latencies (Figure 11). Relative to the other schemes, FIFOA performs well with respect to 99th percentile latency despite its poor average latency performance. This is due to the inherent fairness in a first-in-first-out mechanism.

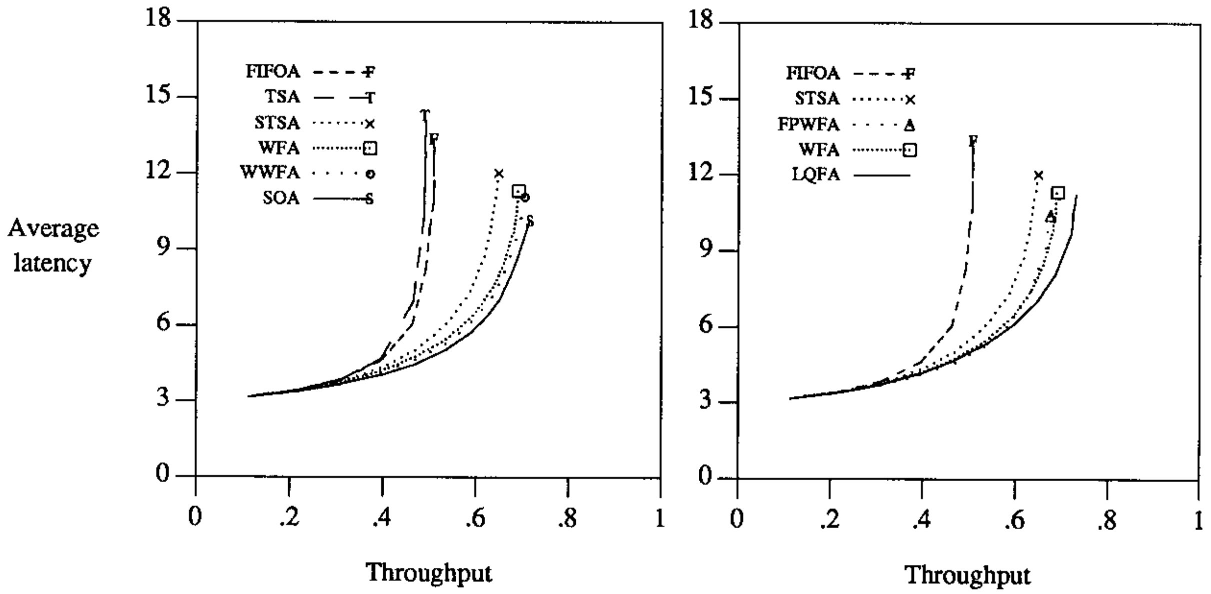


Figure 13: Average latency vs. normalized throughput of a 64×64 Omega network using 4×4 switches with four packet slots per input buffer.

B. Performance Evaluation of a Single 64×64 *Omega Network*

Figure 13 shows the average latency versus normalized throughput of a 64×64 Omega network, consisting of three stages of 4×4 switches with four packet slots per input buffer. Qualitatively, the results are similar to the results of the single switch simulations. One difference is that FIFOA is now shown to provide approximately the same performance as TSA, while the single switch simulations indicated that the FIFOA performance is significantly better. The reason for this apparent discrepancy is that the performance with FIFOA is more severely degraded by output ports which are blocked due to a full buffer in the next stage. This effect is very important, but is obviously not relevant to a single switch. Another interesting result is that LQFA performs best, even better than SOA. This is due to the fact that LQFA is optimized for reducing the probability of buffers overflowing, which is often the cause of blocking in a highly utilized network.

The simulation results for 99th percentile latency are shown in Figure 14. The effects of blocked output ports, discussed above, manifest themselves strongly in these measurement. For single switch simulations (Figure 12), the inherent fairness of FIFO resulted in similar 99th percentile latencies with

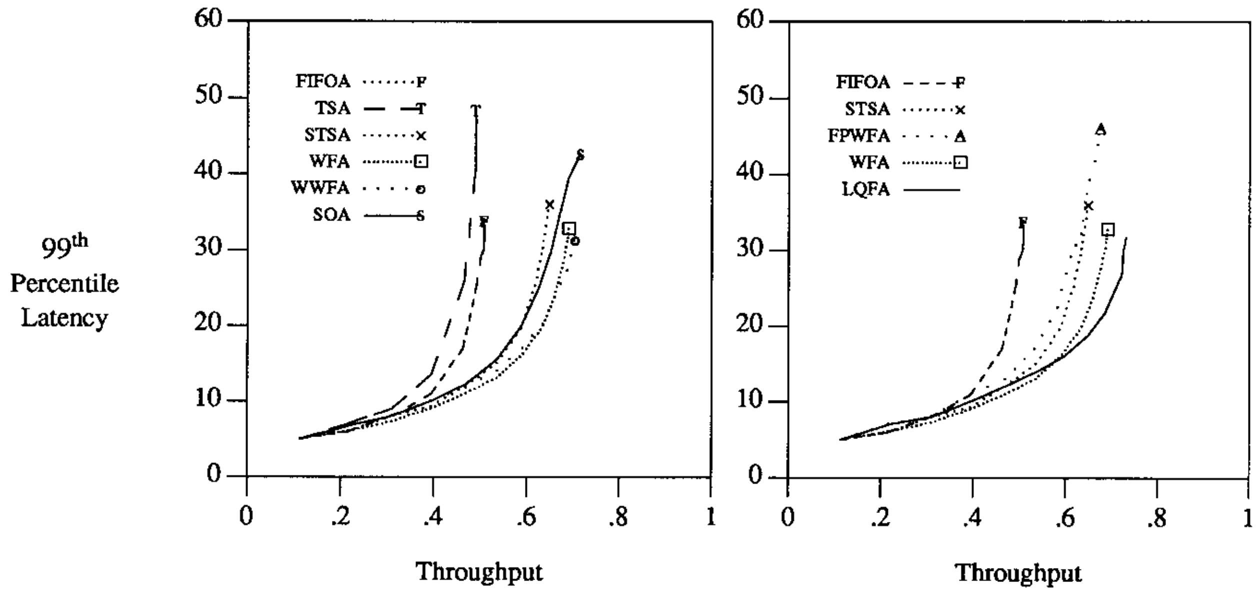


Figure 14: 99th percentile latency vs. throughput of a 64 × 64 Omega network using 4×4 switches with four packet slots per input buffer.

FIFOA and, for example, SOA. On the other hand, with the Omega network simulations the switches with FIFOA reach saturation at a much lower throughput, resulting in different performance characteristics. As could be expected, LQFA provides the best performance with respect to the 99th percentile latency as well as with respect to average latency. It should be noted that the practical arbitration schemes, WFA and WWFA, achieve the best performance relative to all the schemes other than LQFA.

C. The Impact of Buffer Size and Switch Size

All the simulations discussed above were done for 4×4 switches with four packet slots per input buffer. We consider here whether the results of these simulations are significantly different if two important parameters are changed: buffer size — number of packet slots per buffer, and switch size — number of inputs and outputs in each switch.

Figure 15 shows the average latency versus normalized throughput of 64×64 Omega networks of 4×4 switches with two, four, and six packet slots per input buffer. As shown elsewhere [6, 22], the maximum throughput increases as the buffer size increases. For small buffers (two packet slots), there is almost no difference between the various symmetric arbitration schemes and there is only a small performance improvement of those over FIFOA. The performance advantage of multi-queue buffers, and thus of symmetric crossbar arbitration, increases as the buffer size increases. The reason for this is that

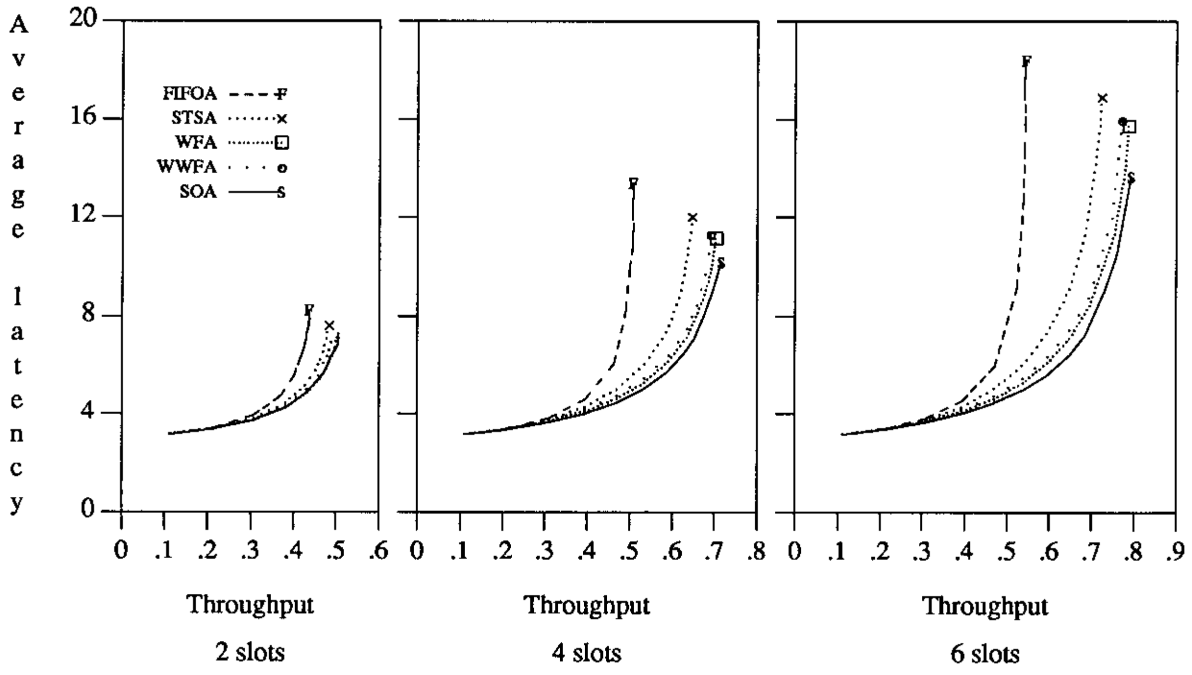


Figure 15: The impact of input buffer size. Average latency vs. normalized throughput of 64×64 Omega networks using 4×4 switches with various buffer sizes.

for a larger buffer there is a higher probability that the buffer contains packets destined to several outputs and thus, with a multi-queue buffer, there is a higher probability of sending a packet from the input port. The increase in the relative performance of multi-queue buffers over FIFO buffers is large as the buffer size increases from two slots to four slots, but is relatively small for buffer increase of from four slots to six slots. This is due to the fact that, with a 4×4 switch, the number of different destinations of packets in a particular buffer cannot be above 4, regardless of the size of the buffer. As the buffer size increases from two to four slots, there is a significant increase in the difference between the maximum achievable throughput with an efficient symmetric crossbar arbitration scheme (SOA) and an inefficient scheme (STSA). However, the difference in performance between efficient and inefficient schemes remains approximately the same, as the buffer size increases from four to six slots. The reason for this is, once again, that with uniform traffic and 4×4 switches, there is only a small increase in the expected number of destinations as the buffer size increases from four to six slots. Hence, the patterns of requests to the arbiter are approximately the same for four and six slot buffers, leading to approximately the same number of connections (packets transferred) per cycle.

Figure 16 shows the average latency per stage versus normalized throughput of 64×64 Omega networks with four packet slot input buffers in their switches, using 6, 3, and 2 stages of 2×2, 4×4, and 8×8 switches, respectively. It should be noted that the average latency through the network is the product

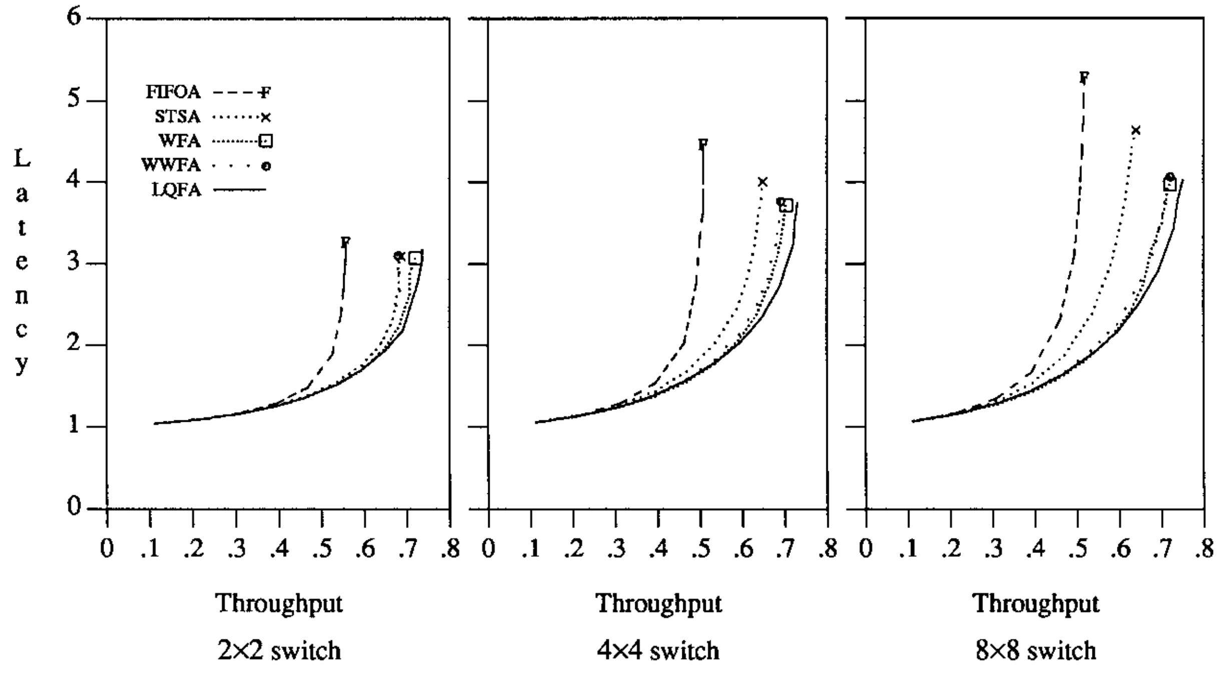


Figure 16: The impact of switch size. Average latency per stage vs. normalized throughput of 64×64 Omega networks with four packet slots per input buffer and various switch sizes.

of the average latency per stage, shown in Figure 16, and the number of stages. For FIFOA, as the switch size increases from 2×2 to 4×4, the maximum throughput decreases [26]. The maximum FIFOA throughput remains approximately the same when the switch size increases from 4×4 to 8×8. A relatively poor symmetric arbitration scheme (STSA) results in approximately the same behavior. As the switch size increases, there is an increase in the performance advantage of the good symmetric arbitration schemes (WFA, WWFA, LQFA), in terms of higher maximum throughput, over FIFOA and STSA. With WFA, WWFA, and LQFA there is no decrease in maximum throughput as the switch size is increased from 2×2 to 4×4. Furthermore, the maximum throughput increases when the switch size is increased to 8×8. The differences in maximum throughput among WFA, WWFA, and LQFA remain approximately the same for all three switch sizes. Hence, with increasing switch size, the arbitration scheme becomes more important for achieving the maximum possible performance. WFA and WWFA consistently achieve almost the same performance as LQFA, which is too complex for practical implementation.

V. Implementation of Symmetric Crossbar Arbiters

In the previous two sections we have shown that the crossbar arbitration policy has a significant impact on performance. The wave front and wrapped wave front arbitration schemes were shown to outperform many other schemes. WFA and WWFA, whose design is fundamentally simple to implement, were shown to achieve nearly the same performance as theoretical schemes which are optimized for high performance without regard to implementation complexity. For a final confirmation that WFA and WWFA are indeed practical arbitration schemes, this section discusses the circuit design and VLSI implementation of WFA and WWFA. Since the implementation of the two schemes is similar, the focus will be on WFA. However, most of the discussion applies to WWFA as well.

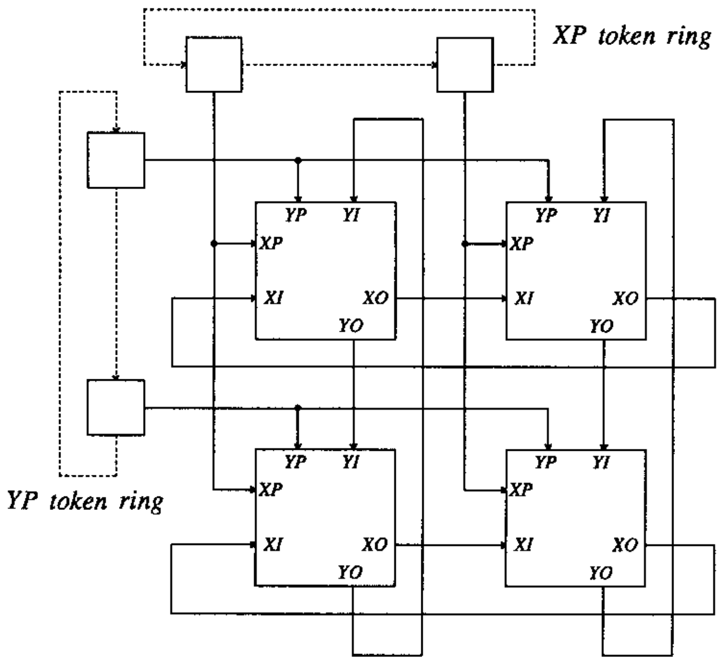


Figure 17: Organization of a 2×2 wave front arbiter.

A. Logic and Circuit Design

The structure of a 2×2 WFA and of a 2×2 WWFA arbiter is shown in Figures 17 and 18, respectively. The same arbitration cell, which is similar to the arbitration cell of Figure 5, can be used for both WFA and WWFA. The cell used in Figures 17 and 18 has the additional *XP* and *YP* inputs, which indicate top priority within a row (X direction), and within a column (Y direction), respectively. The *XI*, *XO*, *YI*, and *YO* signals correspond to the *W*, *E*, *N*, and *S* signals, respectively, of Figure 5. They are relabeled here due to the addition of the explicit priority signal, *XP* and *YP*. The *R* (request) and *G* (grant) signals are as in Figure 5, but are not shown in Figures 17 and 18.

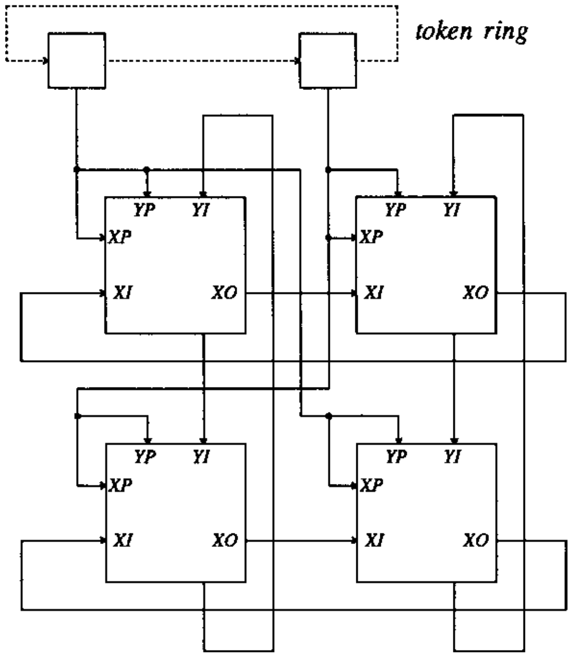


Figure 18: Organization of a 2×2 wrapped wave front arbiter.

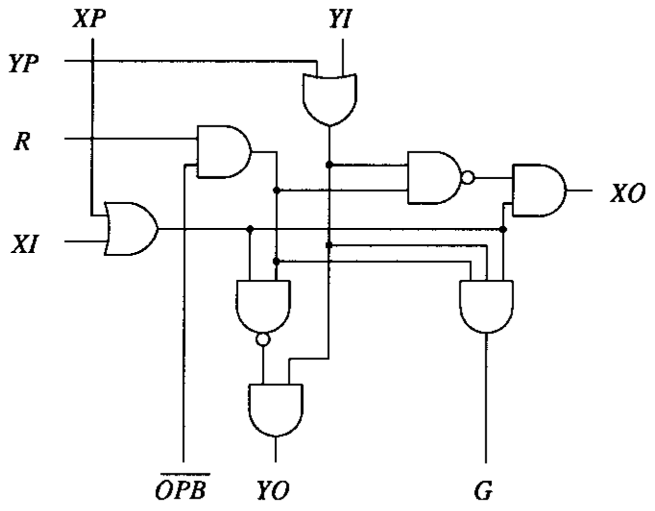


Figure 19: An arbitration cell. *XP* and *YP* indicate top priority in the horizontal and vertical directions, respectively. The $\overline{OPB}$ line is 1 when the output port is <u>not</u> blocked.

A logic diagram of the cells used in Figures 17 and 18 is shown in Figure 19. The *OPB* signal indicates that the output port is blocked, so there is no reason to include in the arbitration any crosspoint connecting to that port (column). The function implemented is a modification of the logic equations presented in Subsection III.C. The difference is that the priority signals, *XP* and *YP*, override the *XI* and *YI* signals. Since the inputs to the cells may go through more than one transition between 0 and 1 before the arbiter settles to its final value, static combination logic is used to implement the cells.

$$G = (R \wedge \overline{OPB}) \wedge (YI \vee YP) \wedge (XI \vee XP)$$
$$YO = (YI \vee YP) \wedge \overline{G}$$
$$XO = (XI \vee XP) \wedge \overline{G}$$

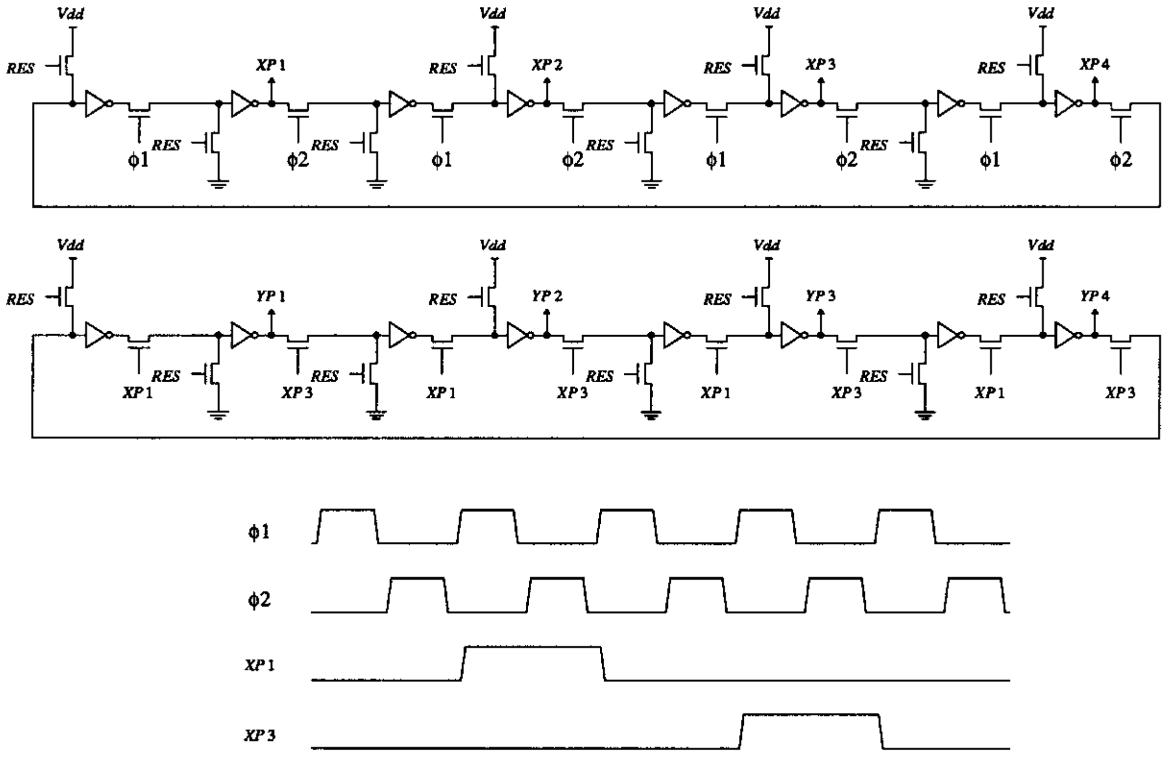


Figure 20: Horizontal and vertical priority token rings for a wave front arbiter. The reset signal (RES) initializes the two token rings to a valid state.

As discussed in Subsection III.C, for WFA, two token rings (shift registers) are used to identify the cell with the highest priority. The horizontal token is advanced every clock cycle, while the vertical token is advanced every *n* cycles, once the horizontal reaches its right-most cell. The token rings are implemented as simple dynamic shift registers which, on system reset, are initialized so that the first cell is 1 while all the others are 0 (Figure 20). For WWFA, only one token ring on this type is needed. With the WFA arbiter, for each clock cycle only a single cell has both the vertical and horizontal priority tokens. For WWFA, the priority line from the token ring delivers a priority token to a whole diagonal of *n* arbitration cells.

B. Prevention of Starvation

In the context of a communication switch, *starvation* of a particular queue occurs if it contains a packet that is never transmitted, even though other packets are transmitted through the switch. If the output ports are never blocked, the fact that each queue has the top priority every n^2 cycles with WFA or every n cycles with WWFA, guarantees that no packet will wait in a queue more than $b\ n^2$ or $b\ n$ cycles, respectively, where b is the number of packet slots in an input buffer. However, since output ports can be blocked due to full buffers in the following stage, it is possible for an ''unlucky'' queue to be prevented from transmission every time it has the top priority. Hence, packets in the ''unlucky'' queue may be delayed indefinitely.

Starvation can be prevented if a top priority queue which contains a packet maintains its top priority until it succeeds in sending one packet. With the WFA arbiter, this can be done by not shifting the horizontal XP shift register if the top priority cell was requested but not granted. For the WWFA arbiter, a similar solution is for the top priority wrapped diagonal to maintain its top priority if any of its arbitration cells were requested but not granted. It should be noted that if these solutions are used, the shift registers may not be shifted for many cycles so the dynamic shift registers shown in Figure 20 may have to be replaced by static shift registers.

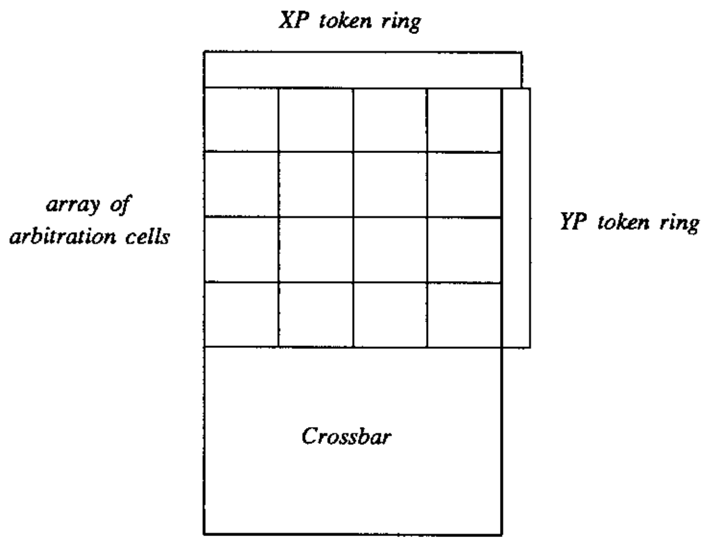


Figure 21: The floorplan of a 4×4 crossbar of eight-bit wide buses with a wavefront arbiter. The modules are drawn to scale.

C. VLSI Layout and Circuit Simulation

Due to their simple regular structure, the WFA and WWFA arbiters are amenable to efficient VLSI implementation. In order to determine the actual implementation complexity and performance, we have laid out a 4×4 crossbar of eight-bit wide internal buses with a wave front arbiter. The layout was done for CMOS technology, using the MOSIS scalable design rules. One possible floorplan of such a switch is based on integrating the crossbar with the arbiter so that each arbitration cell is together with the corresponding crosspoint. We have investigated this possibility but discovered that separating the arbiter from the crossbar, as shown in Figure 21, results in more compact layout.

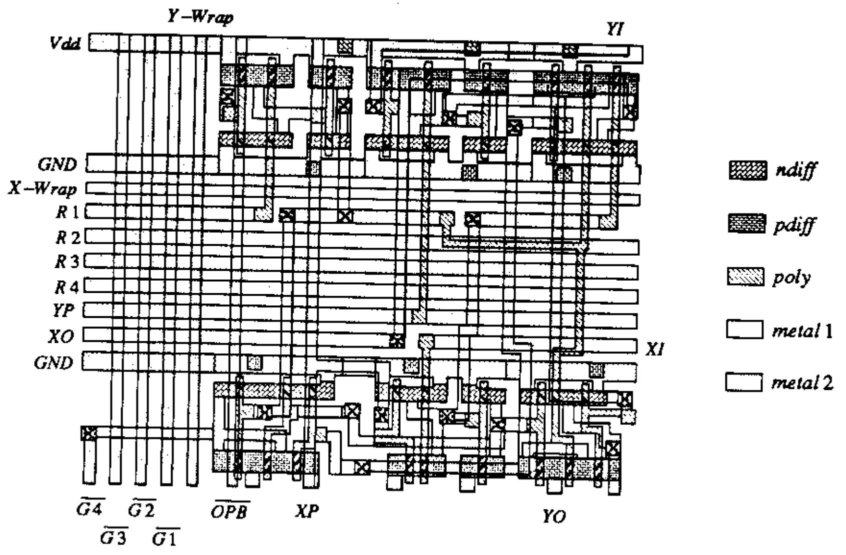


Figure 22: VLSI layout of a single WFA arbitration cell. This cell is 152 λ wide by 127 λ tall.

Figure 21 shows the floorplan of the crossbar with the arbiter. The size of the boxes in the figures are proportional to the size of the modules in the layout. The crossbar itself is 615 λ wide by 376 λ tall. The arbiter, with the priority shift registers is 649 λ by 527 λ. The layout of a single arbitration cell is shown in Figure 22. Assuming a 2 μ technology, circuit simulation using SPICE indicates that the worst case delay for reaching an arbitration result is 15.5 ns. In our current design of the ComCoBB switch [22, 7], the arbitration task must be completed in one 20 ns clock cycle. Hence, for the current design, our straightforward implementation of the wave front arbiter is fast enough. If higher speed is required, the WWFA arbiter can be used.

VI. Summary and Conclusions

Significant improvements in the performance of communication switches can be achieved by using multi-queue input buffers instead of FIFO buffers. A crossbar is often used to connect multi-queue input buffers to the output ports of the switch and an arbiter is needed to resolve conflicting requests for crossbar resources from the various queues. If all the queues of each input buffer are connected to a single crossbar input [22, 5], the arbitration task is *symmetric* with respect to inputs and outputs. Since the result of the arbitration for each crosspoint depends or influences the arbitration of all the crosspoints in the corresponding row and column, the arbiter is relatively complex and cannot be partitioned into independent row or column arbiters.

We have evaluated different symmetric crossbar arbitration schemes using static probabilistic analysis, single switch event-driven simulations, and simulations of a buffered Omega network. We have shown that a good symmetric crossbar arbitration scheme is essential to realizing the potential for performance improvement from multi-queue buffers. In particular, a bad scheme may result in *lower* performance than with FIFO buffers while a good scheme can increase the maximum throughput of the network by more than 40%. As the buffer size and switch size increase, the benefits of a good arbitration scheme increases.

We have investigated several arbitration schemes, including complex theoretical schemes (statically optimal and longest-queue-first arbitrations), which cannot be implemented but could be expected to perform better than practical schemes. We have shown that, in order to achieve fairness, it is important to use a scheme which does not assign static priorities to queues. Two arbitration schemes, which are based on the propagation of an arbitration ''wave'' across an array of arbitration cells, were shown to achieve nearly the same performance as the complex theoretical schemes. The proposed wave front and wrapped wave front arbiters are amenable to simple regular implementation in VLSI. For a 4×4 switch, the layout of a wave front arbiter is larger than the data portion of the corresponding crossbar of eight-bit wide buses. This indicates that the arbiter is a significant module in terms of size, as well as performance, in communication switch implementation. Circuit simulation of our CMOS implementation of the wave front arbiter, using 2 μ technology, indicates that, for a 4×4 crossbar, this arbiter can produce a valid and efficient crossbar configuration in 15.5 ns.

Appendix: The Names of the Arbitration Schemes

FIFOA	FIFO Arbitration
FPWFA	Fixed-Priority Wave Front Arbitration
LQFA	Long-Queue-First Arbitration
SOA	Static Optimal Arbitration
STSA	Skewed Two-Step Arbitration
TSA	Two-Step Arbitration
WFA	Wave Front Arbitration
WWFA	Wrapped Wave Front Arbitration

References

1. L. N. Bhuyan, ''Analysis of Interconnection Networks with Different Arbiter Designs,'' *Journal of Parallel and Distributed Computing* 4(4), pp. 384-403 (August 1987).
2. W. Crowther, J. Goodhue, R. Gurwitz, R. Rettberg, and R. Thomas, ''The Butterfly Parallel Processor,'' *IEEE Computer Architecture Newsletter*, pp. 18-45 (September/December 1985).
3. W. J. Dally and C. L. Seitz, ''The Torus Routing Chip,'' *Distributed Computing* **1**(4), pp. 187-196 (October 1986).
4. W. J. Dally, ''Fine-Grain Message Passing Concurrent Computers,'' *Proceedings of the Third Conference on Hypercube Concurrent Computers*, Pasadena, CA **1**, pp. 2-12 (January 1988).
5. W. J. Dally, ''Virtual-Channel Flow Control,'' *17th Annual International Symposium on Computer Architecture*, Seattle, WA (May 1990).
6. D. M. Dias and J. R. Jump, ''Packet Switching Interconnection Networks for Modular Systems,'' *Computer* **14**(12), pp. 43-53 (December 1981).
7. G. L. Frazier and Y. Tamir, ''The Design and Implementation of a Multi-Queue Buffer for VLSI Communication Switches,'' *International Conference on Computer Design*, Cambridge, MA, pp. 466-471 (October 1989).
8. A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, ''The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer,'' *IEEE Transactions on Computers* **C-32**(2), pp. 175-189 (February 1983).
9. D. B. Gustavson, ''Computer Buses — A Tutorial,'' *IEEE Micro* 4(4), pp. 7-22 (August 1984).
10. R. H. Halstead, T. L. Anderson, R. B. Osborne, and T. L. Sterling, ''Concert: Design of a Multiprocessor Development System,'' *13th Annual Symposium on Computer Architecture*, Tokyo, Japan, pp. 40-48 (June 1986).
11. M. Kumar and J. R. Jump, ''Performance Enhancement in Buffered Delta Networks Using Crossbar Switches and Multiple Links,'' *Journal of Parallel and Distributed Computing* **1**(1), pp. 81-103 (1984).

12. T. Lang, M. Valero, and I. Alegre, "Bandwidth of Crossbar and Multiple-Bus Connections for Multiprocessors," *IEEE Transactions on Computers* **C-31**(12), pp. 1227-1234 (December 1982).

13. T. Lang and M. Valero, "M-users B-servers Arbiter for Multiple-Busses Multiprocessors," *Microprocessing and Microprogramming*, pp. 11-18 (October 1982).

14. D. H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Transactions on Computers* **C-24**(12), pp. 1145-1155 (December 1975).

15. R. J. McMillen and H. J. Siegel, "The Hybrid Cube Network," *Distributed Data Acquisition, Computing, and Control Symposium*, pp. 11-22 (December 1980).

16. T. N. Mudge and B. A. Makrucki, "Probabilistic Analysis of a Crossbar Switch," *9th International Symposium on Computer Architecture*, Austin, Texas, pp. 311-319 (April 1982).

17. T. N. Mudge, J. P. Hayes, and D. C. Winsor, "Multiple Bus Architectures," *Computer* **20**(6), pp. 42-48 (June 1987).

18. G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," *1985 International Conference on Parallel Processing*, pp. 764-771 (August 1985).

19. D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, The MIT Press (1987).

20. C. L. Seitz, "The Cosmic Cube," *Communications of the ACM* **28**(1), pp. 22-33 (January 1985).

21. S. M. Swope and R. M. Fujimoto, "Simon II Kernel Reference Manual," Technical Report UUCS 86-001, University of Utah, Salt Lake City, UT (March 1986).

22. Y. Tamir and G. L. Frazier, "High-Performance Multi-Queue Buffers for VLSI Communication Switches," *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 343-354 (May 1988).

23. D. M. Taub, "Arbitration and Control Acquisition in the Proposed IEEE 896 Futurebus," *IEEE Micro* **4**, pp. 28-41 (August 1984).

24. M. K. Vernon and U. Manbar, "Distributed Round-Robin and First-Come First-Serve Protocols and Their Application to Multiprocessor Bus Arbitration," *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 269-277 (May 1988).

25. C. Whitby-Strevens, "The Transputer," *12th Annual Symposium on Computer Architecture*, Boston, MA, pp. 292-300 (June 1985).

26. H. Yoon, K. Y. Lee, and M. T. Liu, "Performance Analysis of Multibuffered Packet-Switching Networks in Multiprocessor Systems," *IEEE Transactions on Computers* **39**(3), pp. 319-327 (March 1990).