# *Chapter Three*
# Symmetric Crossbar Arbiters

Communication switches are basic components of interconnection networks for multiprocessors and multicomputers. The traffic through the switches is often delayed due to conflicting demands for resources, such as buffer space or output ports. Hence, switches must include *arbiters* that resolve conflicting resource demands. Efficient design and implementation of these arbiters is critical for maximizing network performance.

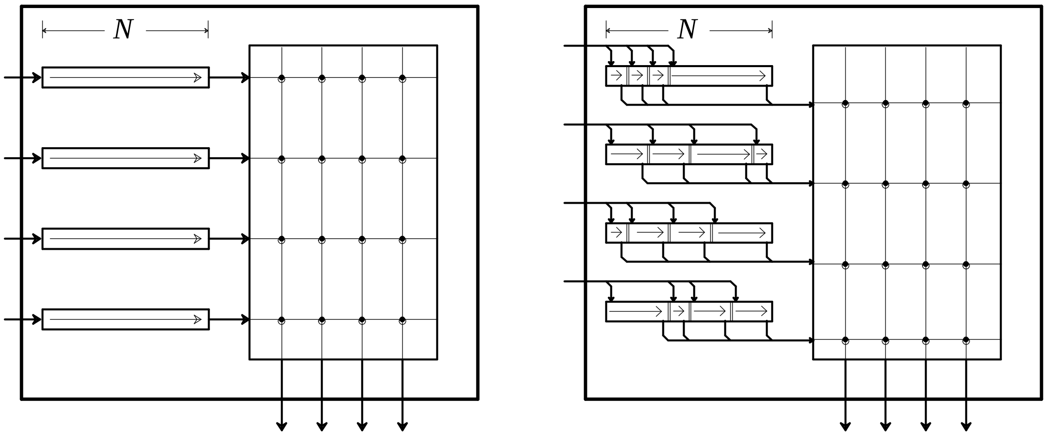
In order to maximize performance, recent communication switch designs allow packets at an input port, destined to different output ports, to be transmitted through the switch in any order. Each input port contends for *multiple* output ports but needs only one for full utilization. Similarly, each output 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 chapter discusses the design and implementation of *symmetric crossbar arbiters* for VLSI communication switches.

In the next section, symmetric crossbar arbitration is defined and its use in modern communication switches is explained. Section 3.2 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 of several symmetric crossbar arbitration schemes is evaluated in Section 3.3. The performance advantage of using an efficient arbitration policy is indicated using probabilistic analysis of single switches. More accurate and complete evaluation is based on event-driven simulations of individual switches and of multi-stage interconnection networks. The VLSI implementation of the best practical arbitration schemes is described in Section 3.4. Logic and circuit design is described, and circuit simulation is used to determine the performance of the proposed arbiters.

## 3.1. Symmetric Crossbar Arbitration

Crossbar switches often include FIFO buffers at their input ports for storing incoming packets that cannot be forwarded immediately due to output port contention or blocking [Dias81b]. The strict FIFO order of handling packets at each input port unnecessarily reduces the throughput of the switch [Tami88a, Tami92]. When the packet at the head of the queue is blocked, all other packets in the same buffer are also blocked, even if they are destined to idle output ports.

Multi-queue buffers avoid the shortcomings of FIFO buffers by partitioning each input buffer into several queues. The packets at the head of the queues in a buffer may be accessed in any order. Even if one of the queues of the buffer is blocked, it may be possible to transmit a packet from the head of another queue, which is destined to a different output port [McMi80, Kuma84, Tami88a, Tami92, Dall90]. A possible organization of a multi-queue buffer is to maintain one FIFO queue for each output port. In order to utilize buffer storage efficiently, it is desirable for all the queues to share common storage. A multi-queue buffer with this organization is called a *dynamically-allocated multi-queue* (DAMQ) buffer [Tami88a, Tami92]. Figure 3.1 shows the switch organizations with FIFO input buffers and DAMQ input buffers.




a) A switch with FIFO buffers. b) A switch with DAMQ buffers.

**Figure 3.1:** Switches with FIFO buffers and DAMQ buffers.

An $n \times n$ crossbar and its associated arbiter is shown in Figure 3.2. 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 this and the following chapters 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.

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

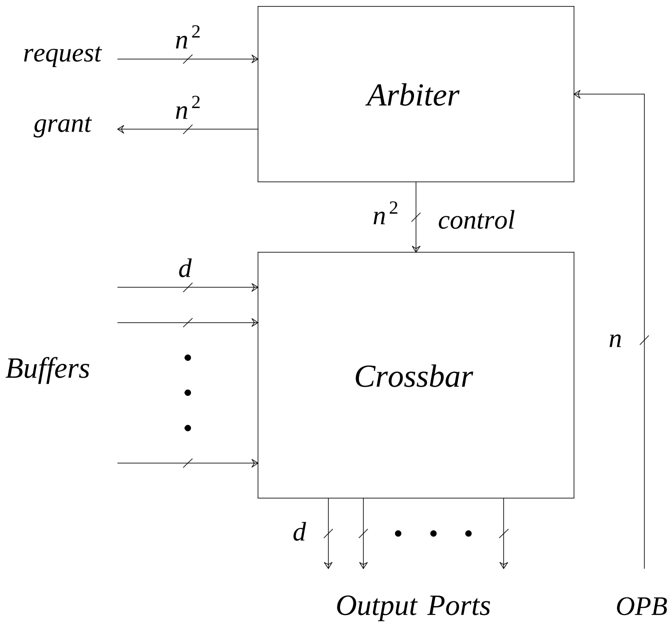



**Figure 3.2:** A crossbar arbiter in context. The crossbar connects $n$ $d$-bit buses from the input buffers to the $n$ output ports. Crosspoints (row/column connections) are requested using the $n^2$ *request* lines, and granted using the *grant* lines. The *OPB* (output port blocked) lines indicate which output ports are blocked and should not participate in the arbitration.

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 described in Chapter 2, 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.

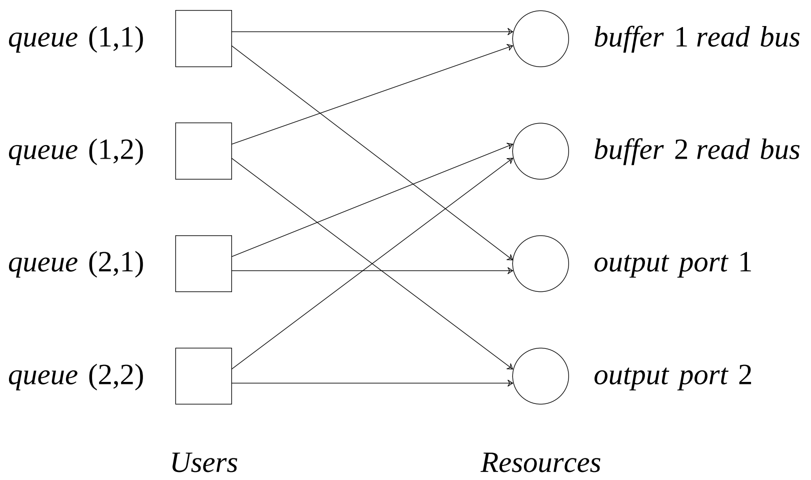
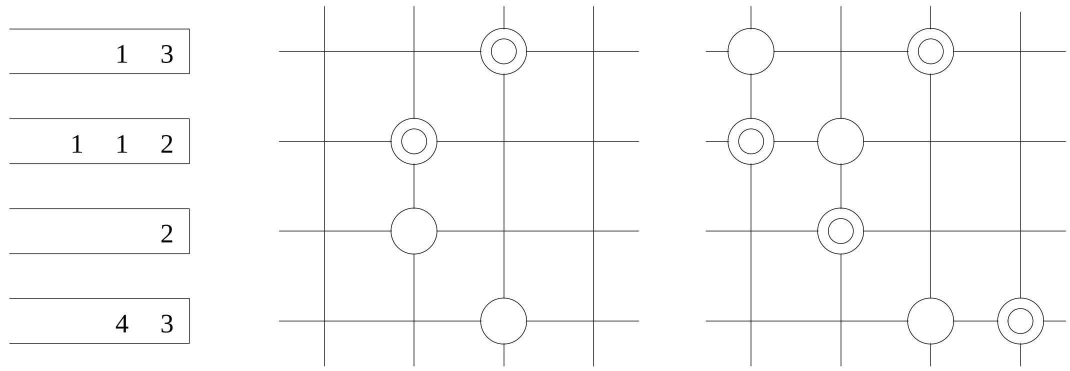



**Figure 3.3:** A user-resource model of symmetric arbitration of a 2×2 crossbar.

A user-resource model of a 2×2 crossbar with multi-queue input buffers is shown in Figure 3.3. 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 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*. Note that conventional crossbar arbitration is a special case of symmetric crossbar arbitration.




*Buffer Contents* *Arbitration* for *FIFO* *Arbitration* for *DAMQ*

**Figure 3.4:** 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 3.4 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.

## 3.2. Symmetric Crossbar Arbiters

With multi-queue input buffers there is the potential for achieving significantly higher network throughput than with conventional FIFO buffers [Tami88a]. 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 the importance of the arbitration scheme for achieving high performance is demonstrated. Several possible schemes are presented and discussed. The focus is 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 3.3 and 3.4, respectively.

### 3.2.1. 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 section we demonstrate that there is such a difference by using static probabilistic analysis to compare two arbitration schemes for a $2 \times 2$ switch: a practical scheme that follows naturally from previous work on arbiter design, 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 [Bhuy87, Vern88]. For a switch with multi-queue buffers, efficient symmetric crossbar arbitration maximizes the number of crosspoints utilized. We consider the performance of a *statically 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 3.5). 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.

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

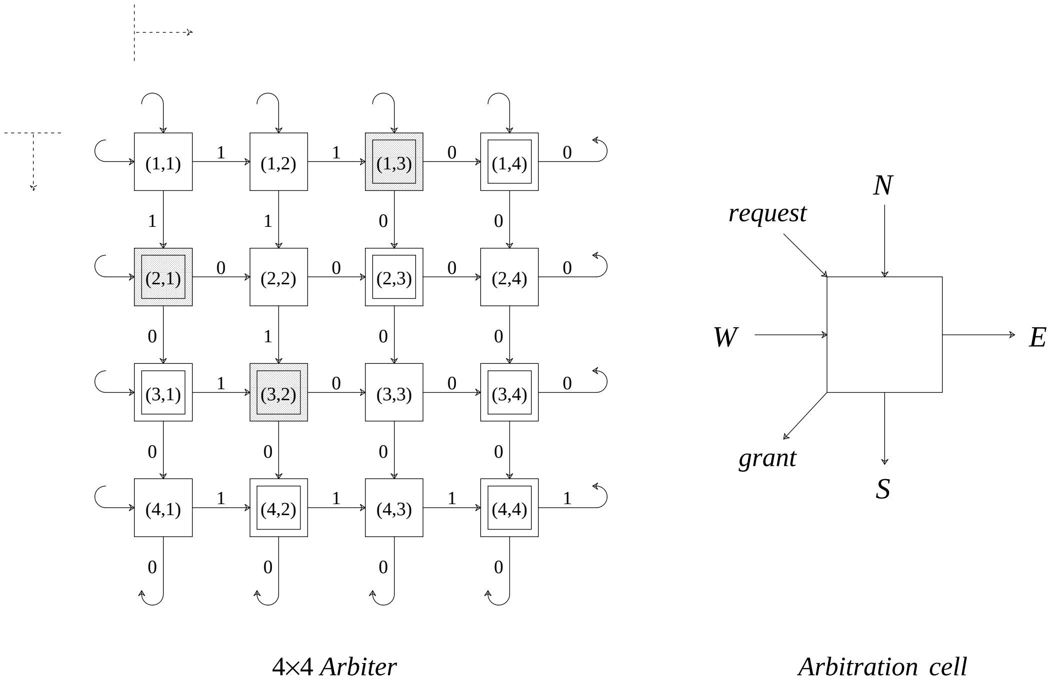



**Figure 3.5:** A two-step symmetric crossbar arbiter. An example arbitration is shown. Double squares indicate that the crosspoint has been requested. Shaded squares indicate that the crosspoint has been granted.

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 3.5 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: FIFOA, SOA, and TSA.

Probabilistic analysis of crossbars in FIFO switches is discussed in [Mudg82]. 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 $\binom{4}{1} p(1-p)^3$ 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 $\binom{4}{3} p^3(1-p)$ 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 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 $2p - 2p^2 + p^3 - ½p^4$.

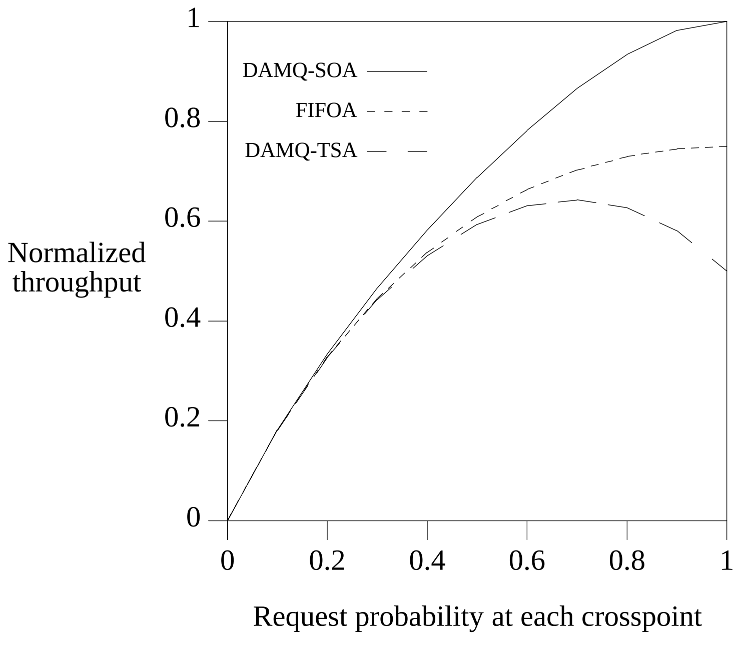



**Figure 3.6:** Normalized throughput based on static probabilistic analysis. A 2×2 switch with FIFOA, SOA, and TSA arbitration schemes.

The results of the above analysis are shown in Figure 3.6. 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.

### 3.2.2. 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 3.7 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.

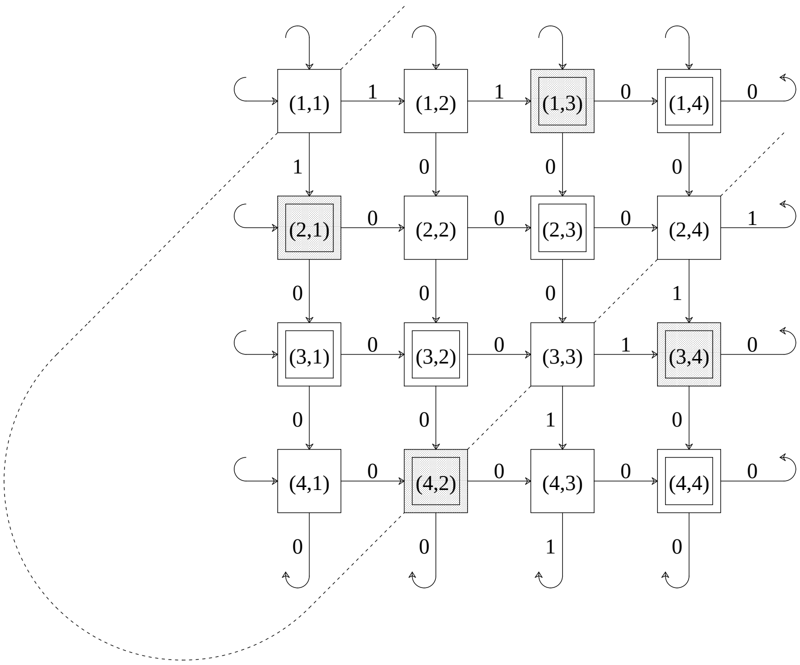



**Figure 3.7:** A skewed two-step symmetric crossbar arbiter. For the example arbitration shown, the top priority cells are (1,1), (2,4), (3,3), and (4,2).

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

### 3.2.3. 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

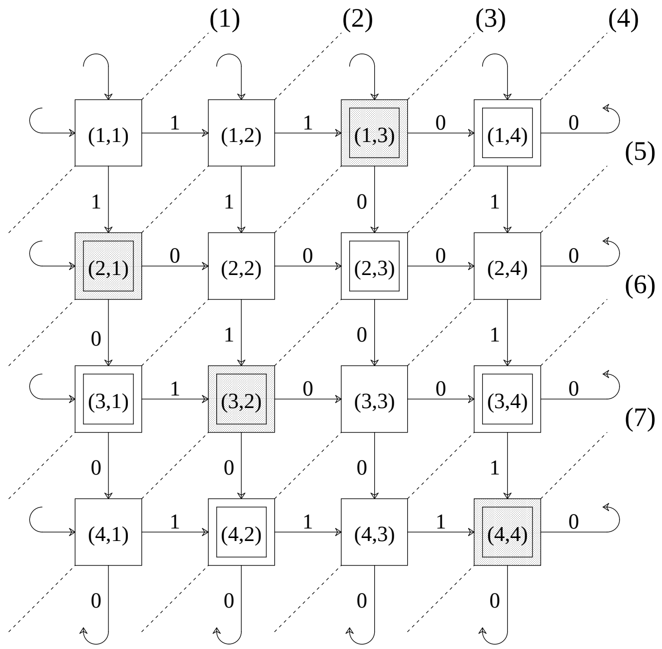



**Figure 3.8:** Wave front symmetric crossbar arbitration. Cell (1,1) has the top priority. The numbered diagonals indicate the progression of the arbitration wave front.

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 3.8 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 Section 3.2.1. 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 Section 3.2.1, $G_{i,j} = R_{i,j} \wedge N_{i,j} \wedge W_{i,j}$, $S_{i,j} = N_{i,j} \wedge \overline{G_{i,j}}$, and $E_{i,j} = W_{i,j} \wedge \overline{G_{i,j}}$. As with TSA and 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.

### 3.2.4. 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 3.9) 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

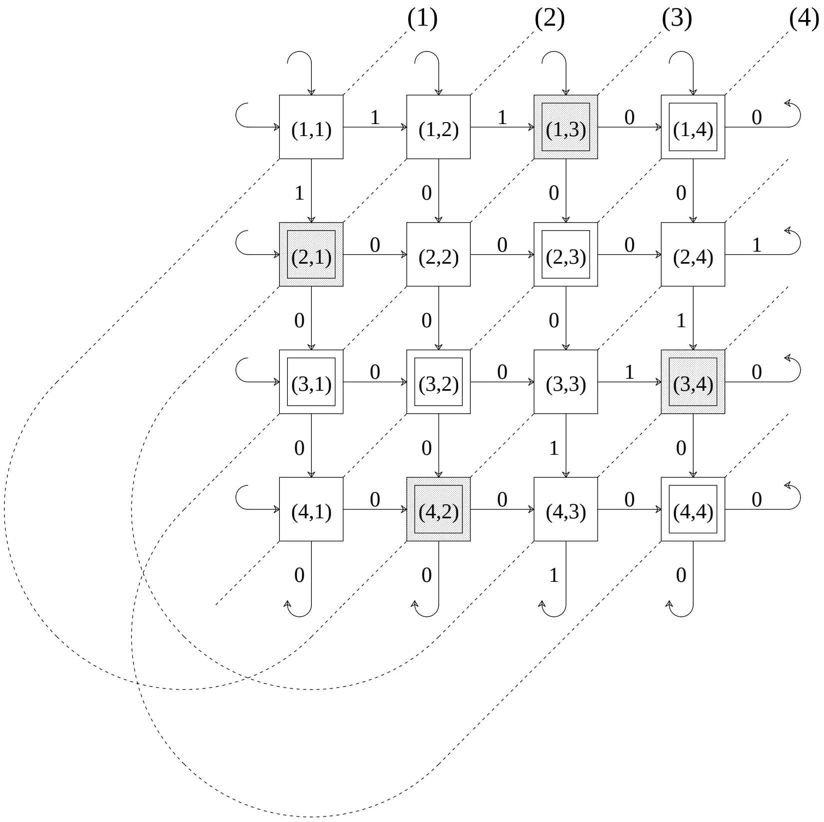



**Figure 3.9:** A wrapped wave front symmetric crossbar arbiter. The ‘‘wrapped diagonal,’’ (1,1), (2,4), (3,3), and (4,2) has the top priority. The numbered diagonals indicate the progression of the arbitration wave front.

the wrapped diagonal of top priority crosspoints, as with STSA.

Figure 3.9 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 $[((i+j-2)\ mod\ n)+1]\,T$ time units. Hence, the WWFA is faster than the WFA, completing an arbitration in $nT$ time units.

### 3.3. Performance Evaluation

In order to understand whether there is a significant performance difference between ‘‘good’’ and ‘‘poor’’ symmetric crossbar arbitration schemes, static probabilistic analysis and extensive simulations have been conducted. The simple static analysis provides an initial indication of the expected throughput for single switches. Realistic performance evaluation for both single switches and 64×64 interconnection networks, which takes into account the queueing effects, is shown using event-driven simulations.

#### 3.3.1. Simple Probabilistic Analysis for a 2×2 Switch

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 $2p - 2p^2 + \frac{3}{2}p^3 - \frac{1}{2}p^4$.

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 3.3.3). 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 3.10 shows the results of the probabilistic analysis of a 2×2 switch with all the arbitration schemes discussed in this section. As expected, SOA outperforms all the practical schemes. It should be noted that three of the practical arbitration schemes proposed (STSA, WFA, and WWFA) achieve nearly the same performance as the SOA scheme. This important result, that the benefits of multi-queue buffers can be fully realized with practical arbitration schemes, is confirmed in Section 3.3.3.

On the other hand, TSA 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.

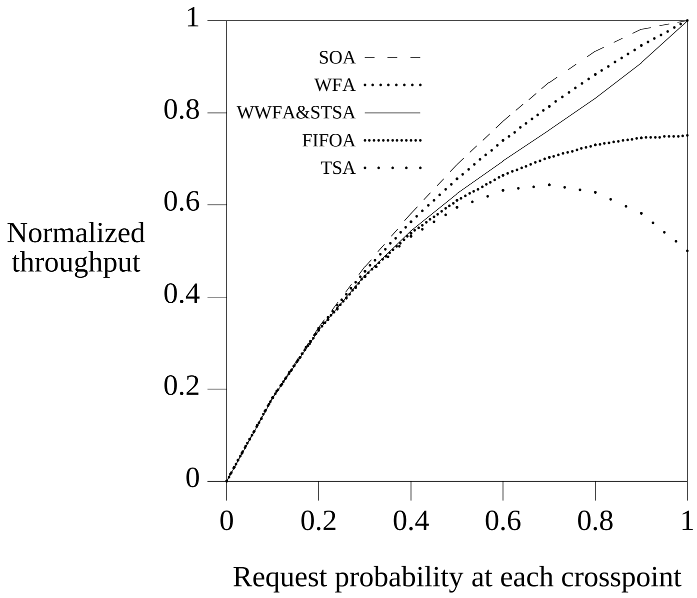



**Figure 3.10:** Probabilistic analysis of throughput for 2×2 switches. A 2×2 switch with SOA, WFA, WWFA, STSA, FIFOA, and TSA arbitration schemes.

### 3.3.2. Probabilistic Analysis for an n×n Switch

The probabilistic analysis in Section 3.3.1 is only practical for small switches since the number of cases is limited. In this section, a more general method is presented for performance analysis of the different arbitration schemes for any size $n \times n$ switches. The SOA scheme is not included here due to the difficulty of analyzing its performance for any size switches.

For FIFOA, if the crosspoint request probability is $q$, the normalized throughput is $1-(1-q)^n$ for an $n \times n$ switch. As shown in Section 3.2.1, $q$ is equal to $\frac{1}{n}[1-(1-p)^n]$. Hence, the normalized throughput is $1-\left\{1-\frac{1}{n}[1-(1-p)^n]\right\}^n$.

For TSA, as in Section 3.2.1, we can assume that crosspoint (1,1) has the top priority without loss of generality. Let $v_{i,j}$ denote the probability that crosspoint $(i,j)$ wins the arbitration for column $j$ in the first arbitration step. Then $v_{i,j} = (1-p)^{i-1} p$. The throughput for row $i$ is $1 - \prod_{j=1}^{n} (1 - v_{i,j})$, which is equal to $1 - \prod_{j=1}^{n} [1 - (1-p)^{i-1} p]$. The normalized throughput is 1/*n* of the total throughput, yielding $\frac{1}{n} \sum_{i=1}^{n} \left\{ 1 - \prod_{j=1}^{n} [1 - (1-p)^{i-1} p] \right\}$.

For STSA, we can assume that the wrapped diagonal (1,1), $(n,2)$, $(n-1,3)$, $\cdots$, $(2,n)$ has the top priority without loss of generality. Since the throughput for any row or any column is the same due to the symmetry of STSA, we will derive the normalized throughput by using the sum of the grant probabilities of row 1 in the following. Again, let $v_{i,j}$ denote the probability that crosspoint $(i,j)$ wins the arbitration for column $j$ in the first arbitration step. Then $v_{1,j} = (1-p)^{j-1} p$. The grant probability of crosspoint $(1,j)$ can be calculated based the equation: $g_{1,j} = \prod_{k=1}^{j} (1 - v_{1,k}) v_{1,j}$. The throughput of row 1 is $\sum_{j=1}^{n} g_{1,j}$, which is equal to $\sum_{j=1}^{n} \prod_{k=1}^{j} (1 - v_{1,k}) v_{1,j}$. Hence, the normalized throughput is:

$$\sum_{j=1}^{n} \left\{ \prod_{k=1}^{j} [1 - (1-p)^{k-1} p] (1-p)^{j-1} p \right\}.$$

The analysis of WFA is more complex than the above schemes, because the arbitration progresses in two dimensions instead of one dimension. For a boundary crosspoint $(1,j)$, the probability of getting the grant is $(1-p)^{j-1} p$, because it can get the grant only when the crosspoints $(1,k)$, $k < j$, do not request it. Similarly, the probability of getting the grant for a boundary crosspoint $(i,1)$ is $(1-p)^{i-1}p$. For an arbitrary ''inside'' crosspoint, the analysis is not obvious, especially for crosspoints $(i,j)$ where $i$ and $j$ are large numbers. In the following theorem, we establish an equation for the relation between $g_{i-1,j-1}$, $g_{i,j-1}$, and $g_{i,j}$ where $i,j \geq 2$. By this equation and the already available results in the above for the boundary crosspoints, we can derive the grant probabilities for all the crosspoints.

**Theorem 3.1.** In the wave front arbitration scheme, the grant probabilities of crosspoints $(i,j)$, $(i-1,j-1)$, and $(i,j-1)$ have the relation:

$$g_{i,j} = [\sum_{k=0}^{i-2}(1-p)^k p\,]\, g_{i-1,j-1} + (1-p)^i\, g_{i,j-1}$$

where each crosspoint has an independent request probability $p$, and $g_{i,j}$ is the grant probability of crosspoint $(i,j)$.

**Proof.** The key to deriving the grant probability for an arbitrary crosspoint is to condition the boundary crosspoints $(i,1)$, $1 \leq i \leq n$. Let $P[(i,j)]$ denote the probability that crosspoint $(i,j)$ gets the grant, and $P[(i,j)\,|\,(x,y)]$ the probability that crosspoint $(i,j)$ gets the grant, given the condition that $(x,y)$ gets its own required grant. Then,

$$\begin{aligned}
P[(i,j)] = {} & P[(1,1)] \cdot P[(i,j)\,|\,(1,1)] \\
& + P[(2,1)] \cdot P[(i,j)\,|\,(2,1)] \\
& + \cdots \\
& + P[(i,1)] \cdot P[(i,j)\,|\,(i,1)] \\
& + U \cdot W.
\end{aligned}$$

$U$ is the probability that crosspoints $(k,1)$, $k \le i$, do not get the grant. $W$ is the probability that crosspoint $(i,j)$ gets the grant, given the condition that crosspoints $(k,1)$, $k \le i$, do not get the grant. $P[(i,j)\,|\,(1,1)]$ is actually equal to $P[(i-1,j-1)]$. The reason for this is that, given the condition that (1,1) gets the grant, we can virtually remove row 1 and column 1 for deriving the original $P[(i,j)]$. When (1,1) gets the grant, no other crosspoints in the same row and the same column can be granted. The other crosspoints in row 1 will ’’pass’’ the value received from the vertical input to the vertical output. Hence, row 1 can be removed without affecting the grant probability of $(i,j)$. Similarly, column 1 can be removed without affecting the grant probability of $(i,j)$. As a result, $P[(1,1)] \cdot P[(i,j)\,|\,(1,1)]$, which is the first product term in the above equation, is equal to $p \cdot g_{i-1,j-1}$. For the second product term $P[(2,1)] \cdot P[(i,j)\,|\,(2,1)]$, we can virtually remove row 2 and column 1 by the same reasoning. Hence this term is equal to $(1-p)p \cdot g_{i-1,j-1}$. The following product terms in the equation can be derived likewise except the last two product terms. The product term $P[(i,1)] \cdot P[(i,j)\,|\,(i,1)]$ is equal to 0. This is due to the fact that $P[(i,j)\,|\,(i,1)]$ equals 0, because $(i,j)$ cannot get the grant if (i,1) gets it. In the last product term, $Q$ is equal to $(1-p)^i$, and $R$ is equal to $g_{i,j-1}$ because column 1 has no effect on $P[(i,j)]$ and thus can be virtually removed for this. Hence the last product term is equal to $(1-p)^i g_{i,j-1}$. As a result,

$$g_{i,j} = p \cdot g_{i-1,j-1} + (1-p)p \cdot g_{i-1,j-1} + (1-p)^2 p \cdot g_{i-1,j-1} + \cdots$$
$$+ (1-p)^{i-2} p \; g_{i-1,j-1} + 0 + (1-p)^i g_{i,j-1}.$$

This can be expressed as

$$g_{i,j} = [\sum_{k=0}^{i-2} (1-p)^k p\,]\, g_{i-1,j-1} + (1-p)^i \cdot g_{i,j-1}. \qquad \square$$

By using the grant probabilities for the boundary crosspoints and the recurrence equation in Theorem 3.1, we can derive, in order, all the grant probabilities of crosspoints (2,2), (2,3), (2,4), $\cdots$, $(2,n)$, (3,2), (3,3), (3,4), $\cdots$, $(n,n)$. The total throughput is the sum of all the grant probabilities. Hence the normalized throughput can be obtained from $\sum_{i=1}^{n}\sum_{j=1}^{n} g_{i,j}$.

Similar to WFA, the arbitration in WWFA progresses in two dimensions and the analysis can be based on a recurrence equation. Again, to calculate the normalized throughput, the grant probability for each crosspoint is derived first. In WWFA, each crosspoint in the same wrapped diagonal has the same grant probability due to the symmetry. Let $r_k$, $1 \le k \le n$, denote the grant probability of any crosspoint in the $k$th wrapped diagonal, and $g_{i,j} = r_k$ where $[(i+j-1)\ mod\ n] + 1 = k$. Note that the first wrapped diagonal consists of crosspoints $(n,1)$, $(n-1,2)$, $(n-2,3)$, $\cdots$, $(1,n)$, and any of them has the grant probability $r_1$. In the following theorem, we show that $r_k$ can be derived from $r_m$ where $1 \le m \le k-1$.

**Theorem 3.2.** In the wrapped wave front arbitration scheme,

$$r_k = (1-p)^2 \sum_{l=0}^{k-2} \left[ \binom{k-2}{l} (1-p)^{k-2-l}\, p^l\, r_{k-1-l} \right]$$

where each crosspoint has an independent request probability $p$, and $r_k$ is the grant probability of any crosspoint $(i,j)$ in the same wrapped diagonal where $[(i+j-1)\ mod\ n] + 1 = k$.

**Proof.** Assume the crosspoints in the first wrapped diagonal are polled first in WWFA, so $r_1 = p$. For the other wrapped diagonals of crosspoints, we can condition the first wrapped diagonal to get their grant probabilities. Since the arbitration wave front moves toward the right bottom corner, in the first wrapped diagonal only the requests of the crosspoints between crosspoints $(n+1-j, j)$ and $(i, n+1-i)$ and these two crosspoints themselves affect the grant probability of crosspoint $(i, j)$. Let $Q[l]$ denote the probability that there are $l$ crosspoints between crosspoints $(n+1-j, j)$ and $(i, n+1-1)$ which get the grant. Furthermore, let $Q[(i, j) \mid l]$ denote the probability that crosspoint $(i, j)$ gets the grant, given the condition that there are $l$ crosspoints between crosspoints $(n+1-j, j)$ and $(i, n+1-i)$ which get the grant. Since crosspoints $(n+1-j, j)$ and $(i, n+1-i)$ cannot have requests if crosspoint $(i, j)$ gets the grant, we can establish the following equation:

$$
\begin{aligned}
r_k = g_{i,j} = (1-p)^2 \big( & Q[0]\, Q[(i,j) \mid 0] \\
& + Q[1]\, Q[(i,j) \mid 1] \\
& + Q[2]\, Q[(i,j) \mid 2] \\
& + \cdots \\
& + Q[k-2]\, Q[(i,j) \mid k-2] \big).
\end{aligned}
$$

Note that there are $k-2$ crosspoints between crosspoints $(n+1-j, j)$ and $(i, n+1-i)$. The term $(1-p)^2$ is the probability that $(n+1-j, j)$ and $(i, n+1-i)$ have no requests. The term $Q[0]\, Q[(i,j) \mid 0]$ is equal to $\binom{k-2}{0} (1-p)^{k-2} r_{l-1}$, since $Q[0]$ equals $\binom{k-2}{0} (1-p)^{k-2}$ and $Q[(i,j) \mid 0]$ equals $r_{k-1}$. The reason for this is that, if all the crosspoints in the first wrapped diagonal have no requests, they are virtually nonexistent from the viewpoint of deriving $g_{i,j}$. The term $Q[1]\, Q[(i,j) \mid 1]$ is equal to $\binom{k-3}{1} (1-p)^{k-3} r_{l-2}$. The reason why $r_{l-2}$ is used here is that this requesting crosspoint between $(n+1-j, j)$ and $(i, n+1-i)$ blocks all the crosspoints in its associated row and column. In other words, this row and this column are virtually nonexistent from the viewpoint of deriving $g_{i,j}$. From the arbitration behavior in this case, $Q[(i,j)\,|\,1]$ is equal to $r_{l-2}$. The same reasoning can be applied to the following terms in the above equation. Hence, the equation can be expressed as:

$$r_k = (1-p)^2 \sum_{l=0}^{k-2} \left[ \binom{k-2}{l} (1-p)^{k-2-l} p^l \, r_{k-1-l} \right]. \qquad \square$$

This theorem basically is a recurrence equation, which can be used to derive $r_k$ based on $r_m$ where $1 \le m \le k-1$. Since $r_1$ is equal to $p$, the grant probabilities for the crosspoints in the other wrapped diagonals can be derived in the sequence of $r_2, r_3, \cdots, r_n$ using the equation. The normalized throughput is the average of the grant probabilities for all the wrapped diagonals, which is $\frac{1}{n} \sum_{k=1}^{n} r_k$

Figures 3.11, 3.12, 3.13, and 3.14 show the probabilistic analysis results for FIFOA, TSA, STSA, WFA, and WWFA for 4×4, 8×8, 16×16, and 64×64 switches. In general, performance comparison between different arbitration schemes in these switches is similar to the 2×2 switch. However, the performance difference between ‘‘good’’ schemes and ‘‘poor’’ schemes increases as the switch size increases. When the switch size is large, WFA and WWFA significantly outperform the other schemes, with WFA slightly better.

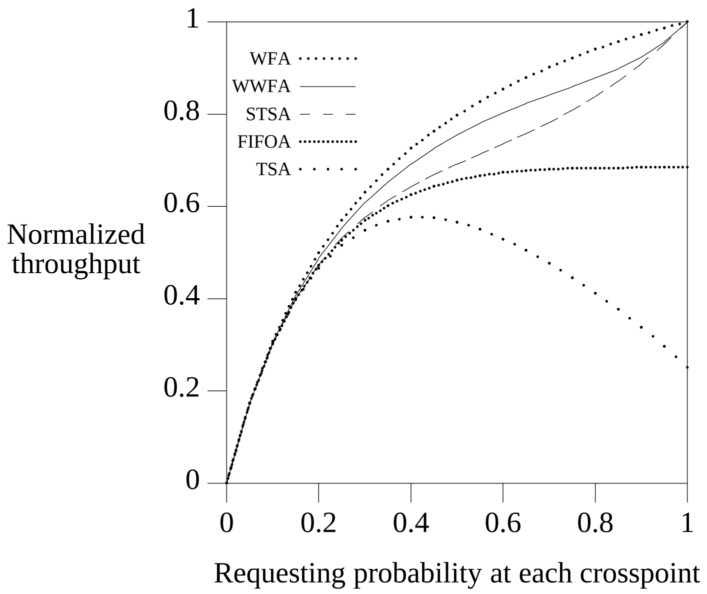



**Figure 3.11:** Probabilistic analysis of throughput for 4×4 switches. A 4×4 switch with WFA, WWFA, STSA, FIFOA, and TSA arbitration schemes.

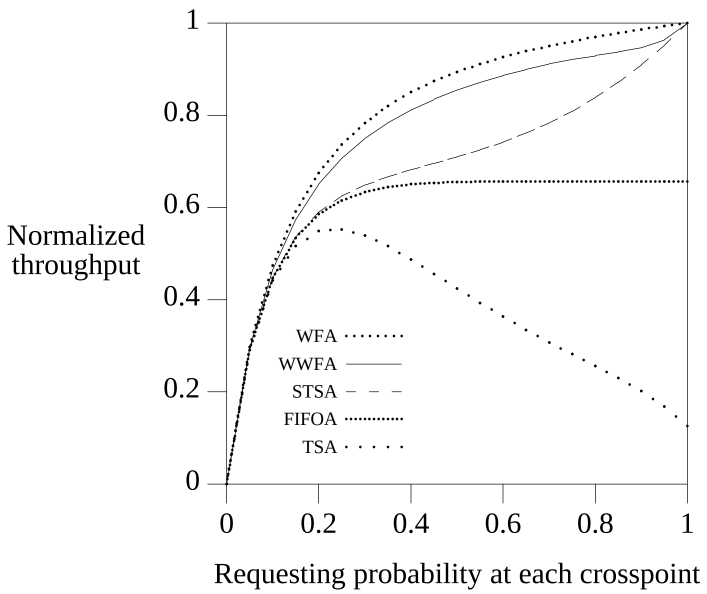



**Figure 3.12:** Probabilistic analysis of throughput for 8×8 switches. An 8×8 switch with WFA, WWFA, STSA, FIFOA, and TSA arbitration schemes.

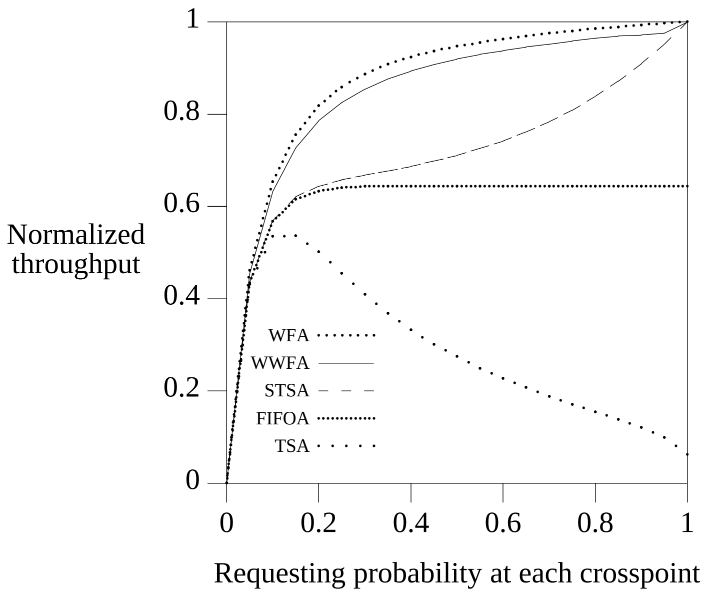



**Figure 3.13:** Probabilistic analysis of throughput for 16×16 switches. A 16×16 switch with WFA, WWFA, STSA, FIFOA, and TSA arbitration schemes.

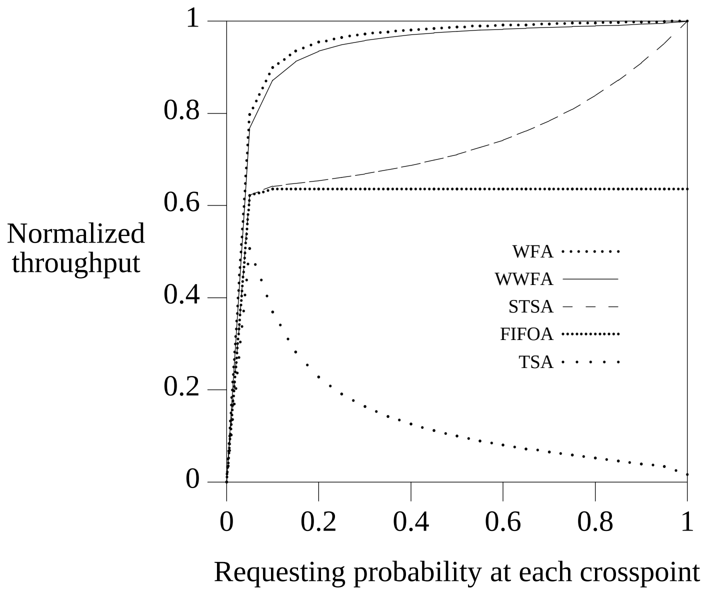



**Figure 3.14:** Probabilistic analysis of throughput for 64×64 switches. A 64×64 switch with WFA, WWFA, STSA, FIFOA, and TSA arbitration schemes.

### 3.3.3. Simulation of Synchronous Networks

The probabilistic analysis used in Section 3.3.1 and Section 3.3.2 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 [Swop86], that allows detailed simulation of arbitrary objects interacting via messages.

#### 3.3.3.1. Performance Measurement

Our simulations focus on a 64×64 Omega network [Lawr75] which uses *blocking* switches. This multistage buffered network operates synchronously. The minimum delay per stage is called a *stage cycle* [Yoon90]. 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.

Ideally, before implementing a system, the performance of the different arbitration schemes should be studied using traces from actual applications. Such traces may include nonuniform traffic patterns and variable length packets. The simulations for synchronous networks in this chapter 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 nodes, 3) packet destinations are uniformly distributed over all the network outputs. Two of the performance measures used are the average latency and the normalized network throughput. The latency is the number of stage cycles from the creation of a packet to its delivery at the destination. Hence, in a three stage network, the minimum latency is three stage cycles. The normalized throughput is the average number of packets received by each output of the network per stage cycle.

Another important measure of the ‘‘quality’’ of arbitration is *fairness*. Ideally, all packets receive fair (equal) treatment so that the latency for all packets should be the same. In reality, some packets are forwarded through the network faster than others. A possible measure of fairness is the difference between the maximum and minimum latency through the network. For all arbitration schemes, the minimum latency is bounded by the speed of the hardware. Hence, the maximum (worst case) latency can be used as a measure of fairness. In simulations, the maximum latency is a poor measure since it is susceptible to statistical anomalies of the simulation. In particular, the simulated maximum latency is sensitive to the initial random number seeds and increases as the number of packets transmitted during the simulation is increased. As a measure of fairness that is less susceptible to statistical anomalies of the simulation, we use the ‘‘$99^{th}$ percentile latency.’’ Considering all the packets transmitted through the network during a simulation run, the $99^{th}$ percentile latency is the minimum of the latencies of the 1% of the packets that received the poorest service (longest latencies) from the network. Hence, 99% of the packets were transmitted through the network faster than the $99^{th}$ percentile latency. Ideally, the $99^{th}$ 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 $99^{th}$ percentile latency may be several times larger than the average latency (see Section 3.3.3.3). Any lack of fairness in resolving conflicts local to the switches will result in increasing the 99$^{th}$ latency and increasing the *difference* between the 99$^{th}$ percentile and average latencies.

We report the results from simulating a single switch as well as from simulating the entire 64×64 network. In all cases we are interested in the steady-state performance. Hence, the data collected during the start-up phase of the simulation was discarded [Mitr82]. In order to determine the duration of the start-up phase, the simulator produced the performance measures (throughput, average latency, and 99$^{th}$ percentile latency) for ‘‘sub-runs’’ of the simulation. Specifically, for the network simulations, for every 1,000 packets that left the network, the performance measures for those 1000 packets were produced. For the single switch simulations intervals of 100 packets were used. The end of the start-up phase is detected when the interval performance measures stop moving in one direction and start oscillating [Mitr82]. In all the simulations, the performance measures started to oscillate after at most five intervals. Hence, for the network simulations the start-up phase lasts less than 5,000 packets and for the single switch simulation it lasts less than 500 packets. In our network simulations the first 32,000 packets were discarded while for the single switch simulations the first 4,000 packets were discarded. Hence, our results reflect steady-state performance.

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. 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, at least eight runs were performed with different random number seeds.

The throughput and latency numbers used are the averages of results from these multiple runs [Welc83]. Each curve is based on twelve points.

For each point used in the curves presented, we calculated the 95% confidence interval [Welc83]. In all cases the confidence interval for the throughput is within 1% of the number reported. For average latencies, the confidence interval is within 3% of the results reported. For the $99^{th}$ percentile latencies, the confidence interval is within 7% of the numbers reported. Out of a total of 432 points used to generate the graphs for $99^{th}$ percentile latencies, there were only 20 points for which the 95% confidence interval was not within 5% of the number reported.

In addition to the six arbitration schemes discussed in Section 3.3.1 (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 [Kuma84]. 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 [Tami88a] are assumed in the evaluation of all seven symmetric crossbar arbitration schemes.

In the following two sections 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 Section 3.3.3.4.

#### 3.3.3.2. Performance Evaluation of a 4×4 Switch

Figure 3.15 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 3.3.1. 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 3.15.

The simulation results for 99$^{th}$ percentile latency are shown in Figure 3.16. TSA results in the worst performance with respect to 99$^{th}$ percentile latency as well as with respect to average latency. SOA results in relatively poor (high) 99$^{th}$ 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 99$^{th}$ percentile latencies even

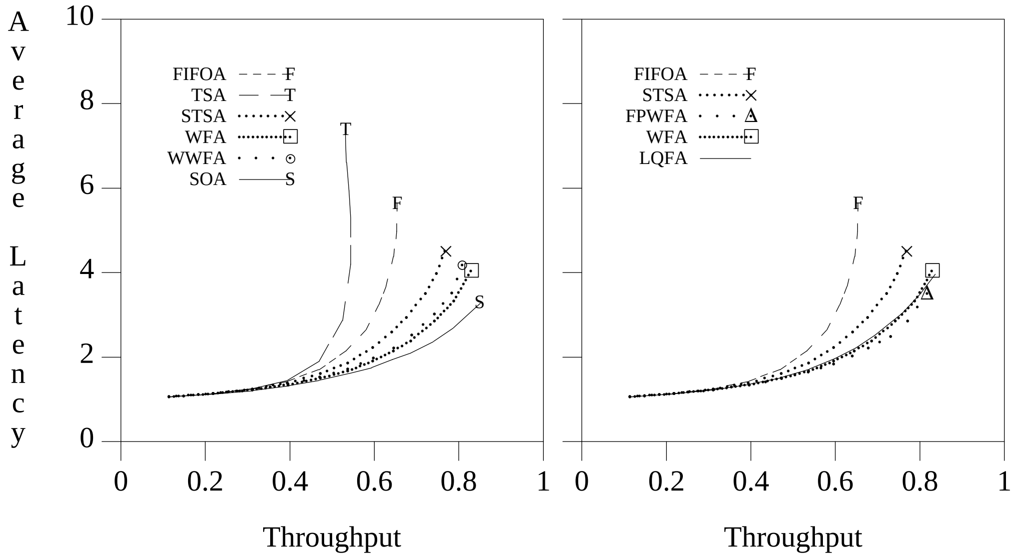



**Figure 3.15:** Average latency vs. normalized throughput of a single 4×4 switch. Four packet slots per input buffer.

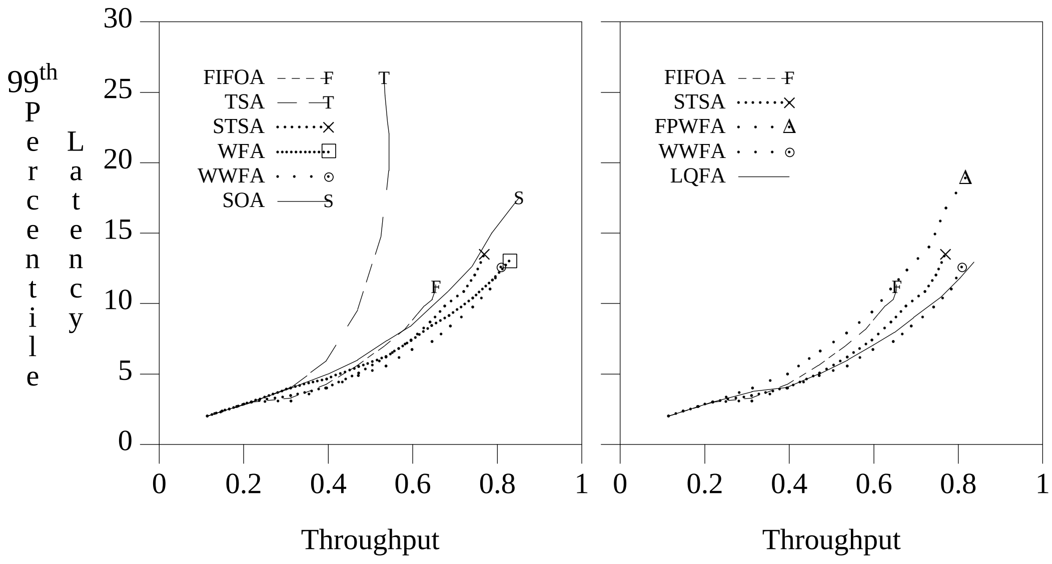



**Figure 3.16:** $99^{th}$ percentile latency vs. normalized throughput of a single 4×4 switch. Four packet slots per input buffer.

though it achieves low average latencies (Figure 3.15). 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.

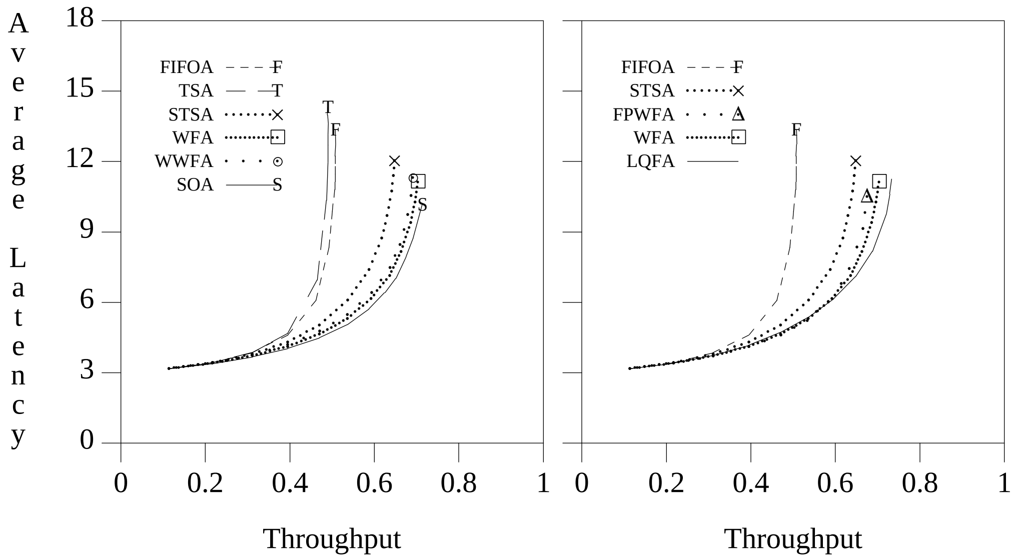



**Figure 3.17:** Average latency vs. normalized throughput of a 64×64 Omega network using 4×4 switches. Four packet slots per input buffer.

#### 3.3.3.3. Performance Evaluation of a 64×64 Omega Network

Figure 3.17 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.

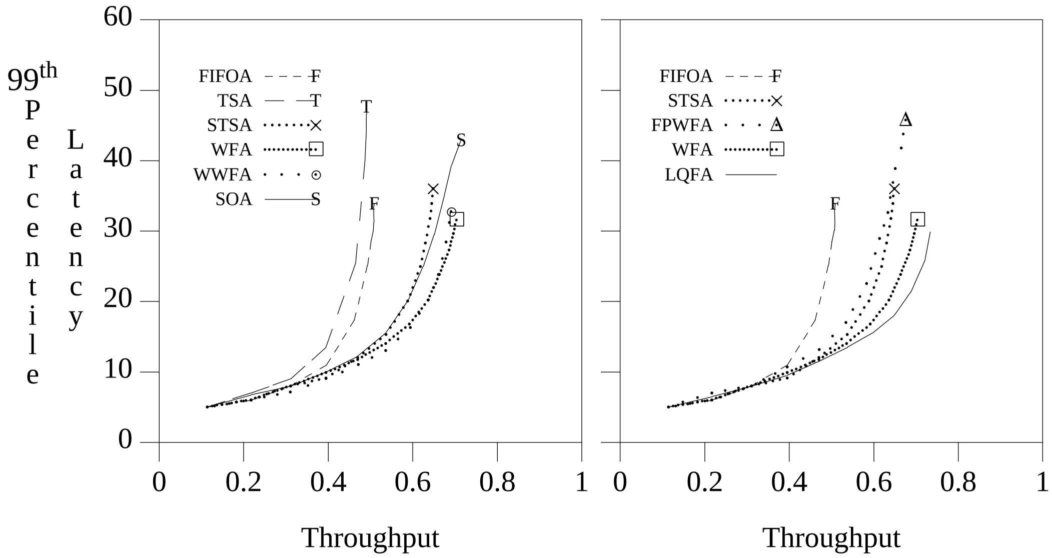



**Figure 3.18:** $99^{th}$ percentile latency vs. throughput of a 64×64 Omega network using 4×4 switches. Four packet slots per input buffer.

The simulation results for $99^{th}$ percentile latency are shown in Figure 3.18. The effects of blocked output ports, discussed above, manifest themselves strongly in these measurement. For single switch simulations (Figure 3.16), the inherent fairness of FIFO resulted in similar $99^{th}$ percentile latencies with 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 $99^{th}$ 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.

#### 3.3.3.4. 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).

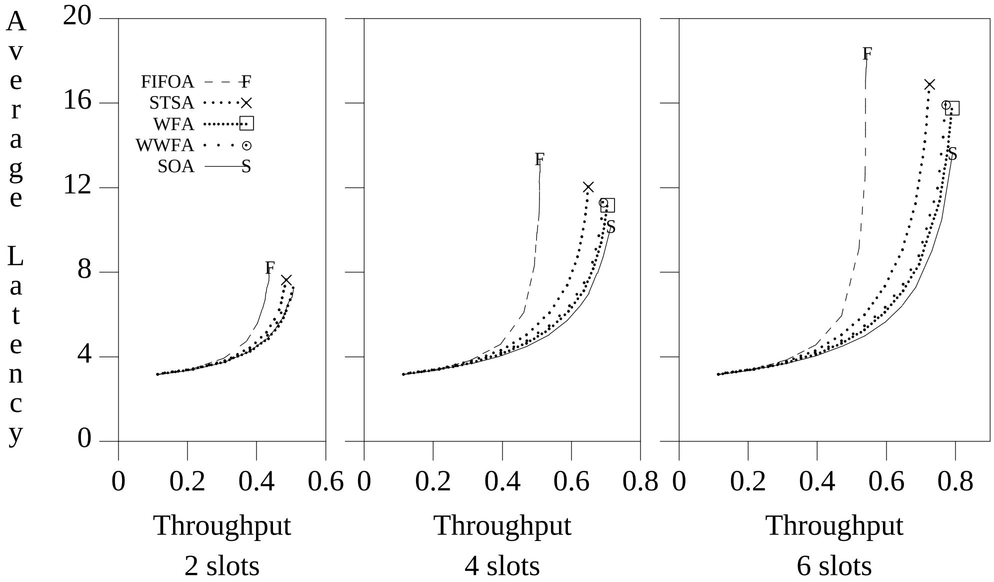



**Figure 3.19:** The impact of input buffer size on average latency. Average latency vs. normalized throughput of 64×64 Omega networks using 4×4 switches with various buffer sizes.

Figure 3.19 shows the average latency versus normalized throughput of 64×64

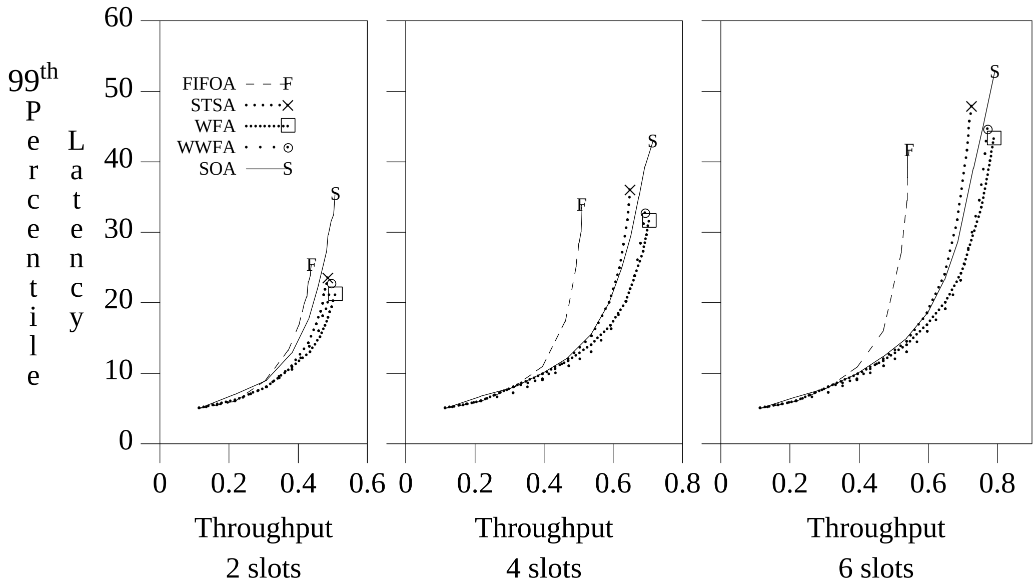



**Figure 3.20:** The impact of input buffer size on 99$^{th}$ percentile latency. 99$^{th}$ percentile latency vs. normalized throughput of 64×64 Omega networks using 4×4 switches with various buffer sizes.

Omega networks of 4×4 switches with two, four, and six packet slots per input buffer. As shown elsewhere [Dias81b, Tami88a], 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 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 3.20 shows the $99^{th}$ percentile latency versus normalized throughput of 64×64 Omega networks of 4×4 switches with two, four, and six packet slots per input buffer. As with the average latencies, the performance advantage of good arbitration schemes increase with increasing buffer size. It is interesting to note that, with respect to $99^{th}$ percentile latency, for small buffers, STSA performs better than SOA, but SOA performs slightly better for large buffers. The reason for the relatively good performance of STSA for small buffers is that STSA is inherently a fair arbitration scheme — each row and each column have equal time at the top priority position. On the other hand, as mentioned earlier, SOA achieves higher maximum throughput at the expense of fairness. For large buffers, the difference in the maximum throughputs achieved by the two schemes becomes

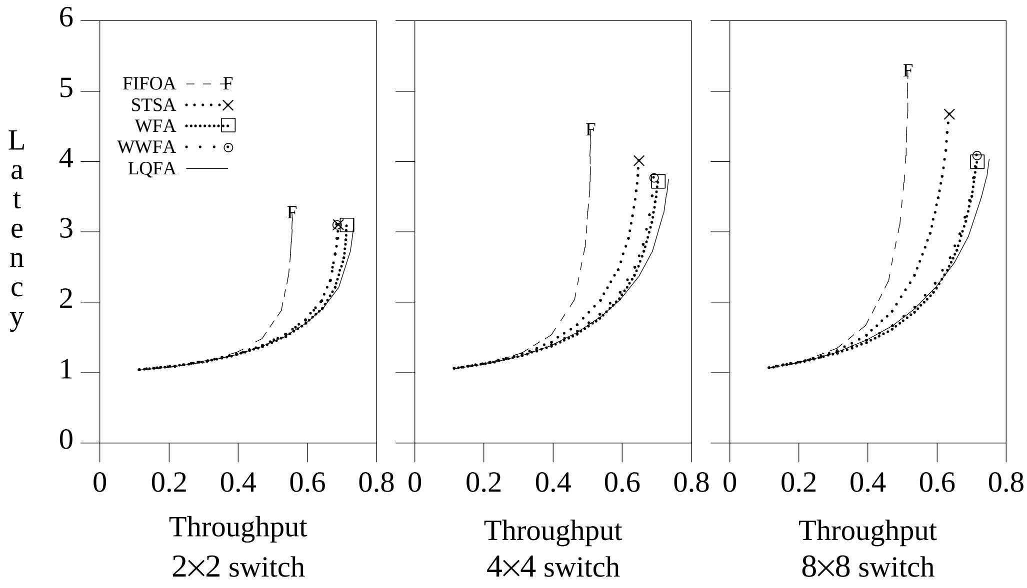



**Figure 3.21:** The impact of switch size. Average latency *per stage* vs. normalized throughput of 64×64 Omega networks with four packet slots per input buffer and various switch sizes.

more pronounced and leads to higher 99th percentile latencies with STSA.

Figure 3.21 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 of the average latency per stage, shown in Figure 3.21, and the number of stages. For FIFOA, as the switch size increases from 2×2 to 4×4, the maximum throughput decreases [Yoon90]. 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.

### 3.3.4. Simulation of Asynchronous Networks

In the simulations presented in Section 3.3.3, we assume that the network operates synchronously. In such synchronous networks, packets are transmitted and received by the communication switches in lock-step. In *asynchronous networks*, it takes multiple clock cycles to transmit an entire packet from one switch to its neighbor. Packets may arrive at a switch during any clock cycle and may be of variable length. In this section, we study the performance impact of the various arbitration schemes in asynchronous networks.

Links and buffers for each switch are assumed to be byte wide. A byte can be read from the buffer, written into the buffer, or transferred through a link in one cycle. Virtual cut-through switching [Kerm79] is used and the minimum delay for a packet going through the communication switch is five cycles [Tami92]. *Senders* generate packets and send them to the network. The interval between packet creation follows a geometric distribution. However, a packet is sent into the network only if the corresponding input buffer has room for the entire packet. Otherwise, the sender blocks.

Figure 3.22 and Figure 3.23 show the performance of a single 4×4 switch with FIFOA, TSA, STSA, WFA, and WWFA in asynchronous networks. The packet size is evenly distributed between 8 and 32 bytes. The buffer size for each input port is 128 bytes. These two figures show that while the symmetric crossbar arbiters significantly outperform the FIFOA, there is no performance difference between the various symmetric crossbar arbiters. Figure 3.24 and Figure 3.25 show the performance of a 64×64 Omega network. Similar to the simulation results for singe switches, the various symmetric crossbar arbiters exhibit very little performance difference. Given a request pattern in a crossbar, a poor arbitration scheme connects fewer crosspoints than a good arbitration scheme, leading to idle resources. However, the arbiter attempts to connect crosspoints for the crossbar continuously in every clock cycle. The idle crosspoints which could otherwise be connected in a good arbitration are likely to be connected in the arbitration in the next clock cycle. Hence, the negative performance impact of a poor arbitration scheme in an asynchronous network is likely to be very small.

Our simulation studies show that even with a small packet size such as 2 bytes, the performance of a poor arbitration scheme is comparable to that of a good arbitration scheme. Figure 3.26 and Figure 3.27 show the performance of a single 4×4 switch with the various arbiters in asynchronous networks. The packet size is 2 bytes and the buffer size for each input port is 10 bytes. Figure 3.28 and Figure 3.29 show the performance of a 64×64 network composed of two stages of 8×8 switches. The packet size is 2 bytes and the buffer size for each input port is

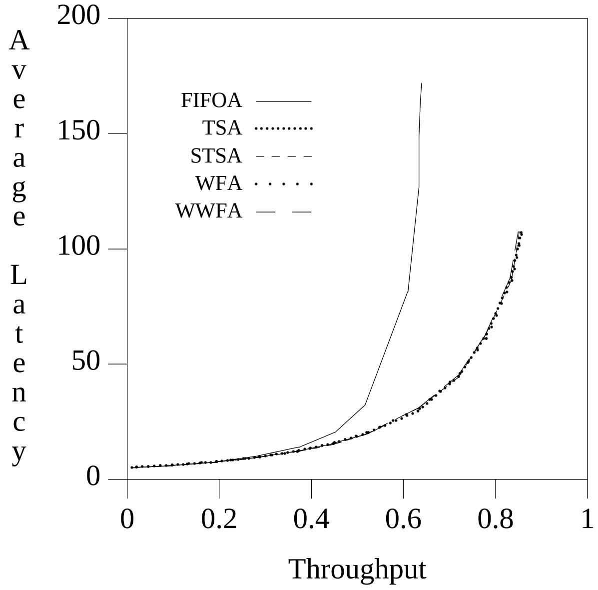



**Figure 3.22:** Performance of a single 4×4 switch in asynchronous networks. Average latency vs. normalized throughput. The packet size is 8-32 bytes. 128 bytes per input buffer.

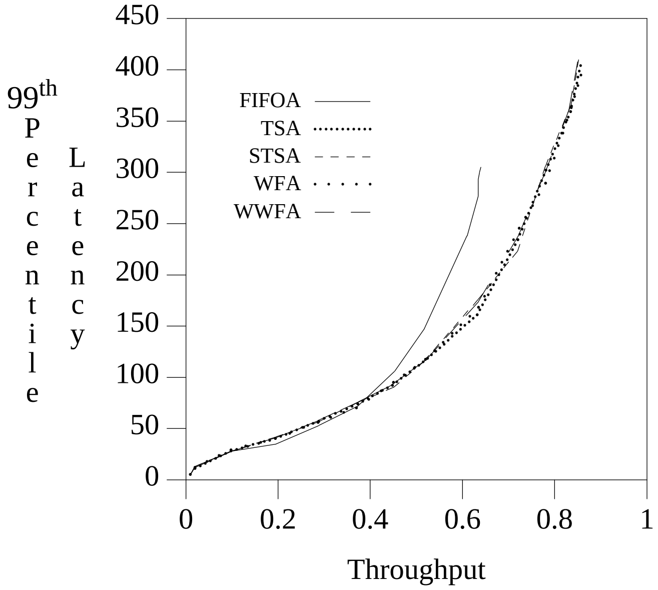



**Figure 3.23:** Performance of a single 4×4 switch in asynchronous networks. $99^{th}$ percentile latency vs. normalized throughput. The packet size is 8-32 bytes. 128 bytes per input buffer.

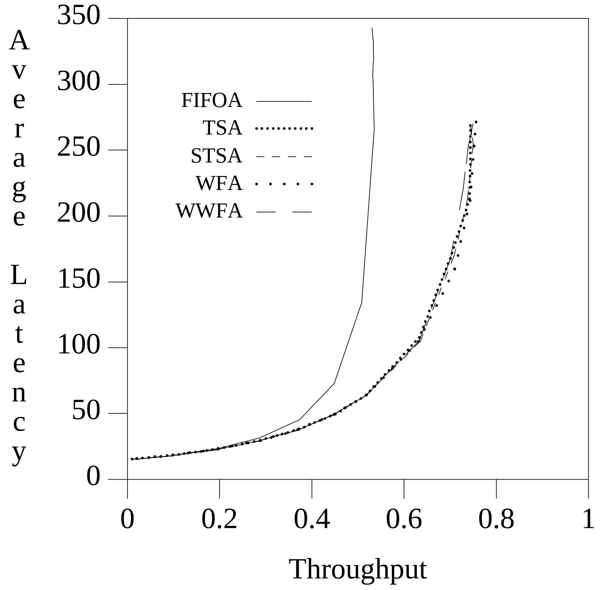



**Figure 3.24:** Performance of an asynchronous 64×64 network. Average latency vs. normalized throughput. The packet size is 8-32 bytes. 128 bytes per input buffer.

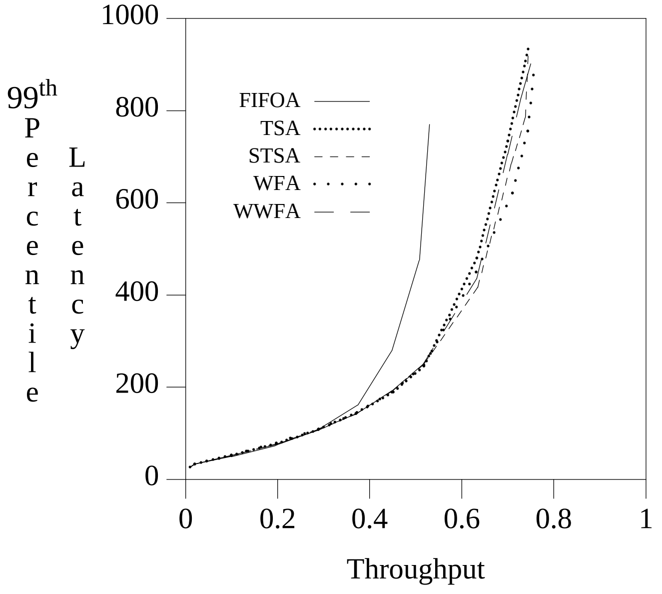



**Figure 3.25:** Performance of an asynchronous 64×64 network. 99th percentile latency vs. normalized throughput. The packet size is 8-32 bytes. 128 bytes per input buffer.

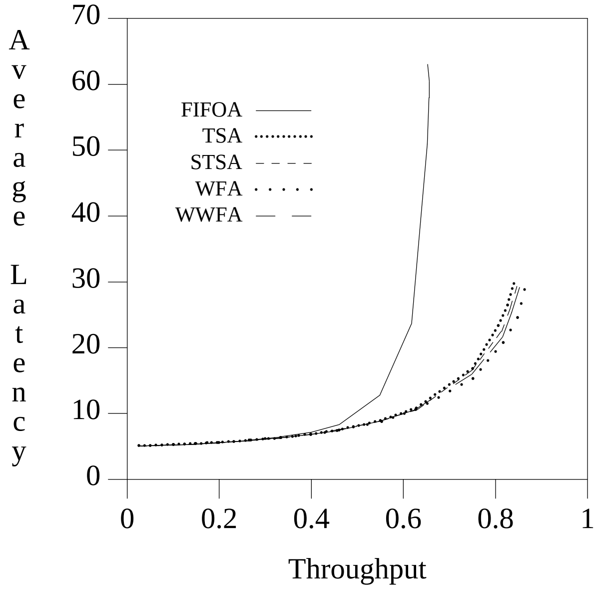



**Figure 3.26:** Performance of a single 4×4 switch in asynchronous networks. Average latency vs. normalized throughput. The packet size is 2 bytes. 10 bytes per input buffer.

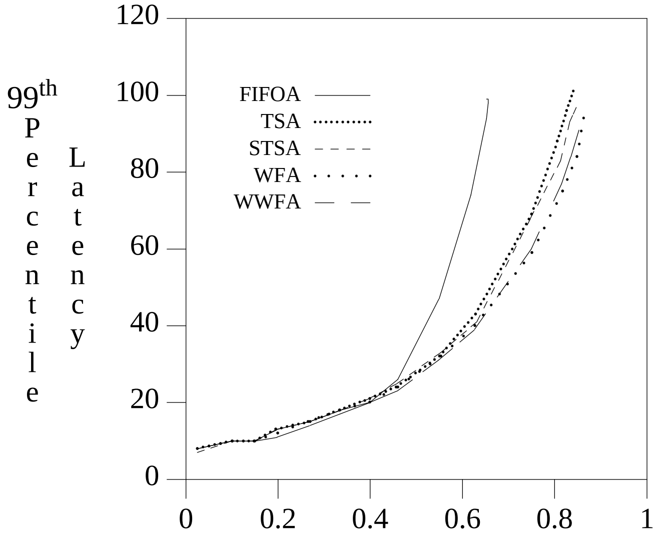



**Figure 3.27:** Performance of a single 4×4 switch in asynchronous networks. 99th percentile latency vs. normalized throughput. The packet size is 2 bytes. 10 bytes per input buffer.

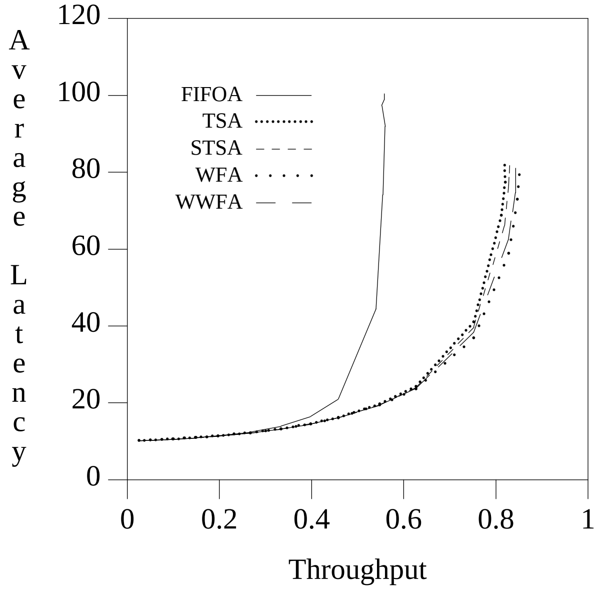



**Figure 3.28:** Performance of an asynchronous 64×64 network. Average latency vs. normalized throughput. The switch size is 8×8. The packet size is 2 bytes. 16 bytes per input buffer.

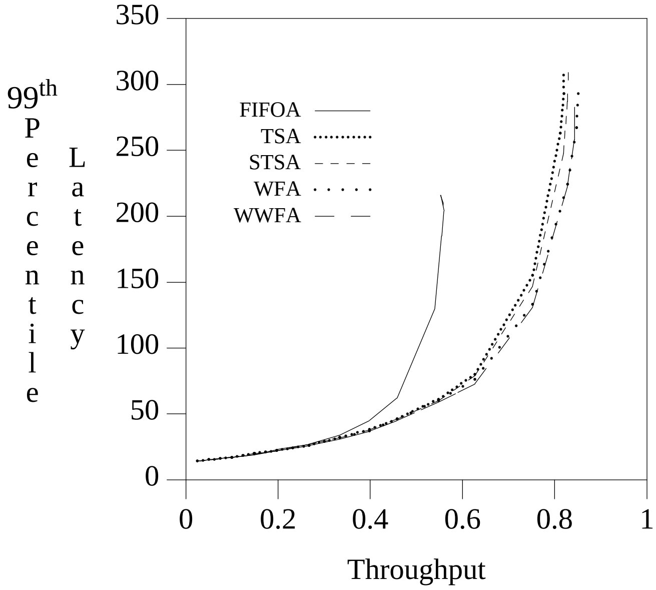



**Figure 3.29:** Performance of an asynchronous 64×64 network. 99th percentile latency vs. normalized throughput. The switch size is 8×8. The packet size is 2 bytes. 16 bytes per input buffer.

16 bytes. These figures show that even if the packet size is very small, the good symmetric crossbar arbiters achieve almost the same performance as the poor symmetric crossbar arbiters. The reason for this is that, again, with a poor arbitration scheme, the idle crosspoints which could otherwise be connected in a good arbitration are likely to be connected in the arbitration in the next clock cycle. Hence, the performance of a poor arbiter is very close to that of a good arbiter.

We also simulated for a 64×64 network composed of two stages of 8×8 switches, where each input of the switches has a large buffer. Figure 3.30 and Figure 3.31 show the simulation results. The packet size is still 2 bytes, but the buffer size is 80 bytes. The performance difference between the various symmetric crossbar arbiters in Figure 3.30 and Figure 3.31 is larger than that in Figure 3.28 and Figure 3.29. The reason for this is that a poor symmetric crossbar arbiter such as TSA connects much fewer crosspoints than a good symmetric crossbar arbiter when almost every crosspoint has a request. As we mentioned in Section 3.2, with TSA the probability that multiple crosspoints in a row win the first arbitration step is increased. Thus, compared to a good arbitration scheme, much fewer crosspoints can be connected due to substantial conflict in the second arbitration step. As a result, it is no longer true that the idle crosspoints which could otherwise be connected in a good arbitration are likely to be connected in the arbitration in the next clock cycle.

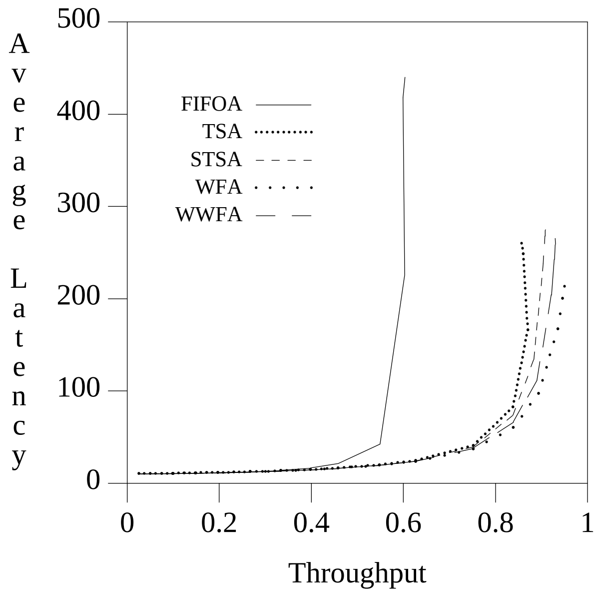



**Figure 3.30:** Performance of an asynchronous 64×64 network. Average latency vs. normalized throughput. The switch size is 8×8. The packet size is 2 bytes. 80 bytes per input buffer.

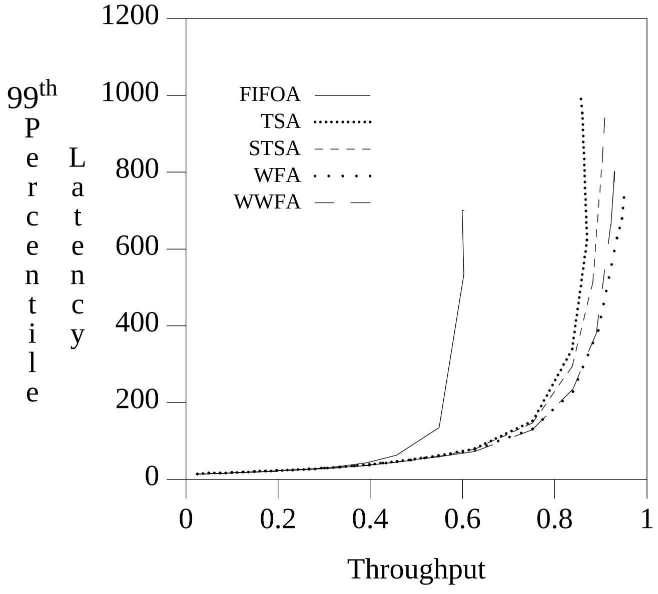



**Figure 3.31:** Performance of an asynchronous 64×64 network. 99th percentile latency vs. normalized throughput. The switch size is 8×8. The packet size is 2 bytes. 80 bytes per input buffer.

## 3.4. Implementation

In Section 3.2 and Section 3.3 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.

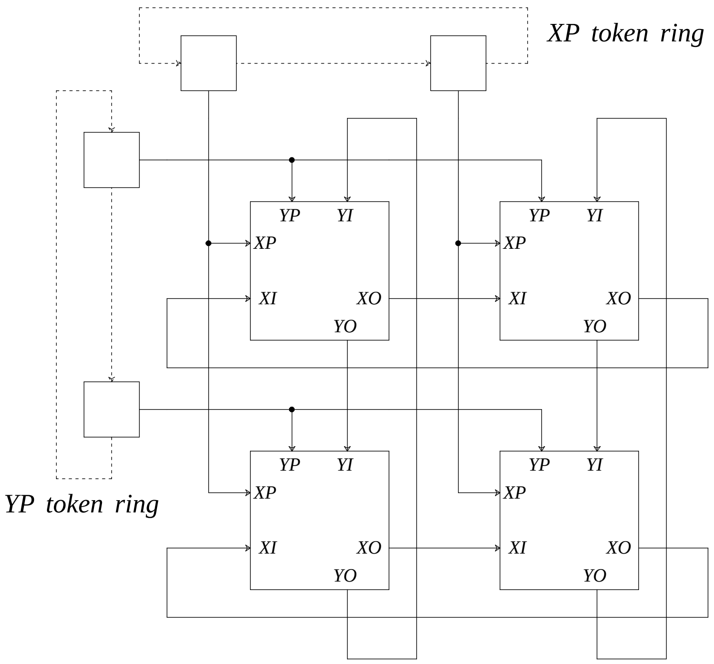



**Figure 3.32:** Organization of a 2×2 wave front arbiter.

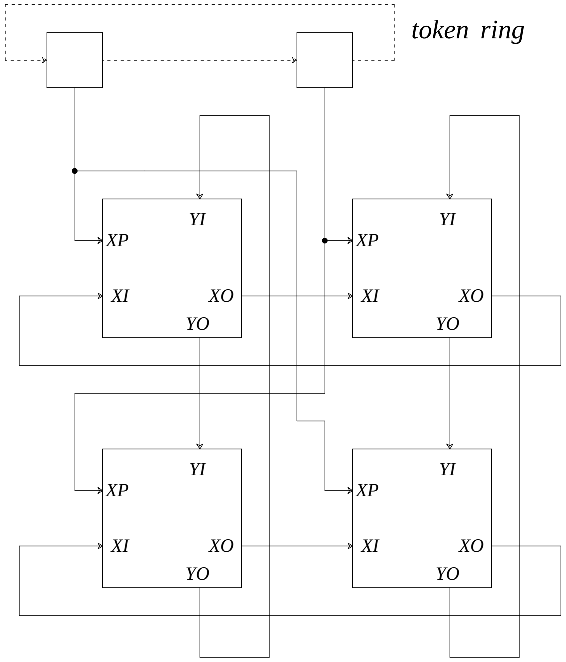



**Figure 3.33:** Organization of a 2×2 wrapped wave front arbiter.

### 3.4.1. Logic and Circuit Design

The structure of a 2×2 WFA and of a 2×2 WWFA arbiter is shown in Figures 3.32 and 3.33, respectively. The same arbitration cell, which is similar to the arbitration cell of Figure 3.5, can be used for both WFA and WWFA. The cell used in Figures 3.32 and 3.33 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 3.5. 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 3.5, but are not shown in Figures 3.32 and 3.33.

A logic diagram of the cells used in Figures 3.32 and 3.33 is shown in Figure 3.34. The $OPB$ signal indicates that the output port is blocked, so there is

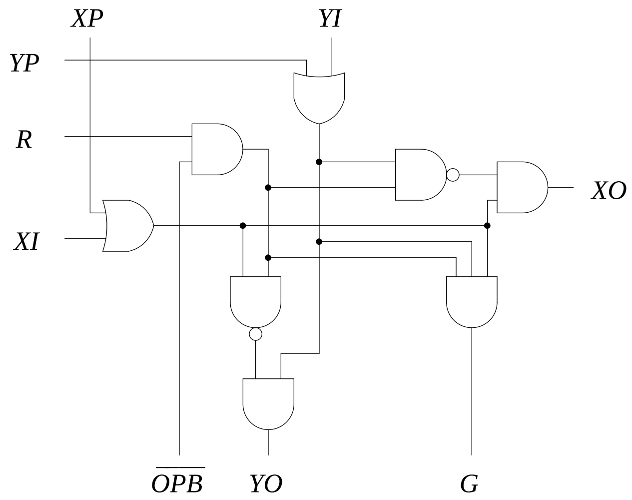



**Figure 3.34:** An arbitration cell. $XP$ and $YP$ indicate top priority in the horizontal and vertical directions, respectively. The $\overline{OPB}$ line is 1 when the output port is *not* blocked.

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 Section 3.2.3. 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 combinational logic is used to implement the cells.

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

$$YO = (YI \vee YP) \wedge \overline{G}$$

$$XO = (XI \vee XP) \wedge \overline{G}$$

As discussed in Section 3.2.3, 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

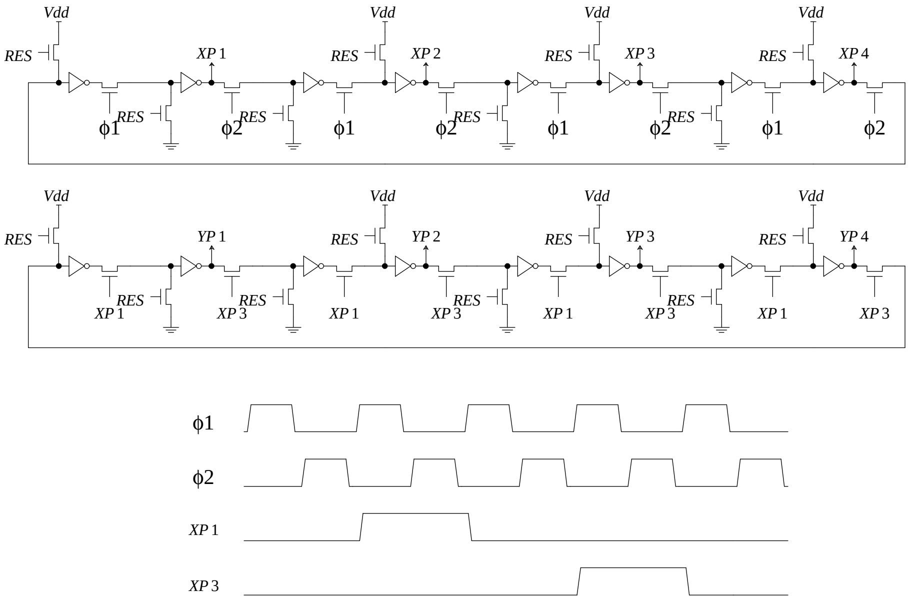



**Figure 3.35:** Horizontal and vertical priority token rings for a wave front arbiter. The reset signal (RES) initializes the two token rings to a valid state.

dynamic shift registers which, on system reset, are initialized so that the first cell is 1 while all the others are 0 (Figure 3.35). For WWFA, only one token ring of 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.

### 3.4.2. 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

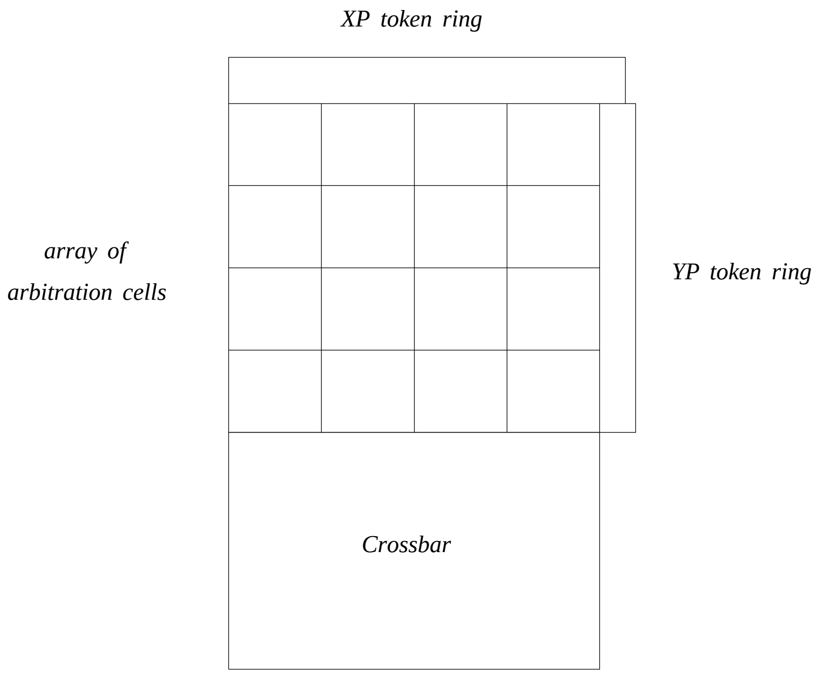



**Figure 3.36:** The floorplan of a 4×4 crossbar of eight-bit wide buses with a wave front arbiter. The modules are drawn to scale.

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 3.36, results in more compact layout.

Figure 3.36 shows the floorplan of the crossbar with the arbiter. The boxes in the figure are drawn to scale with respect to the sizes of the modules in the layout. The crossbar itself is 615 $\lambda$ wide by 376 $\lambda$ tall. The arbiter, with the priority shift registers is 649 $\lambda$ by 527 $\lambda$. The layout of a single arbitration cell is shown in Figure 3.37. It should be noted that the crossbar and the arbiter are small relative to the input buffers. In our layout of a single 96 byte, four queue, DAMQ buffer,

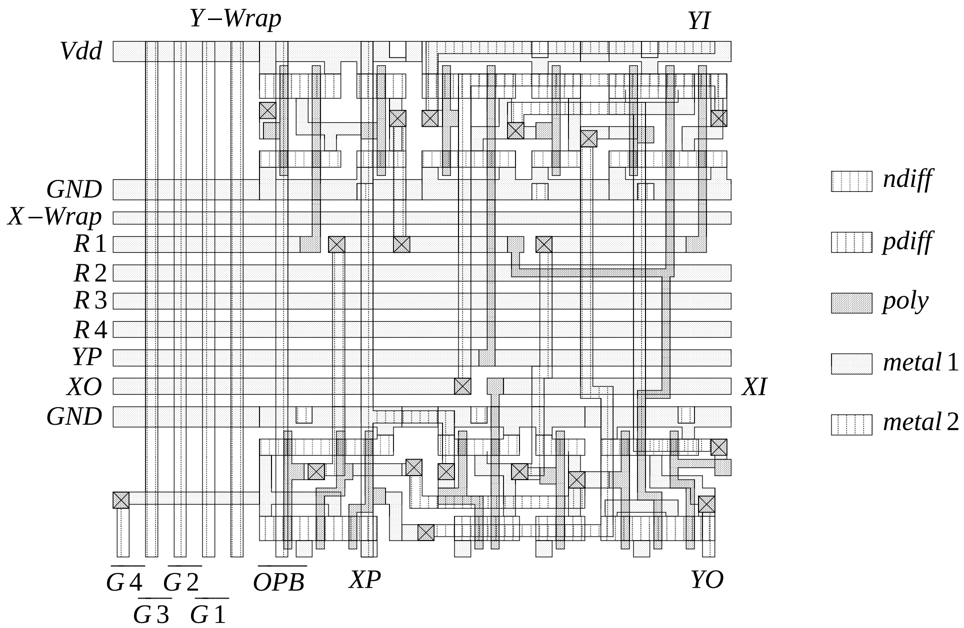



**Figure 3.37:** VLSI layout of a single WFA arbitration cell. This cell is 152 λ wide by 127 λ tall.

the size of the storage is, approximately, 2500 λ × 1000 λ, while the size of the control is 3000 λ × 900λ [Fraz89]. 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 [Fraz89, Tami88a], 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. With the WWFA arbiter, each arbitration can be completed in 10 ns from our circuit simulation.

### 3.5. 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, the arbitration task is *symmetric* with respect to inputs and outputs. Since the result of the arbitration for each crosspoint depends on 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. Our simulations of networks consisting of switches using different types of arbiters provided an evaluation of the *quality of arbitration* of these arbiters. The quality of arbitration of an arbiter is a measure of the ‘‘performance’’ of the arbiter in a network operating in steady state for a long period of time. Specifically, for a network operating in steady state, it is a measure of the average number of requests per unit time that are granted by the arbiter relative to the average number of requests per unit time that would be granted by an ‘‘optimal arbiter.’’ Although these simulations can be used to measure the quality of arbitration of the various arbiters, they did not take into account the fact that there are differences in the speeds of the circuits implementing the different arbiters (see Section 3.2). Depending on the implementation, these differences may impact the latency of packets through the network.

In synchronous networks, a ‘‘good’’ symmetric crossbar arbitration scheme is essential to realizing the potential for performance improvement from multi-queue input buffers. In particular, a ‘‘poor’’ 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. The wave front arbiter and the wrapped wave front arbiter, 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 efficient wave front and wrapped wave front arbiters are amenable to simple regular implementation in VLSI. For a $4 \times 4$ 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 $2\ \mu$ technology, indicates that, for a $4 \times 4$ crossbar, this arbiter can produce a valid and efficient crossbar configuration in 15.5 ns.

For asynchronous networks, the choice among the various symmetric crossbar arbiters has a less impact on performance than for synchronous networks. Just as in synchronous networks, for a given set of requests and port states, different arbiters will allow different connections to be made. For requests not granted, packets are blocked until the next arbitration is performed. For synchronous networks, this delay is a stage cycle. For asynchronous networks, this delay is likely a clock cycle, which corresponds to only a fraction of a stage cycle. The increase of the network latency due to this delay is thus relatively small in asynchronous networks. Hence, the impact of the arbitration scheme on the performance of an asynchronous network is less than that of a synchronous network.

The impact of the arbitration scheme on the performance of an asynchronous network depends on the packet size, switch size, and input buffer size. If the packet size is large and the switch size and input buffer size are small, the number of requests to be arbitrated in a clock cycle is small. For some practical values of these parameters, our simulations show that the quality of arbitration of a ‘‘poor’’ arbiter is approximately the same as that of a ‘‘good’’ arbiter. As the packet size decreases and the switch size and input buffer size increase, the difference in the quality of arbitration between a ‘‘poor’’ arbiter and a ‘‘good’’ arbiter increases since there are more requests to be arbitrated in each clock cycle. However, even if the packet size is small and the switch size and buffer size are large, the difference in the quality of arbitration between a ‘‘good’’ arbiter and a ‘‘poor’’ arbiter is less significant than in synchronous networks.

Three factors should be considered in designing a crossbar arbiter for VLSI communication switches: quality of arbitration, circuit speed, and circuit size. The circuit sizes of the different arbiters considered are approximately the same. The circuit speeds of STSA and WWFA are approximately the same and faster than the other designs. For synchronous networks, either the wave front arbiter or the wrapped wave front arbiter should be considered for their superior quality of arbitration relative to other arbiter designs. The quality of arbitration of the wave front arbiter is slightly better than that of the wrapped wave front arbiter. If using the wave front arbiter rather than the wrapped wave front arbiter does not require a slower clock, the wave front arbiter is the best choice. Otherwise, the wrapped wave front arbiter should be used.

For asynchronous networks, there is a much smaller difference in the quality of arbitration of different arbiters for a wide range of packet sizes, switch sizes, and input buffer sizes. If the packet size is large and the switch size and input buffer size are small, there is no difference which arbiter to use since their quality of arbitration and circuit size are approximately the same. If the packet size is small and the switch size and input buffer are large, either the wave front arbiter or the wrapped wave front arbiter should be used in order to exploit their better quality of arbitration. If the circuit speed of the arbiter has an impact on the network performance, the wrapped wave front should be considered.