Chapter Five
Starvation Prevention

Multi-queue input buffers lead to high performance in multiprocessor interconnection networks by allowing packets at an input port to be processed in non-FIFO order. *Symmetric crossbar arbiters* efficiently resolve conflicting requests in switches with multi-queue input buffers. While these arbiters lead to excellent performance in terms of throughput and average latency, they do not guarantee *fairness*. Hence, it is possible for an ‘‘unlucky’’ packet to be left in a switch buffer for a long time, potentially forever, while other packets are forwarded quickly through the switch.

This chapter describes and evaluates a technique for preventing such *starvation* situations. The viability of the technique is demonstrated by implementing it in VLSI. Simulations are used to evaluate the performance of starvation-free arbiters. Their performance is compared with that of the arbiters without starvation prevention under uniform traffic and nonuniform traffic.

In Section 5.1, the starvation problems with symmetric crossbar arbiters in synchronous networks and asynchronous networks are discussed. The design of starvation-free arbiters for both types of networks are presented. The performance impact of starvation prevention under uniform and nonuniform traffic is evaluated in Section 5.2. Section 5.3 describes the VLSI implementation of a starvation-free arbiter. This starvation-free arbiter is compared with an arbiter without the starvation prevention mechanism to measure the additional cost in terms of circuit delay and chip area.

5.1. Starvation Prevention for Crossbar Arbitration

Without a starvation prevention mechanism, communication switches with multi-queue input buffers cannot guarantee that a packet arriving at the switch will be forwarded to the appropriate output within a finite period of time. This is obvious if the top priority is fixed at a queue (arbitration cell) which is not the one holding the packet in question. Even if the top priority is rotated to a different cell every cycle, eventual transmission of the packet is *not* guaranteed. Specifically, although the queue containing the packet periodically gets the top priority, it is possible that whenever that happens the output port that corresponds to this queue is blocked (e.g., due to a full buffer in the next switch). Hence, low priority queues may be allowed to transmit their packets while the top priority queue remains blocked and priority is then shifted to another queue. Such *starvation* conditions may delay packets in an ‘‘unlucky’’ queue indefinitely. In this section, the starvation problems in synchronous networks and asynchronous networks are described and their solutions are presented respectively.

5.1.1. Starvation Prevention in Synchronous Networks

Starvation prevention in *synchronous* networks is relatively simple [Tami93]. In a synchronous network, packets are transmitted and received by the communication switches in lock-step. The steps are called *stage cycles* [Yoon90]. Assuming no contention, an entire packet is transmitted from one switch to its neighbor in a single stage cycle. No packet is ‘‘in transit’’ across a stage cycle boundary so *all* the resources assigned by the crossbar arbiter — input ports and output ports — are freed at the end of each stage cycle. In such networks, each queue in an $n \times n$ communication switch has the top priority every n^2 stage cycles if a WFA arbiter is used. If the output ports are never blocked, it is guaranteed that no packet will wait in a queue more than $b n^2$ cycles, assuming 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 this unlucky queue remains in the switch indefinitely.

In synchronous networks, the WFA arbiter can guarantee prevention of starvation if a nonempty top priority queue maintains its top priority until it succeeds in sending one packet. We call a WFA arbiter that uses this scheme to rotate the priority, a *round-robin* (RR) arbiter. An RR arbiter can be implemented by not shifting the horizontal XP shift register if there is a top priority request but it is rejected (see Figure 3.32).

With the WWFA arbiter, the starvation prevention mechanism is slightly more complex, because the wrapped diagonal with the top priority corresponds to n queues. It does not solve the starvation problem that we shift the priority when *any* request in the top priority wrapped diagonal is granted. With this scheme, it is possible that the required output port from an unlucky queue is always blocked whenever the top priority visits. On the other hand, it does not solve the starvation problem either that we shift the priority only when *all* requests in the top priority wrapped diagonal are granted. With this scheme, the top priority can stay with the same wrapped diagonal indefinitely if all the requests cannot be granted at the same time.

Figure 5.1 shows a design for achieving starvation prevention for the WWFA

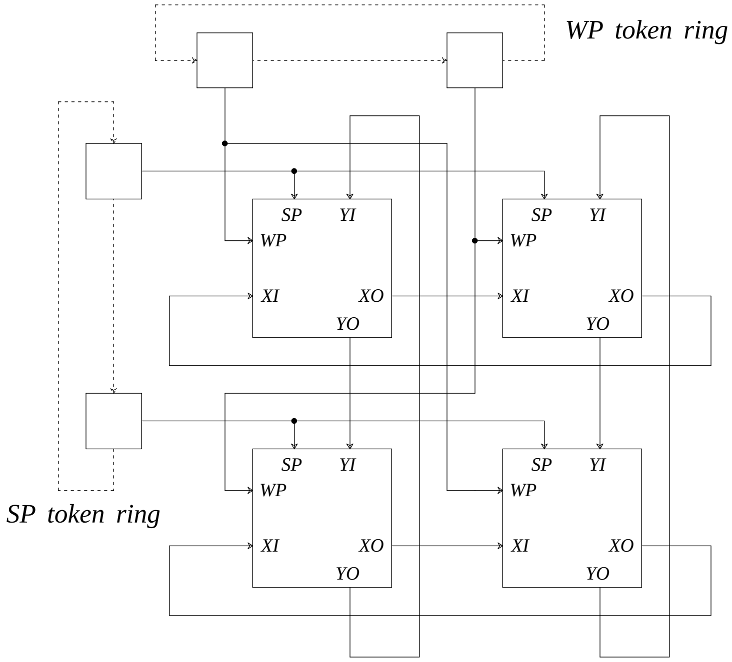

Figure 5.1: Organization of a 2×2 starvation-free wrapped wave front arbiter in synchronous networks.

arbiter in synchronous networks. In addition to the WP token ring, the SP (starvation priority) token ring is used to point to a cell in the top priority wrapped diagonal. The SP token ring is a circular shift register which shifts every time the WP token has traversed the whole WP token ring a round. Each cell in the same wrapped diagonal is pointed to by the SP token ring ‘‘in turn’’ when the top priority visits the wrapped diagonal. Shifting in the WP token ring is decided by the cell pointed by the SP token ring in the top priority wrapped diagonal. If this cell has a request but is rejected due to output port blocking, the WP token ring does not shift until the request is granted. Since any output port cannot be blocked forever, eventually this cell will be granted and the WP token ring will shift. This ensures starvation prevention for each queue.

5.1.2. Starvation Prevention in Asynchronous Networks

Starvation prevention in asynchronous networks is more difficult than in synchronous networks. In asynchronous networks it takes multiple clock cycles to transmit an entire packet from one switch to its neighbor. However, packets may arrive at a switch during any clock cycle and may be of variable length. Thus, different packets utilize input and output ports for transmission for different durations. Once an entire packet is transmitted to a neighbor switch, the crossbar row and column it has been using become available to another packet. Hence, the resources assigned by the crossbar arbiter are not all freed simultaneously so that they can be assigned at will to the top priority queue. It is possible for the top priority queue to be starved indefinitely since the row and column that it needs are never free at the same time.

Figure 5.2 demonstrates a possible scenario of starvation in an asynchronous network with multi-queue buffers. Double squares at a crossbar point indicate that the corresponding queue has a request. Queue (3,2) is starved despite the fact that it maintains the top priority since row 3 and column 2 are never available at the same cycle. When row 3 is released by some other queue in the same row, column 2 is still being used by another queue in the same column. Hence, the arbiter allocates row 3 to another requesting queue in the same row before column 2 is released. Similarly, when column 2 is released by some other queue in the same column, row 3 is still being used by another queue in the same row. Hence, column 2 is allocated to another requesting queue in the same column before row 3 is released.

For an asynchronous network, a starvation-free arbiter can be implemented by

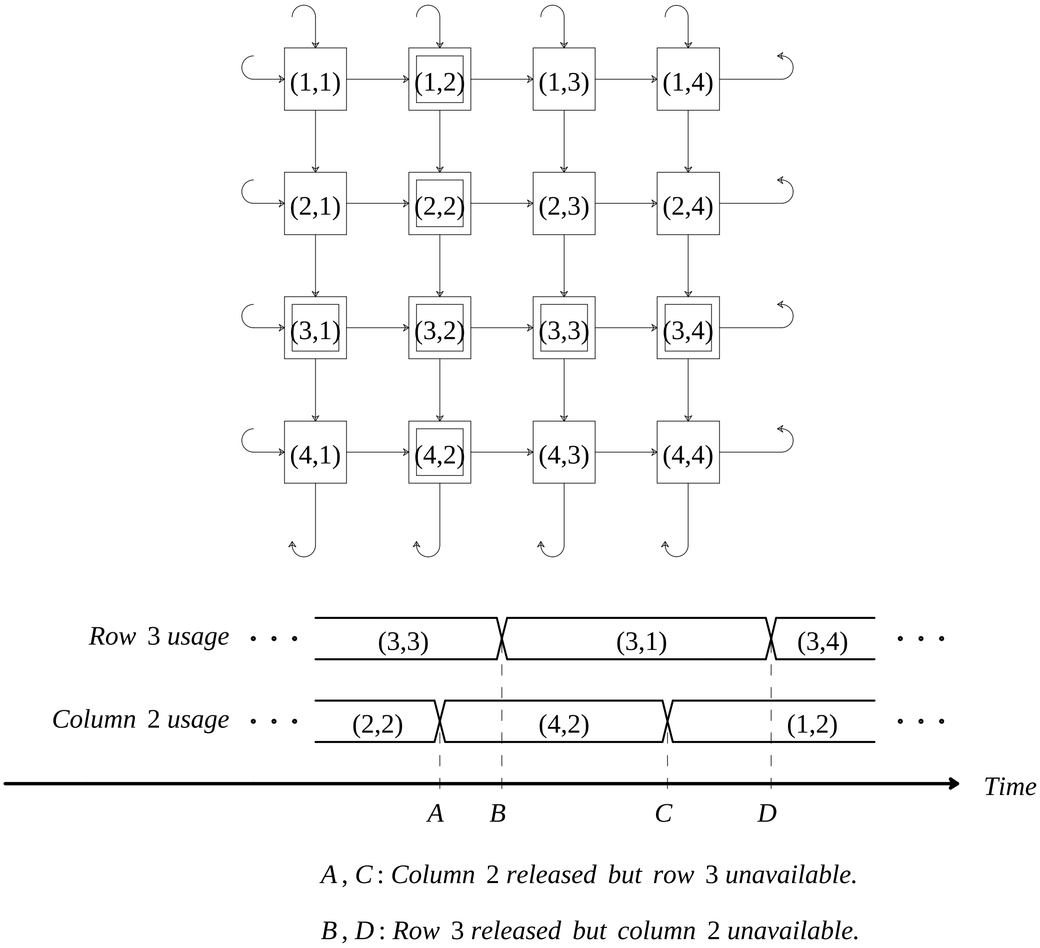

Figure 5.2: An example of starvation for symmetric crossbar arbitration. Double squares at a crossbar point indicate that the corresponding queue has a request. The requesting queue (3,2) keeps being rejected because row 3 and column 2 are occupied by other queues and are not both available at the same cycle for queue (3,2).

modifying the RR arbiter discussed above so that it may *reserve* a resource required by the top priority queue even if the other resource required by this queue is not available. Specifically, in addition to stopping the priority rotation, the top priority queue prevents the assignment of the row and column it needs to other queues until its request is granted. If one resource (either the row or the column) becomes available, the arbiter does not allocate this resource to any other requesting queues. Since the other resource is eventually freed, the request from the top priority queue can be granted then. This starvation-free arbitration scheme is called the *symmetric greedy reservation* (SGR) arbiter. Note that while a row is unavailable only if another queue is using it, a column is unavailable if either another queue is using it or the corresponding output port is currently blocked.

Since some resources are kept idle while reserved, the reservation mechanism for an SGR arbiter may cause performance degradation. To alleviate this problem, a *starvation counter* can be incorporated in the arbiter. The purpose of the starvation counter is to check if the requesting top priority queue is ‘‘near starvation.’’ If the top priority request has been rejected for a certain number of clock cycles (*starvation count threshold*), the arbiter starts to reserve the two required resources. Otherwise, the arbiter operates as an RR arbiter. An SGR arbiter with a starvation count threshold equal to k is denoted as an SGR-k arbiter.

Although both the row and the column are needed for the top priority queue to be granted, reserving one resource is sufficient for resolving the starvation problem. A *row-greedy reservation* (RGR) arbiter is similar to an SGR arbiter except that only the row can be reserved. A column is never reserved and whenever it is freed it is assigned to a queue that can use it immediately. However, a row can be reserved by a top priority queue. After the row has been reserved, the top priority queue obtains and keeps the row the moment it becomes available. Eventually the column required by the top priority queue becomes available, and then the requesting top priority queue can obtain both the row and the column. Likewise, we can reserve the column only. This arbiter is called a *column-greedy* *reservation* (CGR) arbiter.

5.2. Performance Evaluation

We have evaluated the performance of the starvation-free arbiters under uniform traffic and nonuniform traffic for single switches and multistage interconnection networks using the event-driven simulator described in Section 3.3.4. The network is an asynchronous network. Links and buffers are assumed to be byte wide. Each buffer has independent read and write ports and can store 128 bytes. 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]. The simulations use packet sizes which are uniformly distributed between 8 and 32 bytes. *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.

The latency of a packet is the number of cycles that elapse from when the first byte of a packet is generated in the sender to when it leaves the network, including the wait for the blocked network input. The normalized throughput is the average number of bytes received by each network output per clock cycle. Each simulation run is terminated after at least 50,000 packets have arrived at their destination outputs. Statistics were gathered only after 10,000 packets in order to remove start-up effects [Mitr82].

The starvation-free arbiters are compared to two arbiters that do not provide guaranteed packet delivery. One of these two is the round-robin (RR) arbiter, discussed earlier. The other is the *oblivious round-robin* (ORR) arbiter, which is a WFA arbiter that shifts the priority every cycle. Our simulation studies indicate that the arbitration schemes based on the WFA arbiter and those based on the WWFA arbiter achieve approximately the same performance. Hence, only the results for the arbitration schemes based on the WFA arbiter are shown in the evaluation.

5.2.1. Uniform Traffic

Under *uniform traffic* load, packet destinations are uniformly distributed over all the network outputs. Figure 5.3 shows how different arbiters impact the performance of a 4×4 switch under uniform traffic, in terms of average latency vs. normalized throughput. The arbiters compared include the ORR arbiter, the RR arbiter, and the SGR arbiters with starvation count threshold equal to 32, 8, and 0. The results indicate that the reservation mechanism used by the SGR arbiters results in a small performance degradation, showing a lower maximum throughput and a higher average latency. A high starvation count threshold minimizes this performance degradation. Note that an RR arbiter is equivalent to an SGR-∞ arbiter. Figure 5.4 shows the performance impact of these different arbiters on the 99^{th} latency. The simulation indicates that the SGR arbiters also have a higher 99^{th} latency than the ORR and RR arbiters. It appears that under uniform traffic, the reservation mechanism does not improve the 99^{th} latency. Instead, the 99^{th} latency is increased a little.

The comparison of the SGR arbiters and the CGR arbiters is shown in

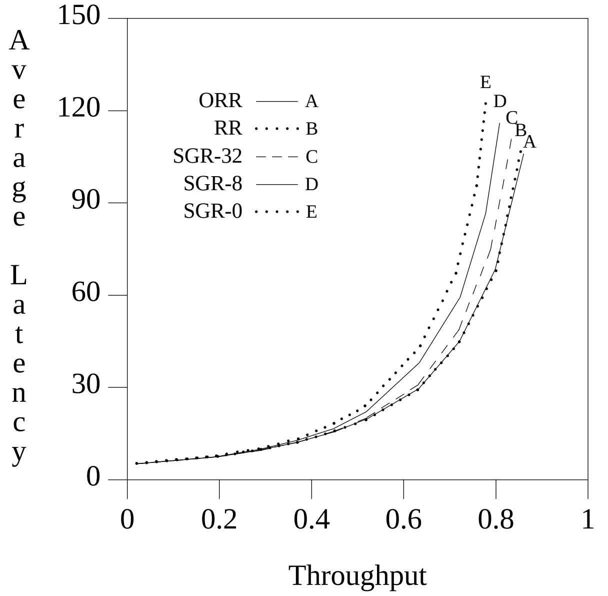

Figure 5.3: Average latency vs. normalized throughput of a 4×4 switch for the different arbiters under uniform traffic.

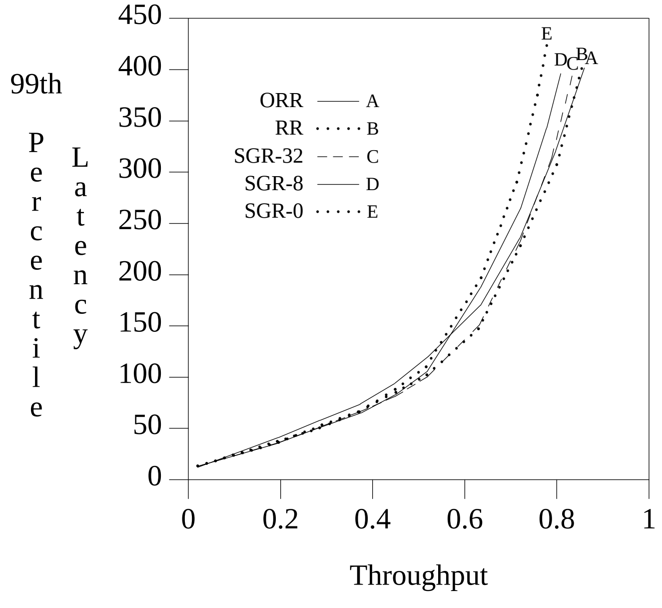

Figure 5.4: 99th percentile latency vs. normalized throughput of a 4×4 switch for the different arbiters under uniform traffic.

Figure 5.5. The RGR-k arbiters are omitted here since our simulation studies show that the performance of an RGR-k arbiter is the same as that of a corresponding CGR-k arbiter. The simulation results in the figure indicate that an SGR-k arbiter performs slightly better than a corresponding CGR-k arbiter. The rest of the chapter focuses on the SGR arbiters.

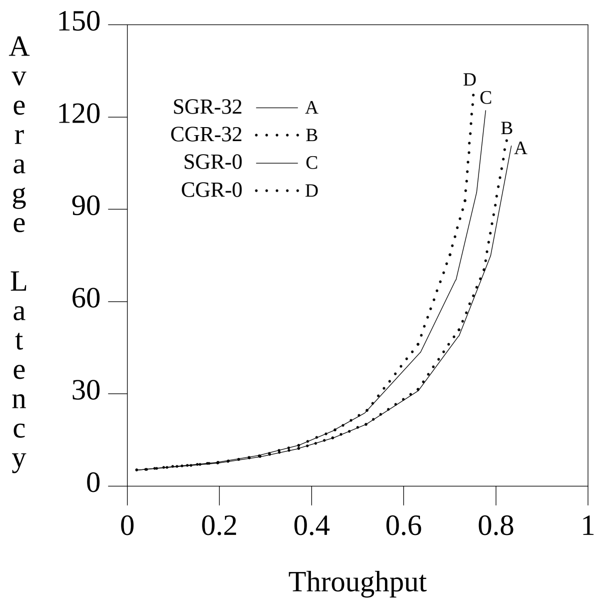

Figure 5.5: Comparison of SGR arbiters and CGR arbiters. Average latency vs. normalized throughput of a 4×4 switch under uniform traffic.

We also consider whether the results of these simulations are different if switch sizes other than 4×4 are used. Figure 5.6 and Figure 5.7 show the simulation results for a 2×2 switch and an 8×8 switch, respectively, under uniform traffic. As shown in the figures, the performance degradation caused by the starvation prevention mechanism is larger for the 2×2 switch and smaller for the 8×8 switch, compared to the 4×4 switch. The reason for this is that when a row or a column is reserved for the top priority queue, the number of resources the arbiter can allocate

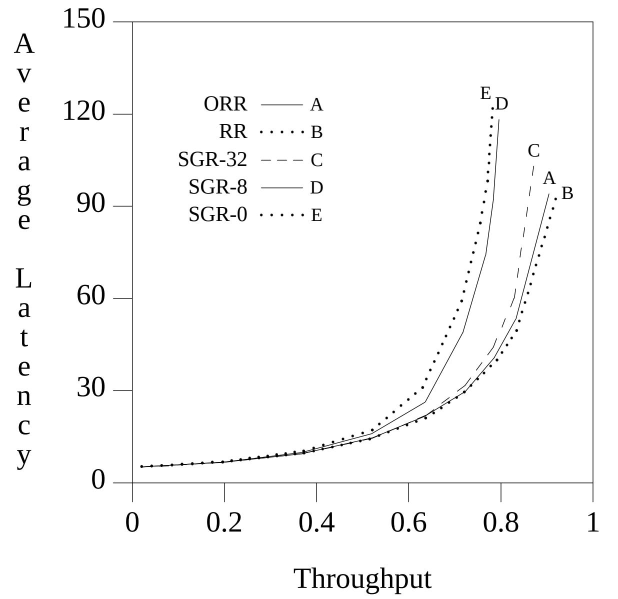

Figure 5.6: Average latency vs. normalized throughput of a 2×2 switch for the different arbiters under uniform traffic.

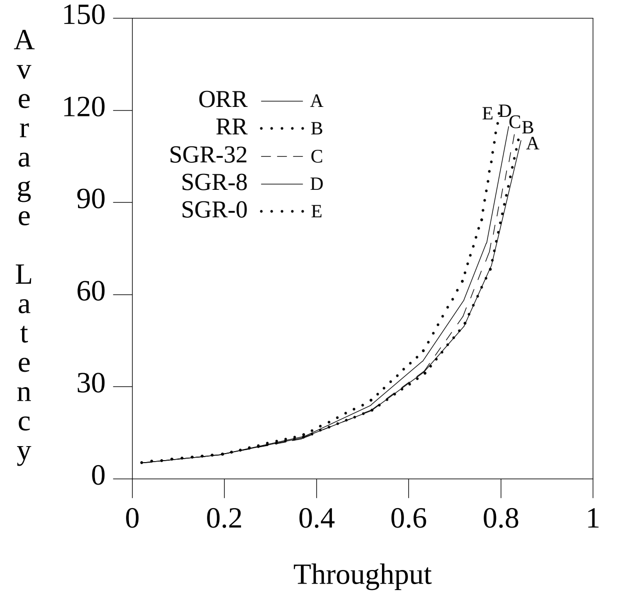

Figure 5.7: Average latency vs. normalized throughput of an 8×8 switch for the different arbiters under uniform traffic.

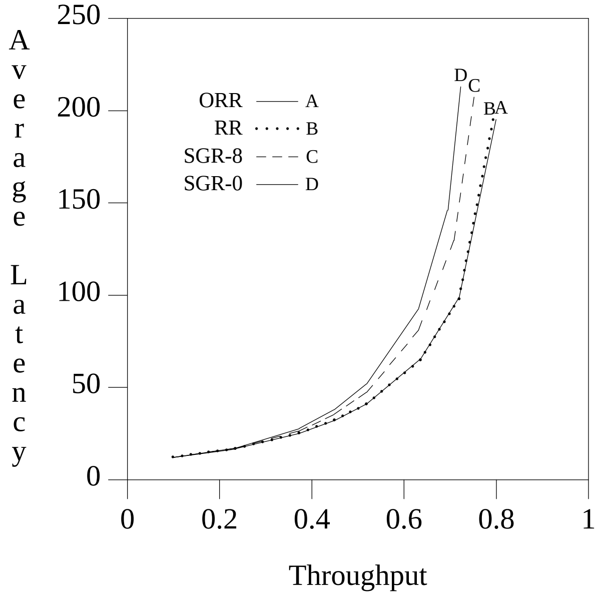

Figure 5.8: Average latency vs. normalized throughput of a 16×16 network for the different arbiters under uniform traffic.

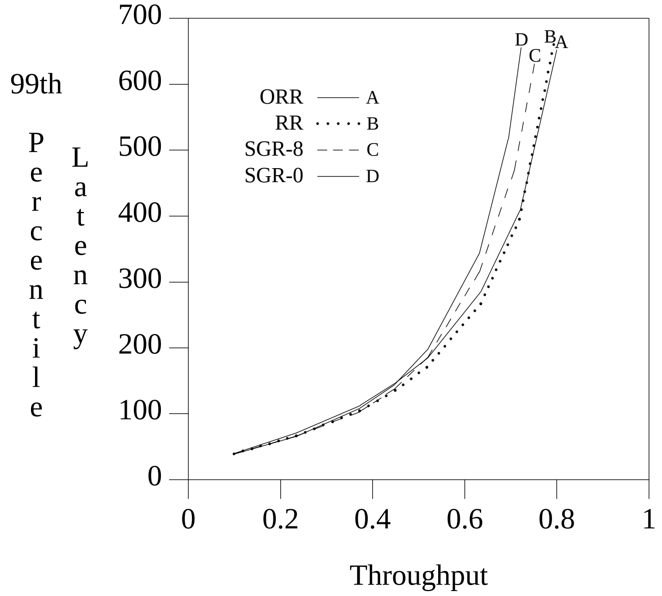

Figure 5.9: 99th percentile latency vs. overall throughput of a 16×16 network for the different arbiters under uniform traffic.

to the other queues is reduced. For example, when a row is reserved for a 4×4 switch, only three rows and four columns can be used. As the switch size increases, the fraction of the reserved resources over the total resources becomes smaller. Specifically, when a resource is reserved, this reserved resource represents a 25%, 12.5%, or 8.25% decrease of the total resources for a 2×2, 4×4, or 8×8 switch, respectively.

Simulation results for a 16×16 network with two stages of 4×4 switches are shown in Figure 5.8 and Figure 5.9. The same conclusions can be derived from the network simulations as from single switch simulations. The SGR arbiters achieve starvation freedom at the cost of a small performance degradation, and this performance degradation decreases as the starvation count threshold increases. Note that under uniform traffic, the reservation mechanism increases the 99th percentile latency a little for the network rather than decreases it, similar to the results for single switches.

5.2.2. Nonuniform Traffic

One of the goals of our simulation studies was to evaluate the performance of the starvation prevention mechanism under nonuniform traffic patterns that could cause some packets to be ‘‘stuck’’ in the switch for an inordinately long time. In the following, several nonuniform traffic patterns are used to study this behavior. We consider a single 4×4 switch in Section 5.2.2.1, and then a 16×16 network with two stages of 4×4 switches in Section 5.2.2.2.

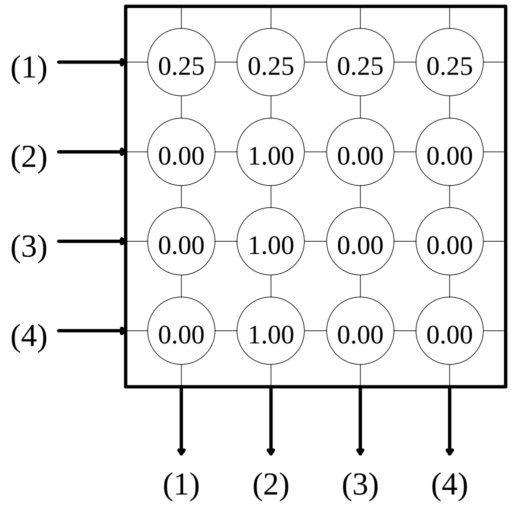

Figure 5.10: A nonuniform traffic pattern for a 4×4 switch. The numbers associated with each crosspoint (or each queue) are the probability distribution of the destinations of the packets in the same buffer for the corresponding output ports.

5.2.2.1. Simulations for a 4×4 Switch

An extreme case of a nonuniform traffic pattern for a 4×4 switch is shown in Figure 5.10. In the figure, the numbers at each crosspoint are the probability distribution of the destinations for packets arriving at the input port. The destinations of the packets arriving at input port 1 are uniformly distributed over the output ports. The packets arriving at input ports 2, 3, and 4 are all destined to a ‘‘hot spot’’ — output port 2. With this traffic pattern, we divide the non-empty queues into three different groups: *unfavored*, *column contenders*, and *row contenders*. The unfavored group consists of only queue (1,2). Queue (1,2) is unfavored because it is the only queue that receives packets and has to compete for *two* resources in order to send these packets. The column contenders group consists of queues (2,1), (3,1), and (4,1). The queues in the column contenders group only compete for an output port (column). The row contenders group consists of queues (1,1), (1,3), and (1,4). The queues in the row contenders group only compete for an input port (row).

To evaluate the performance for different queue groups under nonuniform traffic, we use *overall throughput, queue throughput* and *queue latency* as performance measures. Overall throughput is the average normalized throughput over all the outputs (different outputs can have different normalized throughputs in the nonuniform traffic case). Queue throughput is the average number of bytes transmitted from a defined group of queues. Queue latency is the average latency of the packets which are transmitted from a defined group of queues.

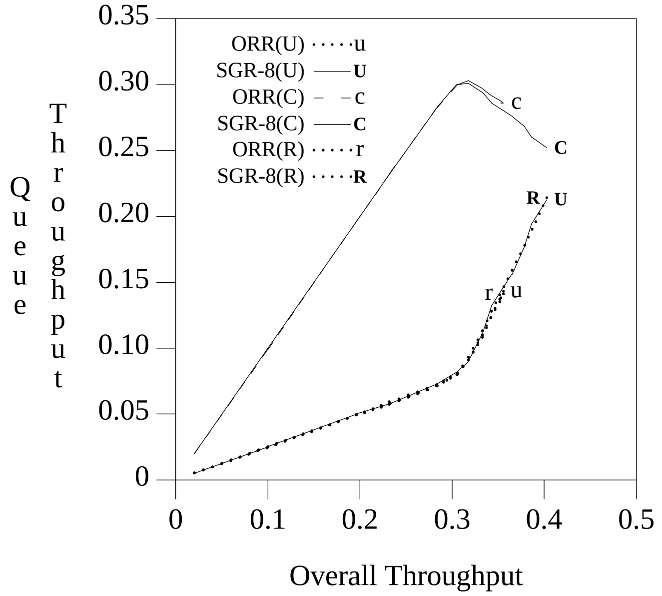

Figure 5.11: The impact of starvation prevention on the queue throughput of different queue groups for the traffic pattern in Figure 5.10. U includes queue (1,2) only; C queues (2,2), (3,2), and (4,2); R queues (1,1), (1,3), and (1,4).

Using the traffic pattern from Figure 5.10, Figure 5.11 shows the performance comparison between the ORR arbiter and the SGR-8 arbiter in terms of queue

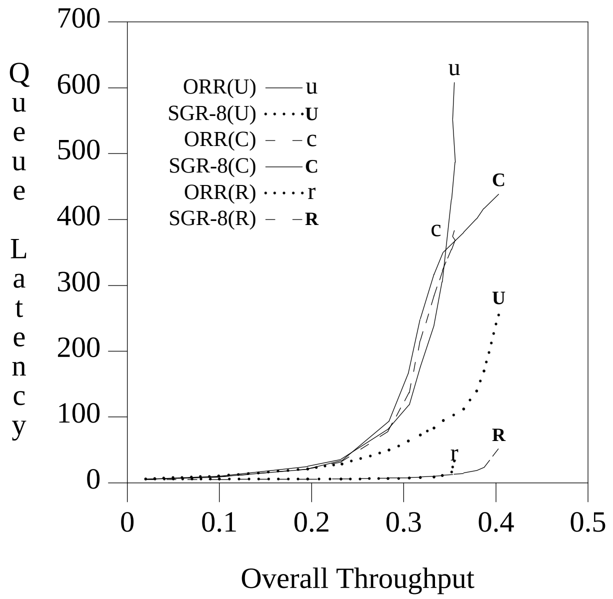

Figure 5.12: The impact of starvation prevention on the queue latency of different queue groups for the traffic pattern in Figure 5.10. U includes queue (1,2) only; C queues (2,2), (3,2), and (4,2); R queues (1,1), (1,3), and (1,4).

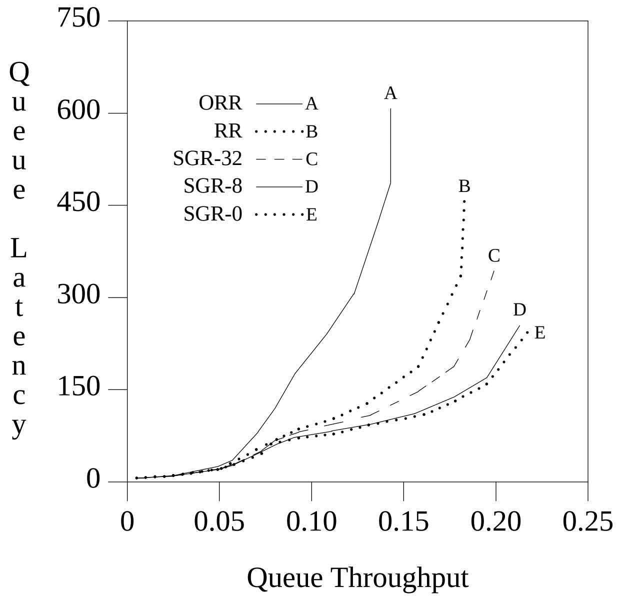

Figure 5.13: The performance impact of different arbiters on queue group U under the traffic pattern in Figure 5.10.

throughput vs. overall throughput for three groups of queues. Figure 5.12 shows the performance comparison in terms of queue latency vs. overall throughput. The SGR-8 achieves a higher maximum overall throughput than the ORR arbiter (0.403 vs. 0.355). Furthermore, the results indicate that the reservation mechanism for a starvation-free arbiter has little impact on groups C and R. However, under high throughput, it significantly reduces the queue latency of queue (1,2).

Figures 5.13, 5.14, and 5.15 show queue latency vs. queue throughput — the latency and throughput shown are only for the specific queue group in question. However, in all three figures the *applied load* from all the senders is varied uniformly to obtain these results.

Figure 5.13 shows the performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 5.10. A significant reduction in latency is achieved by changing the priority shifting scheme from ORR to RR. However, the reservation mechanism of the SGR arbiters leads to additional reductions in latency. As the starvation count threshold is decreased, the queue latency decreases since the reservation mechanism is triggered earlier.

Figure 5.14 shows the performance impact of the different arbiters on queues (1,1), (1,3), and (1,4) under the traffic pattern in Figure 5.10. The ORR arbiter achieves the lowest maximum queue throughput. The reason for this is that with ORR queue (1,2) is more likely to be starved, causing the row 1 input buffer to fill up with packets destined for output 2. In this case, no sender 1 packets destined to the other outputs can enter the switch. The SGR arbiters can achieve a slightly higher queue throughput than the RR arbiter since they are more likely to prevent input buffer 1 from filling up with queue (1,2) packets. This effect increases with

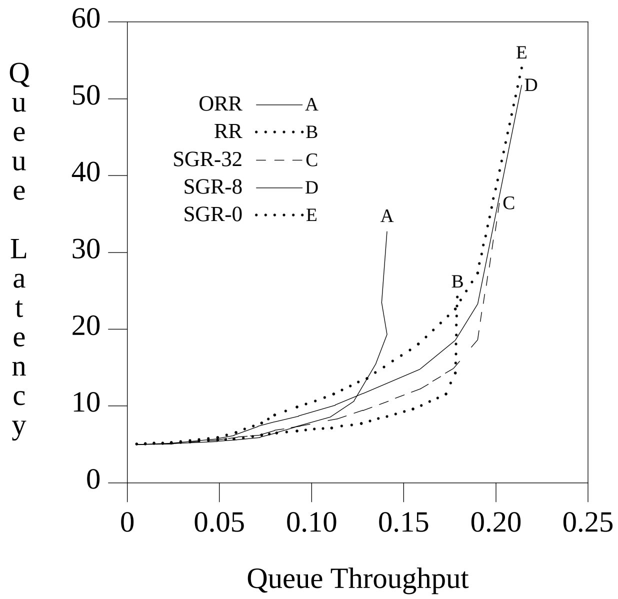

Figure 5.14: The performance impact of different arbiters on queue group R under the traffic pattern in Figure 5.10.

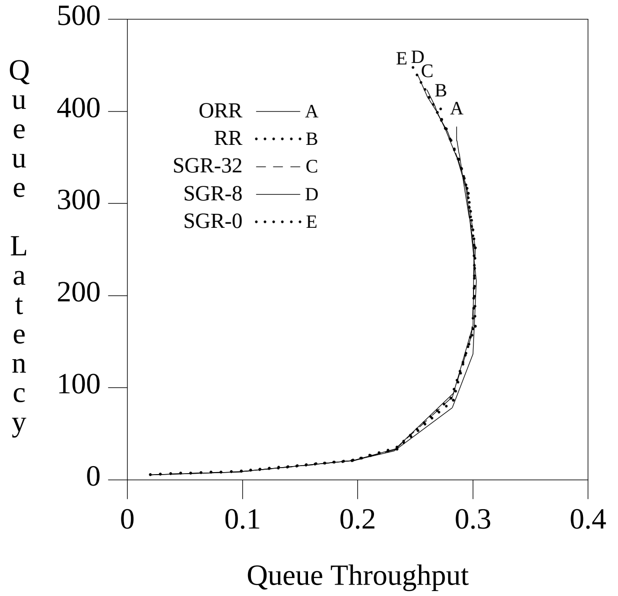

Figure 5.15: The performance impact of different arbiters on queue group C under the traffic pattern in Figure 5.10.

decreasing starvation count threshold since the reservation mechanism is triggered earlier. On the other hand, for a given queue throughput, decreasing the starvation count threshold increases the latency since the reservation mechanism is more likely to be triggered, allowing transmission from queue (1,2) and delaying packets in the other queues of input 1.

Figure 5.15 shows the performance impact of the different arbiters on queues (2,2), (3,2), and (4,2) under the traffic pattern in Figure 5.10. It might be expected that queues (2,2), (3,2), and (4,2) would exhibit poorer performance for starvation-free arbiters due to the reservation of column 2 for queue (1,2). Figure 5.15 shows that the maximum achievable queue throughput is the same for all the arbiters. However, as shown in Figure 5.11, the maximum queue throughput for this group (group C) is not achieved when the overall switch throughput is maximized. Instead, the maximum queue throughput for group C is achieved at a point where the reservation mechanism has no impact on the queue throughput for queue (1,2). As the applied load is increased above this point, the group C throughput actually decreases. This decrease is most pronounced with the SGR-0 arbiter precisely because this arbiter reserves column 2 for queue (1,2) most often.

Figure 5.16 shows average latency vs. overall throughput under the traffic pattern in Figure 5.10 for all the packets, regardless of which inputs they arrive and which outputs they leave. Figure 5.17 shows the results for 99^{th} percentile latency vs. overall throughput. Unlike the results under uniform traffic, the ORR arbiter has the highest average latency and 99^{th} latency when the overall throughput is high. Changing the priority shifting scheme from ORR to RR reduces the average latency and the 99^{th} latency. The reservation mechanism of the SGR arbiters leads

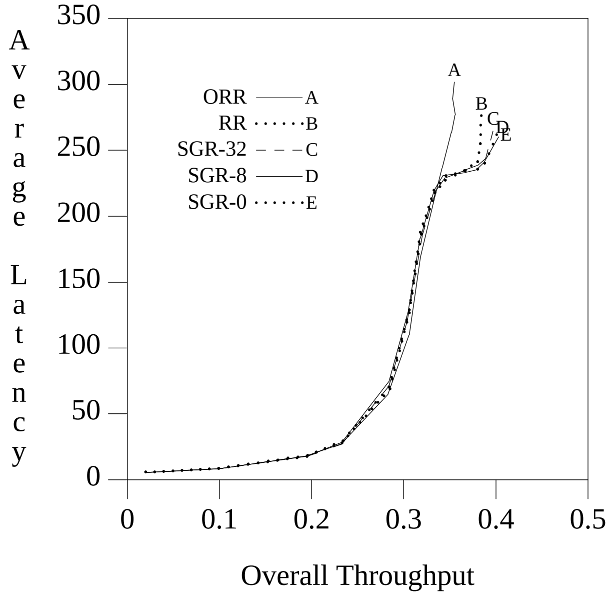

Figure 5.16: Average latency vs. overall throughput of a 4×4 switch for the different arbiters under the traffic pattern in Figure 5.10.

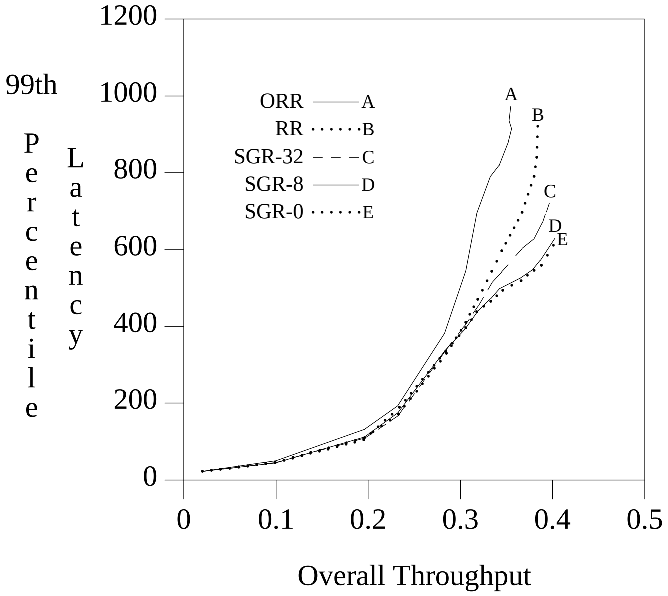

Figure 5.17: 99th percentile latency vs. overall throughput of a 4×4 switch for the different arbiters under the traffic pattern in Figure 5.10.

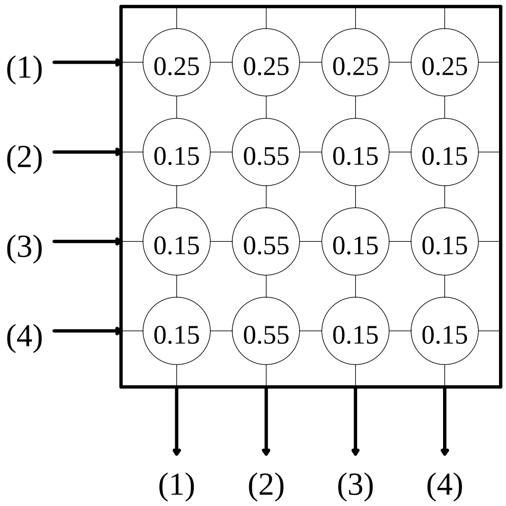

Figure 5.18: A nonuniform traffic pattern for a 4×4 switch. The numbers associated with each crosspoint are the probability distribution of the destinations of the packets in the same buffer for the corresponding output ports.

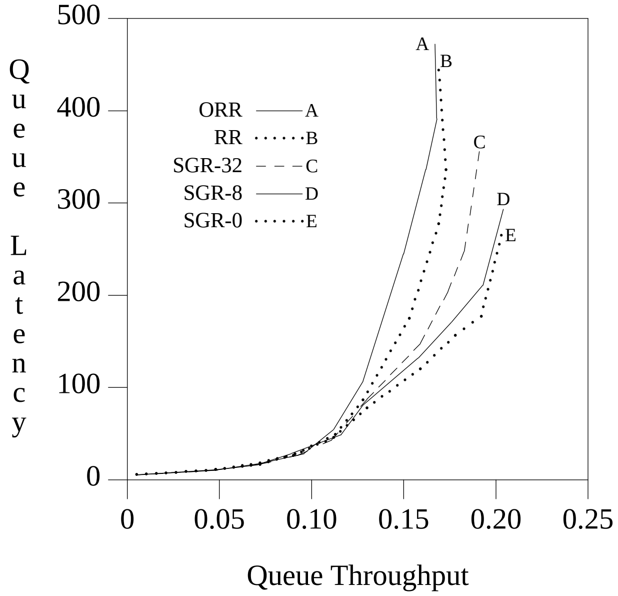

Figure 5.19: The performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 5.18. Queue latency vs. queue throughput.

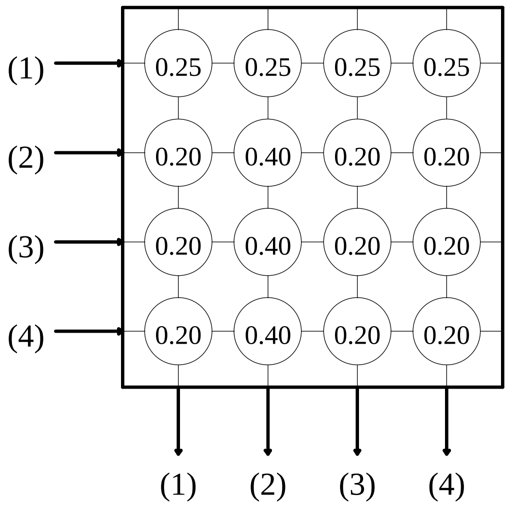

Figure 5.20: A nonuniform traffic pattern for a 4×4 switch. The numbers associated with each crosspoint are the probability distribution of the destinations of the packets in the same buffer for the corresponding output ports.

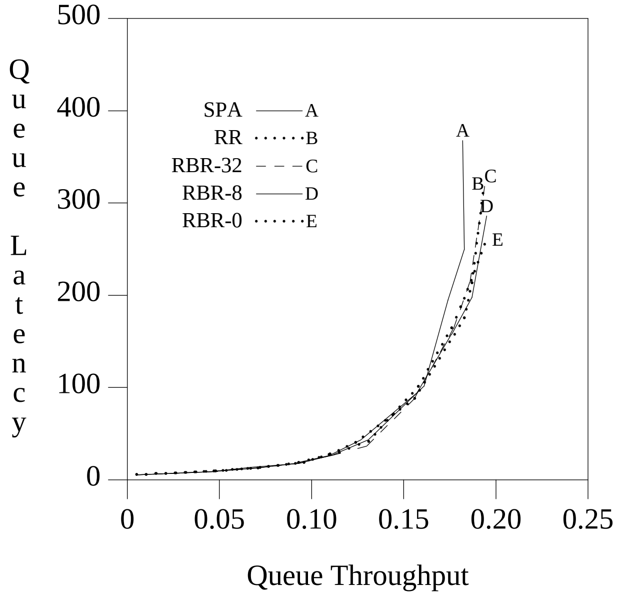

Figure 5.21: The performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 5.20. Queue latency vs. queue throughput.

to further reduction, particularly in 99^{th} latency.

Figure 5.18 shows a nonuniform traffic pattern similar to Figure 5.10 but the packets arriving at input ports 2, 3, and 4 are not all destined to output port 2. Only 55% of these packets are destined to output port 2, and the destinations of the other 45% of the packets arriving at inputs 2, 3, and 4 are evenly distributed over outputs 1, 3, and 4. Figure 5.19 shows the performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 5.18. The simulation results indicate that performance improvement provided by the reservation mechanism for queue (1,2) is still significant.

Figure 5.20 shows another nonuniform traffic pattern where 40% of the packets arriving at inputs 2, 3, and 4 are destined to output port 2, and the destinations of the other 60% are evenly distributed over outputs 1, 3, and 4. Figure 5.21 shows the performance impact of different arbiters on queue (1,2) under the traffic pattern in Figure 5.20. In this case, the performance difference between the different arbiters is very small for queue (1,2). This indicates that as the percentage of the packets destined to the hot spot from input ports 2, 3, and 4 decreases, the latency for queue (1,2) decreases since it is easier for queue (1,2) to obtain column 2.

5.2.2.2. Simulations for a 16×16 Network

Figure 5.22 shows a nonuniform traffic pattern for a 16×16 network. The network consists of two stages of 4×4 switches. In this traffic pattern, 80% of the packets generated at senders 4, 8, and 12 are destined to a hot spot — receiver 1. The destinations of the other 20% of the packets are uniformly distributed over all

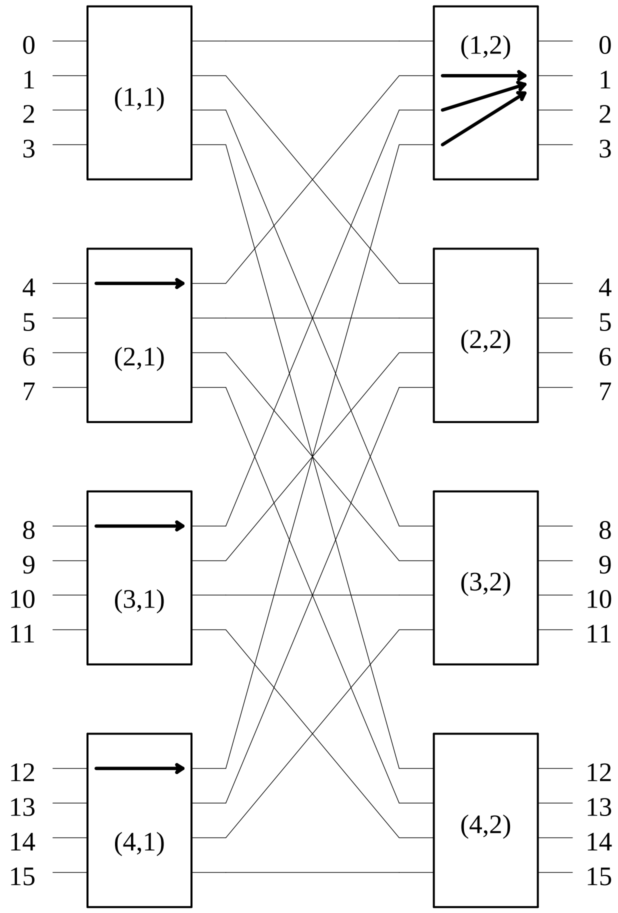

Figure 5.22: A nonuniform traffic pattern for a 16×16 network using 4×4 switches. 80% of the packets generated at senders 4, 8, and 12 are destined to receiver 1. The destinations of the other 20% of the packets are uniformly distributed over all the network outputs. All the packets generated at the senders other than 4, 8, and 12 are uniformly distributed over all the network outputs.

the network outputs. The other 13 senders generate packets whose destinations are uniformly distributed over all the network outputs. With this traffic pattern we divide the first-stage switches into two classes: switches handling *uniform* traffic patterns and switches handling *nonuniform* traffic patterns. For the switches handling uniform traffic patterns, all the inputs are injected from identical senders with uniform traffic. Only switch (1,1) belongs to this category. The switches handling nonuniform traffic are injected with traffic from two different types of senders. For each one of these switches, three of the inputs are injected from identical senders with uniform traffic. The fourth input is injected with traffic from a sender generating the nonuniform traffic as previously described. Switches (2,1), (3,1), and (4,1) belong to this category. Based on the traffic generated and the connected first-stage switches, we can partition the senders into three groups. Senders which generate *uniform* traffic and are connected to the switches handling *uniform* traffic patterns are called *UU senders*. This group includes senders 0, 1, 2, and 3. Senders which generate *nonuniform* traffic and are connected to the switches handling *nonuniform* traffic patterns are called *NN senders*. This group includes senders 4, 8, and 12. Senders which generate *uniform* traffic and are connected to the switches handling *nonuniform* traffic patterns are called *UN senders*. To measure the performance of each group of senders in the following, we define *sender throughput* as the average normalized throughput over all the senders in a group. *Sender latency* is the average latency of the packets transmitted from a defined group of senders.

Using the traffic pattern from Figure 5.22, Figure 5.23 and Figure 5.24 show the performance difference between the three sender groups in terms of the sender throughput vs. overall throughput and the sender latency vs. overall throughput, respectively. The arbitration schemes simulated include the ORR, RR, and SGR-8 arbiters. When the overall throughput is under 0.3, different groups have the same sender throughput. When the overall throughput is above 0.3, the network starts to saturate and the sender throughputs of different groups are distinguished from each

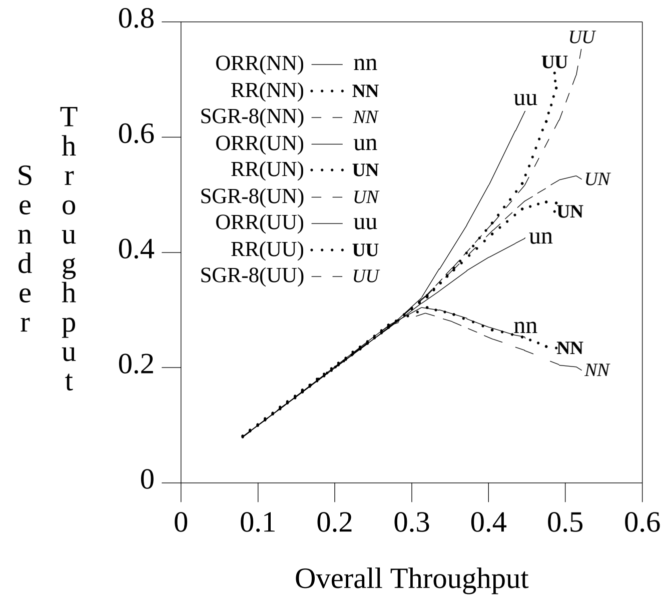

Figure 5.23: Sender throughput vs. overall throughput of a 16×16 network for the ORR, RR, and SGR-8 arbiters under the traffic pattern in Figure 5.22.

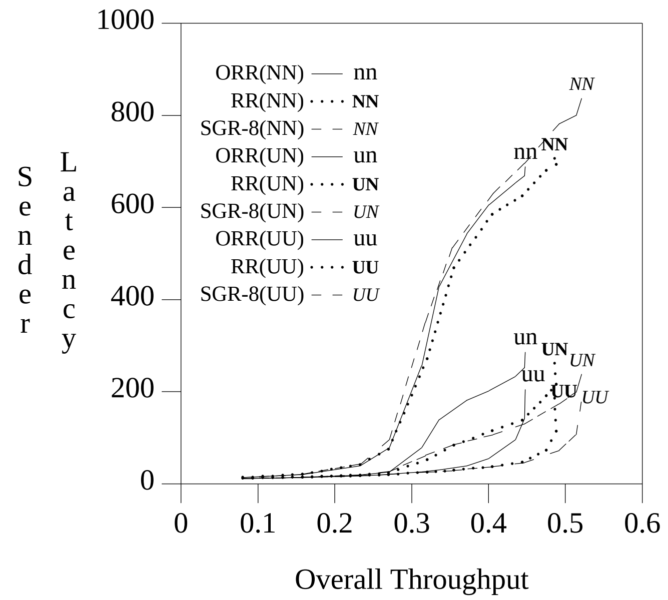

Figure 5.24: Sender throughput vs. overall throughput of a 16×16 network for the ORR, RR, and SGR-8 arbiters under the traffic pattern in Figure 5.22.

other. The NN senders have the lowest sender throughput and the highest sender latency since 80% of their packets are destined to the hot spot. As the overall throughput increases above the 0.3 point, the sender throughput of the NN senders is reduced due to *tree saturation* [Pfis85b]. Packets generated by the NN senders and destined to receivers other than the hot spot are blocked at the senders, since in the first-stage switches, the input buffers are full of the packets destined to the hot spot. For the UU senders and UN senders, the sender throughputs continue to increase as the overall throughput increases above the 0.3 point. The UU senders have a higher sender throughput and a lower sender latency than the UN senders. The reason for this is that for the packets generated by the UU senders, only those destined to receivers 0 to 3 have to traverse a switch handling nonuniform traffic (switch (1,2)), while all the packets generated by the UN senders have to traverse one or two switches handling nonuniform traffic.

Figure 5.23 and Figure 5.24 also show the impact of the arbitration scheme on the performance of three different groups of senders. For the NN senders, the performance of the RR arbiter is slightly better than that of the ORR arbiter, achieving approximately the same sender throughput and slightly lower sender latency. However, the performance of the SGR-8 arbiter for the NN senders is slightly worse than that of the ORR arbiter, with the SGR-8 arbiter achieving a little lower sender throughput and slightly higher sender latency. For the UN and UU senders, the RR arbiter performs considerably better than the ORR arbiter, achieving a higher sender throughput and a significantly lower sender latency. The performance of the SGR-8 arbiter for the UN and UU senders is even better than that of the RR arbiter, with the SGR-8 achieving a higher sender throughput and a

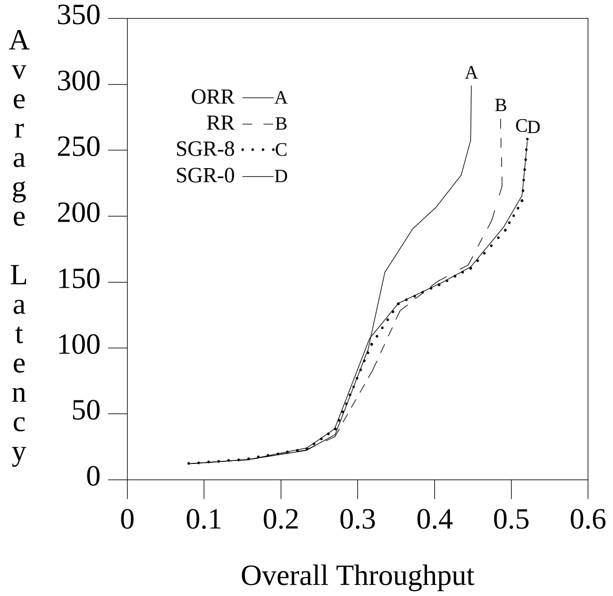

Figure 5.25: Average latency vs. overall throughput of a 16×16 network for the different arbiters under the traffic pattern in Figure 5.22.

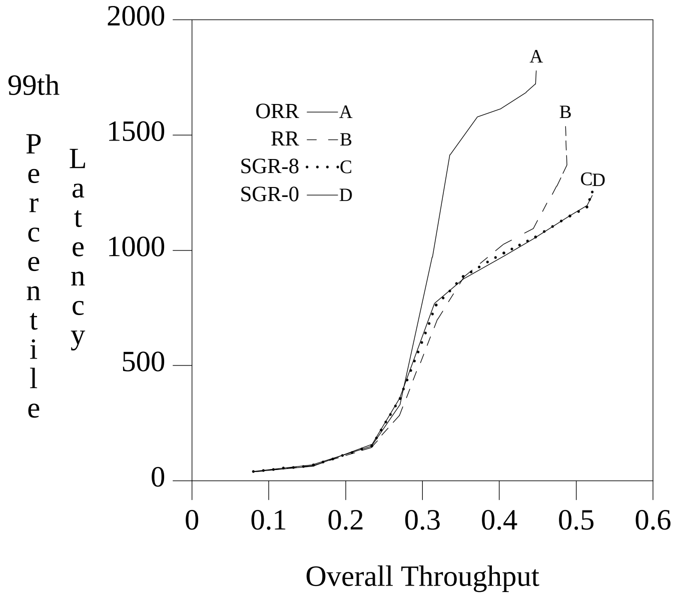

Figure 5.26: 99th percentile latency vs. overall throughput of a 16×16 network for the different arbiters under the traffic pattern in Figure 5.22.

lower sender latency. Overall, the fair priority rotation mechanism used by the RR arbiter improves the performance of the UN and UU senders without reducing the performance of the NN senders. The reservation mechanism used by the SGR-8 arbiter improves the performance of the UN and UU senders at a cost of a small performance degradation of the NN senders, in addition to guaranteeing starvation freedom. These results indicate that by using a fairness mechanism, the RR and SGR-8 arbiters reduce the negative performance impact of the NN senders on the ‘‘innocent’’ UN and UU senders.

Figure 5.25 and Figure 5.26 show the average latency vs. overall throughput and the 99th percentile latency vs. overall throughput, respectively. Unlike the results for a 16×16 network under uniform traffic (see Figure 5.8 and Figure 5.9), the ORR arbiter has the lowest overall throughput. At the same overall throughput, the average latency and the 99th percentile latency for the ORR arbiter are also the highest. Changing the priority shifting scheme from ORR to RR increases the overall throughput, and decreases the average latency and the 99th percentile latency. With the reservation mechanism, the SGR arbiters further improve the network performance, in addition to guaranteeing starvation freedom.

5.3. Implementation

We have shown that starvation prevention can be achieved by adding a reservation mechanism to the symmetric crossbar arbiters. Its performance impact has been evaluated under uniform and nonuniform traffic. Under uniform traffic, the starvation-free arbiters is outperformed by the arbiters without starvation prevention. However, the performance difference is very small with a large starvation count threshold. Under nonuniform traffic, the starvation-free arbiters can significantly improve the performance for queues inclined to be starved. In order to demonstrate the viability of the design of the starvation-free arbiters, we have laid out a WFA-based 4×4 SGR-32 arbiter in custom VLSI. In the SGR arbiters, the capability of starvation prevention is efficiently implemented owing to the regular structure of the symmetric crossbar arbiters. To reserve the row and the column, the top priority arbitration cell simply ‘‘disables’’ the wave front output signals in horizontal and vertical directions (see Figure 5.27).

The basic structure of a WFA-based SGR-k arbiter is similar to the WFA in Figure 3.32, which is shown again in Figure 5.27 for convenience. The detailed inputs and outputs of the arbitration cell for an SGR-k arbiter are shown in Figure 5.28-(a). Figure 5.28-(b) shows the starvation counter used for the starvation count threshold for the whole arbitration array. Signals XP, YP, XI, XO, YI, YO, R, and G have been described in Chapter 3. QT (queue transmitting) comes from the input buffer and there is a separate QT for each arbitration cell. When QT is 1, it indicates that the corresponding queue is still transmitting a packet through the crosspoint, so the associated row and column should not be grabbed by any other queues in the same row or column. There is an RA (row available) for each row. RA is generated by a NOR gate whose inputs are the QT signals in the corresponding row. Similar to RA, there is a CA (column available) signal for each column. However, the inputs of the NOR gate for generating CA include OPB as well as the QTs in the corresponding column.

Each arbitration cell generates a TPR (top priority rejected) signal, which becomes high if the cell has the top priority and the request is rejected. There is an

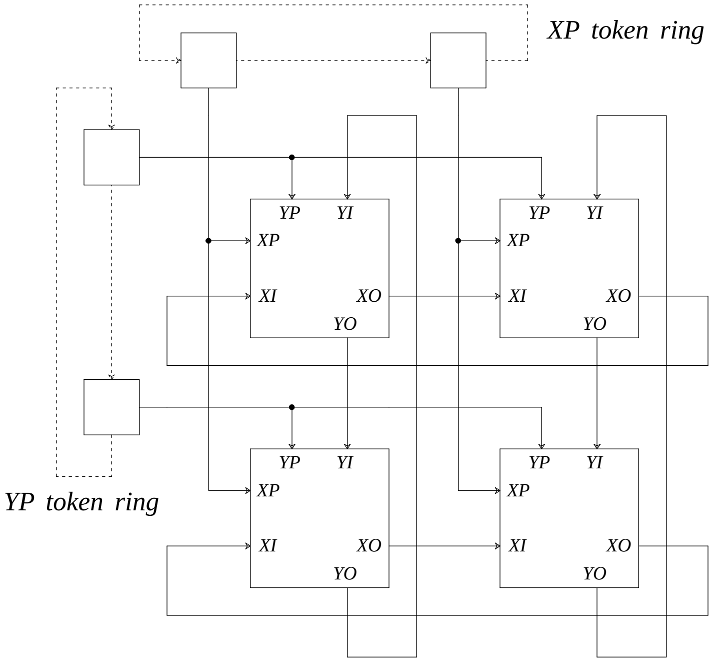

Figure 5.27: Organization of a 2×2 SGR-k arbiter.

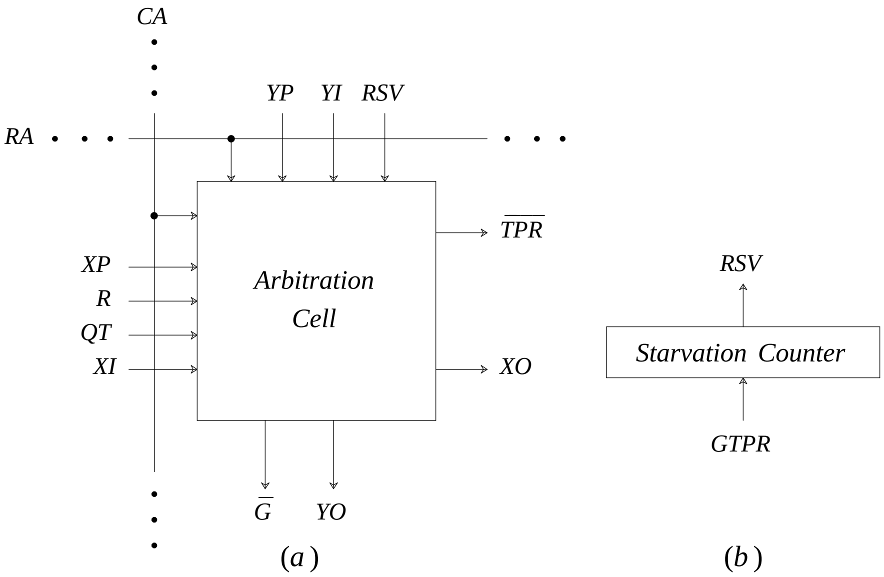

Figure 5.28: An SGR-k arbiter. *(a)* Inputs and outputs of the arbitration cell. *(b)* Starvation counter of the whole arbitration array.

OR tree for the whole arbitration array. All the *TPR*s are the inputs of this OR tree. The output of the OR tree is *GTPR* (global top priority rejected), which is sent to the starvation counter. This *GTPR* indicates if the top priority request is rejected regardless of where the top priority is. If *GTPR* is 1, the starvation counter is incremented by 1 until it reaches the starvation count threshold. When the starvation counter reaches the starvation count threshold, *RSV* (reservation) becomes 1. The *RSV* signal is broadcast to every arbitration cell to inform the top priority cell if it should start to reserve the required row and column. This reservation can be achieved by simply disabling the *XO* and *YO* outputs. The detailed circuit of the arbitration cell is shown in Figure 5.29.

Figure 5.30 shows the floorplan of the WFA-based SGR-32 arbiter. The size of the boxes in the figure is proportional to the size of the modules in the layout. The layout of the crossbar is 990 λ wide by 380 λ tall, while the layout of the arbiter is 1050 λ wide by 930 λ tall, including the circular shift registers and the starvation counter. SPICE circuit simulations using 2 μ CMOS technology indicate that the worst case delay for the arbiter is 23 ns. A WWFA-based SGR-32 arbiter has been laid out too, and the worst case delay for the arbiter is 15.5 ns.

In order to measure the cost of adding the starvation prevention mechanism, a WFA-based 4×4 RR arbiter has also been laid out. The layout area of this RR arbiter is 1010 λ wide by 870 λ tall. The worst case delay for the RR arbiter is 20 ns. Hence, if the arbitration delay is part of the critical path of the switch, adding the starvation prevention mechanism does have a small performance penalty which was not taken into account in the previous section.

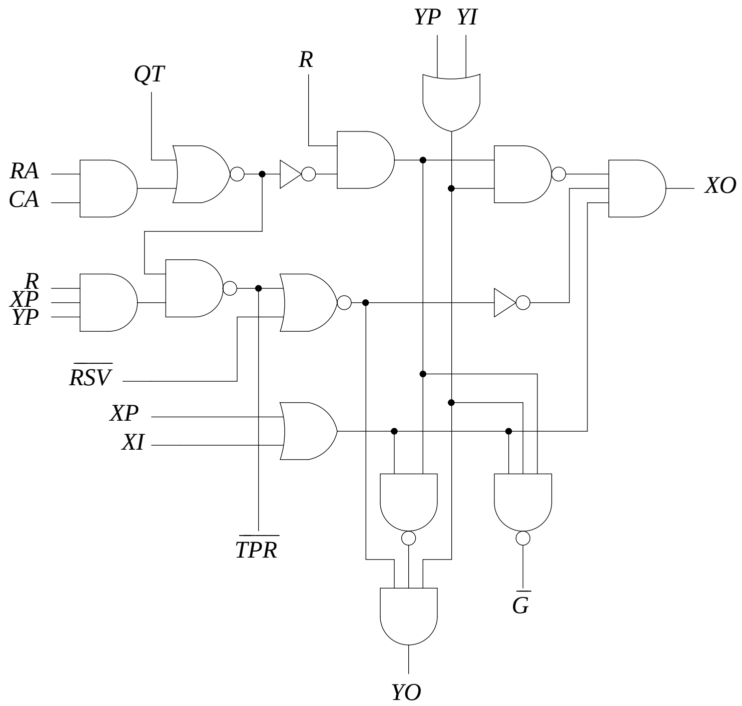

Figure 5.29: Detailed logic design of the arbitration cell.

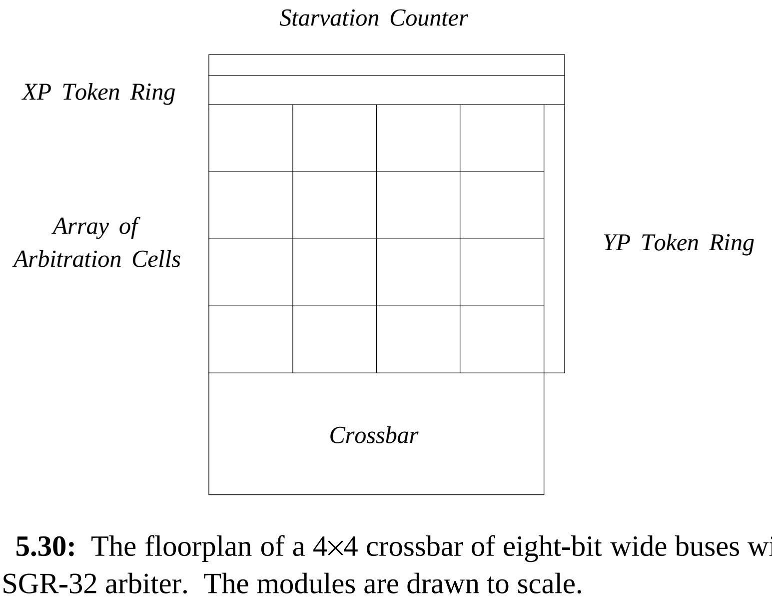

Figure 5.30: The floorplan of a 4×4 crossbar of eight-bit wide buses with the SGR-32 arbiter. The modules are drawn to scale.

5.4. Summary and Conclusions

While our proposed symmetric crossbar arbiters lead to excellent performance in terms of throughput and average latency, they do not guarantee fairness. Hence, it is possible for an ‘‘unlucky’’ packet to be left in a switch buffer for a long time, potentially forever, while other packets are forwarded quickly through the switch. In a synchronous network, starvation prevention is relatively simple. In an asynchronous network where the communication switches use multi-queue input buffers, eventual packet delivery cannot be guaranteed unless special starvation prevention mechanisms are employed.

We have introduced and evaluated a reservation mechanism which guarantees starvation freedom in an asynchronous network. Simulations show that the proposed starvation-free arbiters cause a minor performance degradation for uniform traffic. However, under certain nonuniform traffic patterns the starvation-free arbiters may outperform arbiters that lack a starvation prevention mechanism. To evaluate the implementation cost of the starvation prevention mechanism, we have laid out in custom VLSI wave front arbiters with and without the starvation prevention mechanism. The overhead of the mechanism was found to be 11% in terms of layout area and, in the worst case, 15% in terms of arbiter circuit performance.