Chapter Six

Buffer Performance Evaluation with Asynchronous Networks

Chapter 4 presented an evaluation of the system-level performance of the DAMQ buffer. A Markov analysis and an event-driven simulator were used to measure the performance of the DAMQ buffer relative to other buffer architectures, in the context of a synchronous multistage interconnection network. As was discussed there, the synchronous network was used because it is easier to reason about switch behavior in a synchronous network, and there exists a large amount research which evaluates switch and buffer architectures using synchronous networks.

However, the DAMQ buffer was not designed with a synchronous network in mind. There are several features which we claim to be important for high performance switching (Ch. 3) which are not utilized in a synchronous network. Specifically, transmitting variable length packets does not make sense in a network which defines a constant period of time for the transmission / reception of each packet, and virtual cut-through does not exist in a network which requires packet transmissions to occur on stage cycle boundaries (Ch. 4).

This chapter evaluates the performance of the DAMQ buffer in an asynchronous network. In the next section, the simulator which generated the performance numbers discussed in this chapter is described. In Sec. 6.2, the performance of the DAMQ buffer in an asynchronous multistage interconnection network (omega topology) is evaluated. We compare the latency and throughput of traffic in networks composed of DAMQ switches to that of networks with other buffer architectures. Also, the performance of the asynchronous omega networks are compared to the synchronous network results reported in Ch. 4. Finally, in Sec. 6.3, the simulator is used to demonstrate the ability of the DAMQ buffer to implement deadlock-free dimensional routing and support high communication throughput for a two-dimension torus network.

6.1. Simulating an Asynchronous System

The simulator which was used to generate the results presented in Chs. 4 and 5 was modified to perform the network performance measurements presented in this chapter and in Ch. 8. In addition to modifying the simulator such that it simulated an asynchronous network, the abilities of the simulator were greatly expanded. The capabilities of the simulator are presented by discussing the parameters with which the simulator's behavior is controlled. In the following discussion, the parameters are placed into three groups according to the aspect of the simulator they control: the network architecture, packet injection, and system behavior / data collection. Tab. 6.1 summarizes the simulator's parameters.

6.1.1. Network Architecture

The simulator was modified to allow topologies other than an omega network. Currently, *topT* can be set to omega, two-dimensional torus, two-dimensional mesh or hypercube (results from the first two topologies are presented in this dissertation). Networks of various sizes in these four topologies can be established by setting the *sys_size* parameter. The size of the switches (the number of input and output ports per switch) can be set via the variable *switch_size*. These variables are not independent: for tori and meshes, which are two-dimensional and square, $switch\_size = 5$; for an omega network, size of the network (the number of senders and receivers) is a power of the size of the switches ($sys\_size = switch\_size^n$) and, for hypercubes, $sys\_size = 2^{switch\_size}$.

A fundamental parameter to the simulator is the *hop_delay*. This parameter defines the time from when a packet wins an arbitration and is transmitted to when the packet is able to participate in an arbitration on the next switch. The hop delay is actually an agglomeration of the delays which constitute the minimum switch cut-through time: the time to arbitrate the switch crossbar, the latency of the packet traversing the crossbar, the link and the synchronizer, and the time to route the packet on the next switch. A packet's latency through the network equals the hop delay times the number of hops *plus* the total amount of time the packet spends in buffers. A packet spends zero time in buffers ($latency = hop\_count \times \#hops$) if its sender transmits the packet as soon as it is created and every switch cuts the packet through immediately upon receipt. For all of the simulation results presented in this and following chapters, a hop delay of five clock cycles was used. This figure reflects the four cycles required to cut through the DAMQ buffer (see Ch. 7) plus one cycle to traverse the wires connecting two switches.

Link utilization is limited by the *link_rest_time* parameter. The *link_rest_time* is the number of clock cycles a link must remain idle between packets to allow the synchronization logic within the input port to which the link is connected to be reset. Thus, if the link rest time is *link_rest_time*, and if a buffer transmits the last byte of a packet on clock cycle t_0, then the buffer can transmit the

System-wide parameters	
switch_size	number of input and output ports
sys_size	number of switching elements in the system
buffer	the buffer architecture
blen	the size of the buffer
max_pkt_length	the maximum packet length
topT	the topology
arb	heuristic for crossbar arbitration
flowctl	hop-level flow control
hop_delay	the time for a packet header to make one hop
link_rest_time	the minimum time a link must be idle between packets
#msgs	the number of messages to receive before terminating the simulation
#ignore	the number of messages to receive before starting statistic-gathering
statint	the number of messages to receive in each confidence interval
Per group (n) parameters	
bit_mask	parameter for sender group specification
val	parameter for sender group specification
msgint	inter-message delay distribution
intpar1	parameter for inter-message delay
intpar2	parameter for inter-message delay
Per traffic type ($n \times t$) parameters	
perc	probability of message being of traffic type *t*
selDst	message destination distribution
dpar1	parameter for packet destinations
dpar2	parameter for packet destinations
msglen	message length distribution
LenPar1	parameter for message length
LenPar2	parameter for message length

Table 6.1: *Parameters to the Asynchronous Simulator.*

first byte of a packet on cycle t_1, but the output port through which the first packet was transmitted cannot transmit another packet until cycle t_x,

$x = link\_rest\_time + 1$. The synchronization logic is assumed to be that described in [Tami88a]. That being the case, the input port can be reset on the clock cycle immediately following a packet transmission. The *link_rest_time* is set to two cycles in the simulations, however, to ensure that clock slippage between neighboring switches will not cause data to be lost.

The purpose of this simulator is to explore the relative performance of various communication network architectures. As such, the size of the buffers and their architecture can be specified. To date, FIFO, DAMQ, SAMQ and SAFC buffer architectures have been implemented (see Ch. 3 for discussion of these buffer architectures). Also, a large number of hop-level flow control mechanisms have been implemented — Ch. 8 presents an evaluation of their performance. The multistage interconnection network simulations whose results are presented in this chapter use *blocking flow control*. Packet transmissions are blocked when there is less than *max_pkt_length* bytes available for packet reception. Thus, if a packet transmission begins, it is guaranteed to complete (i.e. this simulator does not simulate wormhole routing).

6.1.2. Packet Injection Parameters

Packets can be injected into the simulated network either stochastically or deterministically (i.e. trace-driven simulations). Deterministic packet injections are controlled by a file which specifies the size, source, destination and time of creation for each packet which traverses the network. To date, this feature has only been used for debugging the simulator. For stochastic packet creation there are a number of parameters. First, there is the inter-message delay (*msgint*) — the time from when one packet is transmitted to when the next message is created. Three delay distributions are supported by the simulator: constant (each message is created a fixed number of clock cycles after the previous is injected into the network), uniform (the delay varies uniformly over a specified range) and geometric. All of the results presented in this dissertation use the geometric inter-message delay distribution.

Second, there is the message length. Three message length distributions (*msglen*) are implemented: constant length, length which varies uniformly over a range, and geometrically-distributed length. The maximum packet size (*max_pkt_length*) must also be specified; if a message is longer than the maximum packet size, it is broken into multiple packets; each packet of the message is created after the previous packet of the message is injected into the network. The simulations presented in this dissertation use a maximum packet length of thirty-two bytes, which is also the maximum packet length supported by the DAMQ Buffer Chip (Ch. 7).

The third set of parameters for stochastic message generation control the assignment of destinations to messages. There are a wide variety of random destination distributions implemented, but only three which are used in this dissertation. First, the uniform distribution over all destinations — each packet created under this distribution has equal probability of being addressed to any destination in the network (for point-to-point topologies, nodes cannot send messages to themselves). Second, there is the hot spot distribution, in which a node or nodes address all messages to a single destination. Finally, there is the uniform distribution over a subset of the destinations.

In order to increase the flexibility of the stochastic message generation, each sender in the network is allowed multiple sets of the parameters for message length and destination selection. These sets are referred to as *traffic types*; each time a sender generates a message, the first function performed is determining which type of traffic the message will be. For example, if a sender is transmitting three percent of its packets to a hot spot, with the other ninety-seven percent of its packets being uniformly distributed, then it will have two traffic types. Three percent of the messages created will be of the first traffic type — hot spot — and ninety-seven percent will be of the second type — uniformly distributed.

6.1.3. Sender Grouping

The model we developed for congestion within communication networks (see Ch. 8) involves a group or groups of processors which create the congestion, and ‘‘innocent’’ groups of processors which must be protected from the effects of this congestion. In order to support this concept, the simulator is able to place the senders into groups. This is done by establishing a bit mask and a value for each group. If $senderID \; \& \; bit\_mask_n = val_n$, and the sender has not been assigned to a group i, $i < n$, then the sender is in group n. This mechanism for grouping works well for simple, regular topologies such as the four currently supported by the asynchronous network simulator.

6.2. DAMQ Buffer Performance: Asynchronous MIN

In this subsection, the asynchronous network simulator is used to explore the performance of a multistage interconnection network (omega topology) transmitting packets in a uniform distribution. First, we present and evaluate the results of networks transmitting constant-length (32-byte) messages. Using constant-length packets allows us to compare these results to the results obtained from the synchronous network which were presented in Ch. 4. Following this comparison, the results for networks transmitting variable-length message are presented and the results are evaluated.

6.2.1. Constant-Length Packets, Asynchronous Network

Figure 6.1 shows results for the synchronous 64×64 omega network which were previously presented in Fig. 4.1 and results for asynchronous 64×64 and 256×256 omega networks. These graphs show the average latency of packets traversing the network vs. the network throughput. The simulations are of networks of 4×4 switches with buffers which can store up to four packets at each input port.

The first difference to note between the synchronous and asynchronous results are the units of throughput and latency used. The synchronous network measures latency in terms of stage cycles. Since the 64×64 network is constructed from three switching stages, the minimum network latency is three stage cycles†. The

† It actually requires four hops to traverse a three-stage omega network. Historically, the output ports of the last switching stage have been treated as sinks. Thus the time to actually perform the last hop is not included in the latency, although time spent in the last switching stage's queues is included.

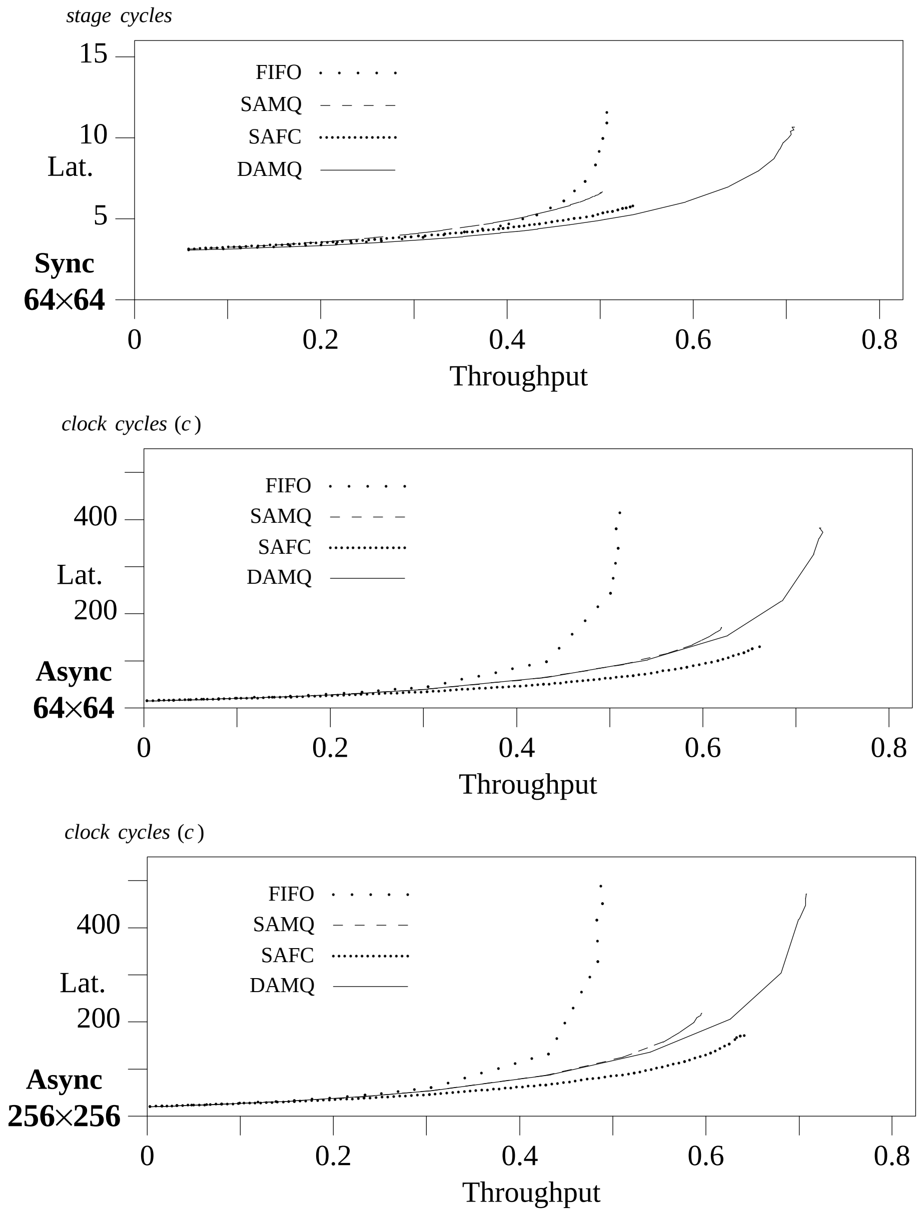

Figure 6.1: *Synchronous vs. Asynchronous Protocols, Omega Network.* Average latency vs. throughput results for 64×64 omega network with synchronous protocol (also in Fig. 4.1) and 64×64 and 256×256 omega networks with asynchonous protocol. 4×4 switches, thirty-two-byte packets, storage for four packets at each input port.

asynchronous network measures latency in terms of clock cycles (c). Since the minimum hop time is five clock cycles, the minimum latency across the 64×64 network is 15 c, and the minimum latency across the 256×256 network is 20 c.

While the curves from the synchronous and asynchronous networks are of similar shape, one sees that, in the asynchronous network results, the average network latency at saturation throughputs is an order of magnitude or more higher than the latencies at low throughputs. In contrast, the average latency at saturation throughputs for synchronous networks is only three to five times higher than it is at low throughputs. This difference is due to virtual cut-through — the asynchronous network supports virtual cut-through, which dramatically reduces packet latencies in networks operating at low throughputs [Kerm79, Ngai89].

Perhaps the most interesting point of comparison between the results of the two network types is the saturation throughput of the different switches. The saturation throughput of the networks of FIFO and DAMQ switches is unchanged between the two networks. SAMQ and SAFC switches, on the other hand, achieve higher throughputs with the asynchronous network than they do with the synchronous. We will first explore why the throughputs of the SAMQ and SAFC networks changed, and then why those of the DAMQ and FIFO networks did not change.

The reason for the increase in the saturation throughput of the SAMQ and SAFC networks has to do with a reduction in the back pressure experienced by the switches in the first stage of the network. As was seen in Tab. 4.1, small statically allocated buffers discard or block packets (depending upon the flow control mechanism) even at low applied loads; their buffer utilization is inefficient relative to dynamic buffer allocation. Virtual cut-through increases the effective buffer size by reducing the amount of time packets spend in buffers. At saturation throughput, a relatively small percentage of the packets traversing the network experience virtual cut-through. The virtual cut-through that does occur, occurs in the latter stages of the network, which experience a lower applied load, shorter packet queues and less back pressure than does the first switching stage. FIFO- and DAMQ-buffered networks, on the other hand, experience little or no reduction of first-stage back-pressure in going to an asynchronous network. Referring again to Tab. 4.1, these buffers discard (or block) packets with a much lower probability at low throughputs, and so receive little benefit from virtual cut-through when operating at saturation throughput.

To understand the behavior of the DAMQ buffer network, we examined the behavior of a single 4×4 switch with DAMQ buffers operating at saturation throughput. Our hypothesis was that the asynchronous network throughput equals that of the synchronous network because the switches *self-synchronize* — the network behaves synchronously in the presence of packets of constant length which are being transmitted at saturation throughput, even though the network is asynchronous. The next subsection explains the reasoning behind this hypothesis and then provides evidence that this is the explanation for the asynchronous network transmitting constant-length packets achieving the same saturation throughput as the synchronous network.

6.2.2. Self-Synchronizing Networks

On a 4×4 switch, each time a buffer completes a packet transmission, it requests a connection to an output port for which it has packets. For a switch in an asynchronous network, this can occur on any given clock cycle; one would expect that the probability would be low that multiple buffers would complete their transmissions on the same clock cycle. Assume that the four buffers B_a, B_b, B_c and B_d of a 4×4 switch are currently connected to and transmitting through the four output ports O_w, O_x, O_y and O_z. When buffer B_a completes its transmission to O_w, if the other three buffers are in the middle of transmitting packets, B_a will be re-connected to O_w or, if it has no more packets destined for that output port, B_a will remain idle. When the latter occurs, B_a will remain idle until another buffer (B_b) completes its transmission. At this point, the arbiter will consider the two buffers (B_a, B_b) and the two output ports (O_w, O_x) simultaneously. The fact that B_a contains no packets destined for O_w increases the probability that B_a *does* have packet(s) for O_x, and that the arbiter will swap the buffer / output port pairing. If the swap does occur ($B_a \to O_x$, $B_b \to O_w$), then these two buffers are synchronized — and as long as they experience saturation throughput, neither output port is blocked, and they have packets for the two output ports, the two buffers will remain synchronized (i.e. they will continue to complete transmissions and arbitrate on the same clock cycle). As the other two buffers run out of packets for the output ports they are associated with, they will join the first two, and eventually all four buffers will operate in synchrony (all four participating in every crossbar arbitration).

This phenomena is illustrated by the solid line of the graph in Fig. 6.2. This graph shows the number of packets transmitted per crossbar arbitration vs. time, for a network consisting of a single switch forwarding thirty-two byte packets at saturation throughput with $link\_rest\_time = 0$. Initially (at time 0), the network is asynchronous; the time each buffer completes a transmission is independent of the other three, and each arbitration results in a single packet being transmitted. However, as time passes, the buffers synchronize, until by time 320, crossbar arbitrations occur every thirty-two clock cycles, and every arbitration considers all four buffers and output ports. While there is no guarantee that all four buffers will transmit after every arbitration, three or four packets were transmitted per arbitration for ninety of the ninety-four arbitrations which occurred between times 320 and 3296 (two packets were transmitted for the other four arbitrations which occurred in that time period).

This scenario for switch self-synchronization ignores an important aspect of our switch design; the link idle time between packets. In the scenario for buffer synchronization just described, the synchrony begins when an idle buffer joins another buffer in a two-way arbitration. However, in our model, when B_b completes a transmission while B_a is idle, B_b's output port (O_x) cannot receive data for another two clock cycles (assuming $link\_rest\_time = 2$). B_b can, however, transmit another packet immediately; if B_b has packets destined for O_w (the output port that B_a was previously transmitting to), it will immediately grab that output port and begin to transmit. If, in turn, the B_a has packets destined for O_x, it will be connected to that output port in a crossbar arbitration two clock cycles later.

The two-cycle link rest time thus causes the switch to achieve a ‘‘loose’’ synchrony, in which the buffers contend for output ports over a span of clock

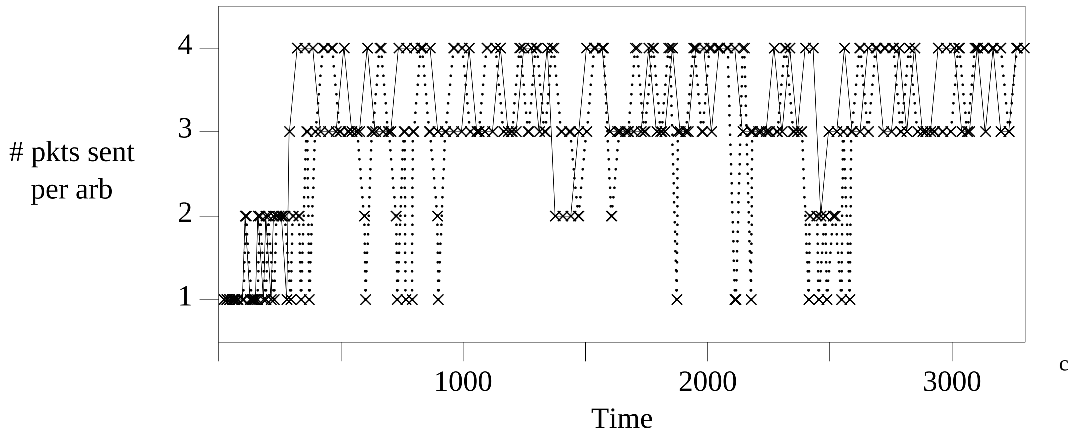

Figure 6.2: *Self-Synchronizing DAMQ Switches.* Results of a 4×4 network (a single DAMQ switch), transmitting thirty-two byte packets at saturation throughput. Shown is the number of packets transmitted per arbitration vs. time (clock cycles) for networks with zero cycles of link idle time between packets (solid line) and with two cycles of link idle time between packets (dotted line).

cycles, as opposed to all participating in a crossbar arbitration on the same clock cycle. This is a more dynamic situation than the ‘‘true’’ synchrony which produced the solid line in Fig. 6.2. For a single buffer transmitting a sequence of packets, the interval between two succeeding transmissions of a single buffer may be as low as thirty-two clock cycles or as high as forty without losing synchrony.

In order to visualize this loose synchrony, we defined a logical arbitration $arbitration^L$ to be a set of crossbar arbitrations such that each arbitration occurs on the same cycle as another member of arbitration^L or *link_rest_time* clock cycles from another member of arbitration^L. Thus, if $link\_rest\_time = 2$, and if buffer B_a begins a transmission on cycle t_x, and buffer B_b begins a transmission on cycle t_{x+2}, then B_a and B_b have achieved loose synchrony, the arbitrations which occurred on t_x and t_{x+2} are members of the same arbitration^L, and the number of packets transmitted on this arbitration^L is two. Further, if B_c begins a transmission on cycle t_{x+4}, and B_d begins one on cycle t_{x+6}, then *four* packets where transmitted on this arbitration^L. On the other hand, if the transmission from B_b were to occur on t_{x+1}, then B_a and B_b are not operating in synchrony, even though the times in which their transmissions begin are even closer together, and the arbitrations for B_a and B_b are members of different arbitration^{L}s.

The dotted line in Fig. 6.2 shows the DAMQ switch forwarding thirty-two byte packets at saturation throughput with $link\_rest\_time = 2$. Each arbitration^L for the loose synchrony case is marked with two $\times$-s — one marks the time of the first arbitration in the set, the second marks the time of the last. There are clear differences between this loose synchrony and the tight synchrony achieved when the link rest time was zero. There is a higher probability of transmitting only one or two packets for an arbitration^L under loose synchrony. Also, the interval between arbitration^{L}s is irregular; the switch even appears to lose synchrony for a series of arbitrations between 2400 c and 2600 c. However, these differences are the exception — for the majority of the 3300 cycles of simulation displayed in 6.2, the loosely-synchronized DAMQ chip behaves very similarly to the tightly-synchronized case. Further, as the throughput figures attest, the loosely-synchronized DAMQ network behaves similarly to the synchronous DAMQ network evaluated in Ch. 4.

One could prevent a circuit from self-synchronizing in the presence of constant-length packets by not using the same link rest time after every packet. This would be detrimental to the network performance, however. When the network synchronizes, it gives the arbitration logic more buffer / output port pairings to choose from, which improves port utilization. Evidence of this is provided simply by the fact that, when buffers become synchronized, they go from being idle to transmitting, and buffers lose synchrony when they go from transmitting to being idle.

Switches with the FIFO and SAMQ buffer architectures also self-synchronize. Switches composed of FIFO buffers synchronize more quickly than do DAMQ switches because of their susceptibility to output port contention. SAMQ buffered switches will also synchronize in a shorter time span that do DAMQ buffered switches. Although their behavior is the same as DAMQ buffers with respect to crossbar arbitration, SAMQ buffers utilize their buffer space more poorly than do DAMQ buffers. Thus, SAMQ buffers are more likely to become idle, which is when loose synchronization occurs. SAFC switches, on the other hand, can transmit uniformly-distributed constant-length packets at saturation throughput indefinitely without synchronizing. This is due to the fact that the queues within a single SAFC buffer operate independently of one another, so each buffer can participate in every arbitration without being in synchrony.

6.2.3. Variable-Length Packets, Asynchronous Network

In this section, the performance of multistage interconnection networks transmitting variable-length packets is examined. Given the conclusion of the previous section — that switches with DAMQ buffers maintain their high performance in asynchronous networks by achieving a loose synchrony — one might expect the SAFC switch performance to improve relative to the other buffers in the presence of variable-length packets. The SAFC switch is the only architecture examined which is not synchronized by forwarding constant-length packets at saturation throughput; all four input ports participate in every output port arbitration, and this is unaffected by the asynchrony caused by variable-length packets.

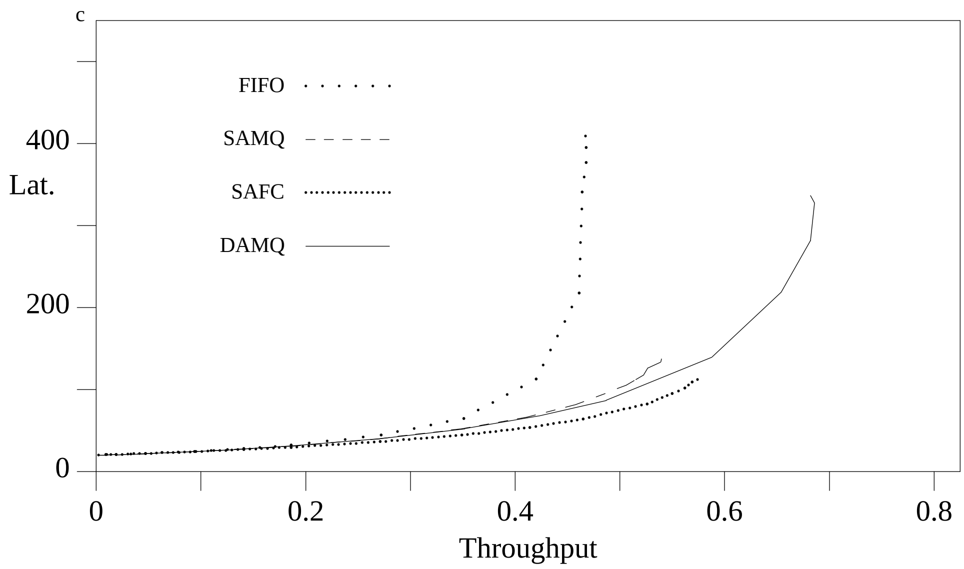

Figure 6.3: *Lat. vs. Thpt., Variable-Length Packets.* 256×256 omega network composed of 4×4 switches. Packets vary in length uniformly from six to thirty-two bytes. Packet destinations randomly chosen, uniform distribution.

Figure 6.3 indicates that this is not the case. On the contrary, the reduction in the saturation throughput of networks of DAMQ and FIFO buffers (0.71 to 0.68 for the 256×256 DAMQ buffer network) is much less than the reduction in the throughputs of the networks with SAMQ and SAFC buffers (0.64 to 0.57 for the 256×256 SAFC buffer network). The reason for this is buffer memory fragmentation; with packets of varying length, statically allocated buffers lose more of their buffer memory to fragmentation than does the DAMQ buffer. With

128-byte statically allocated buffers, each queue is limited to thirty-two bytes in length. Since the flow control mechanism blocks packets destined for a queue when there are less than thirty-two bytes available in the queue, a single byte of data stored in a queue will cause that queue to block. With a dynamically allocated buffer, however, there only need be thirty-two bytes available in the entire buffer memory in order for any queue to to receive a packet.

Thus, while variable-length packets eliminate any form of synchrony (which reduces the throughput of switches implemented with DAMQ buffers), it increases the average number of packets stored within the DAMQ buffer, which in turn increases the switch's ability to utilize its output ports. For SAFC switches, on the other hand, variable-length packets cause buffer memory to be lost to fragmentation. This strongly impacts the SAFC switch throughput, as inefficient buffer utilization is what limits the performance of statically allocated buffers.

6.2.4. Network Throughput with Increased Buffer Memory

Figures 6.4 and 6.5 present network saturation throughput vs. buffer size for 4×4 and 64×64 networks composed of 4×4 switches with DAMQ, SAMQ and SAFC buffering. Fig. 6.4 presents the constant-length packet results. For small buffer memories, the DAMQ buffer provides the highest saturation throughput of the three buffer types, since it more efficiently uses the small buffer memory. As the buffers grow in size, static vs. dynamic buffer allocation becomes less of an issue, but the saturation throughput of the 4×4 network approaches its asymptote (0.941 for thirty-two byte packets with two cycles link idle time between packets) before the SAFC buffer performance overtakes the DAMQ buffer performance.

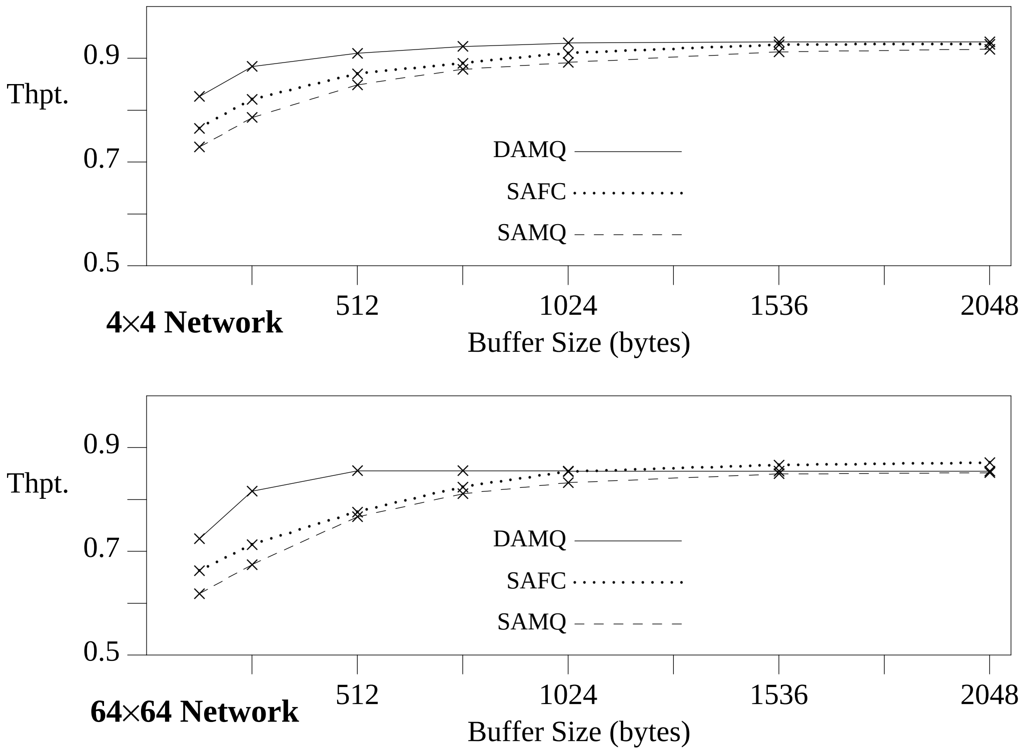

Figure 6.4: *Saturation Throughput vs. Buffer Size, Constant-Length Packets.* Shown is the maximum netork throughput as a function of the amount of buffer memory at each input port for a 4×4 network (a single chip) and a 64×64 network.

Fig. 6.5 shows the same thing occurring with variable-length packets. While the higher connectivity of the SAFC switch will support a higher throughput than a DAMQ switch as buffer sizes increase, the fact is that at the buffer sizes where static allocation is no longer a handicap, switch throughputs are so close to the their maximum that the additional connectivity of the SAFC buffer provides little in the way of throughput enhancement.

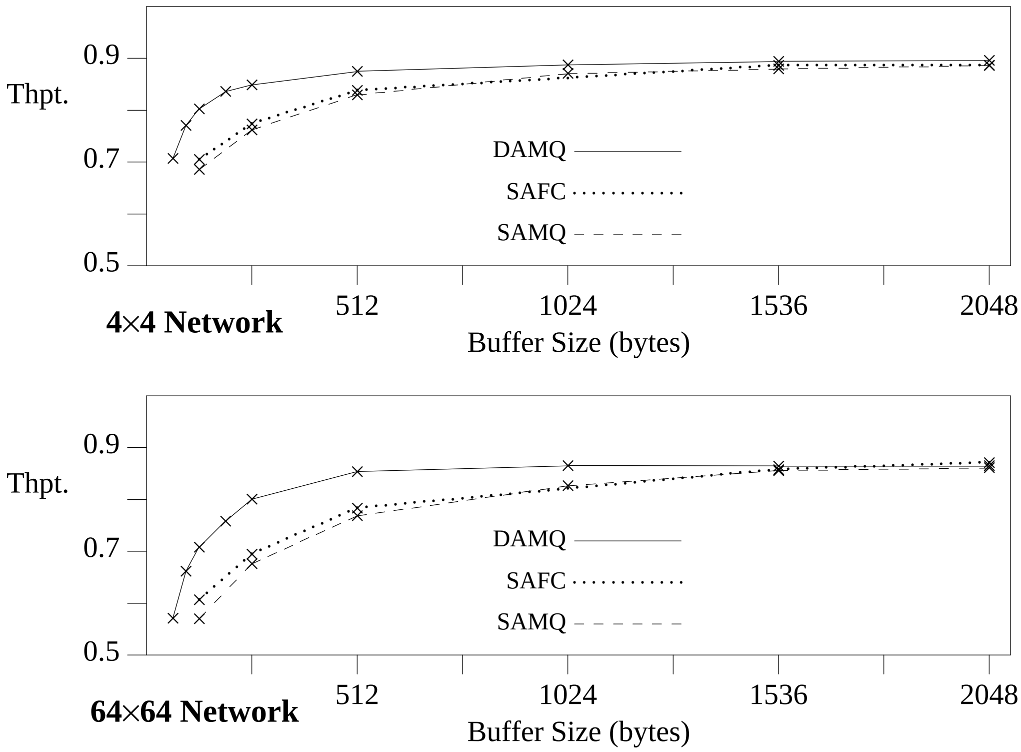

Figure 6.5: *Saturation Throughput vs. Buffer Size, Variable-Length Packets.* Shown is the maximum netork throughput as a function of the amount of buffer memory at each input port for a 4×4 network (a single chip) and a 64×64 network.

6.2.5. Summary: Asynchronous MIN Performance

Our investigations into asynchronous multistage interconnection performance described in Sec. 6.2 had three goals: (1) evaluate the performance of the DAMQ buffer in the context of an asynchronous network, (2) confirm the simulation results which were presented in Ch. 4 and (3) gain new insight into the behavior of communication switches.

The simulation results presented in this section demonstrate the performance of the DAMQ buffer in an asynchronous network. Multi-queue buffering continues to provide a significant performance advantage over FIFO buffering — dynamic buffer allocation is as significant an advantage for asynchronous networks as it was for synchronous. The performance difference between dynamic and static buffer allocation is diminished under a traffic load of constant-length packets, where statically allocated buffers benefit from an increased effective buffer size due to virtual cut-through. However, statically allocated buffers are more susceptible to fragmentation that is the DAMQ buffer when forwarding packets of varying length; with variable-length packets, the DAMQ buffer performance improves relative to both SAMQ and SAFC buffers.

6.3. Network Performance, Torus Topology

Chapter 4 demonstrated the DAMQ buffer's performance for synchronous multistage interconnection networks (MIN). The previous section of this chapter demonstrated that the DAMQ buffer operates equally well in asynchronous MINs. This section examines the suitability of multi-queue buffers for the torus topology. The results of simulating torus networks constructed from switches implemented with DAMQ and SAFC buffers are presented; the performance of these buffer architectures are compared to each other and to the results of wormhole routing presented in [Dall90b].

6.3.1. Dimensional Routing with Multi-Queue Buffers

In [Dall87a], Dally and Seitz present a deadlock-free routing algorithm for k-ary n-cubes (cube-connected networks consisting of n dimensions and a radix of k). The algorithm has two components to prevent resource dependency cycles (the cause of deadlock in communication networks). To prevent inter-dimensional cyclical dependencies, packets traverse the dimensions in a fixed order. That is to say, a packet being sent from processor P_a to P_b is first routed in dimension d_{n-1}, then d_{n-2}, etc. Thus, a packet traversing d_i can never be blocked by a packet traveling in d_j, $j > i$, and in a network of dimension n $(d_{n-1} \cdots d_0)$, packets traveling in d_0 will never be blocked by packets traversing a different dimension.

To prevent intra-dimensional cyclical dependencies, packets traverse a dimension on one of two virtual circuits. All packets entering the dimension are injected into circuit V_0. If the packet passes through the *wrap-around link* (the link connecting nodes N_{k-1} and N_0, for a network of radix k), then the packet is moved to V_1. All communication links are multiplexed between V_0 and V_1, and there are separate buffers for the virtual circuits at every input port. There cannot be a cyclical dependency within V_0 or V_1 (since neither virtual circuit forms a complete loop in a dimension), and while packets in V_1 are able to block packets in V_0 via the wrap-around link, packets in V_0 cannot block packets in V_1. Thus, cyclical dependencies are prevented. While the proof of deadlock-freedom for dimensional routing presented in [Dall87a] is for a torus network with unidirectional links, it is easily extended to tori implemented with bidirectional links, with the caveat that a packet, having taken a hop within a dimension, may not change direction (double back upon itself) within that dimension.

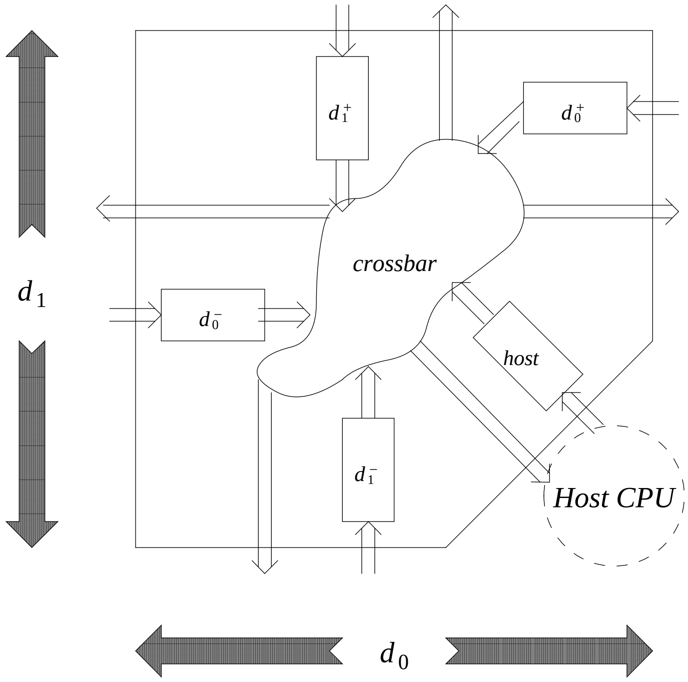

Figure 6.6: *Switch for Two-Dimensional Torus.*

A two-dimension torus with bidirectional links can be constructed from 5×5 switches. Each dimension is allocated two input / output port pairs (one pair for the positive direction, one for the negative), and the switch has an input / output port pair to communicate with the local (host) processor (Fig. 6.6). To support dimensional routing, a multi-queue buffer is placed at each input port. We will examine the internal structure of the buffers residing at the d_1 input port.

In this torus, packets first traverse d_1, and then d_0. Packets arriving at a d_1 input port are in one of five classes, and are appended to the packet queue associated with that class (Fig. 6.7). First, the packet could be destined for the local (host) CPU; if so, it is appended to Q_h. Second, the packet can have completed its traversal of this dimension (d_1), and be destined for one of the two

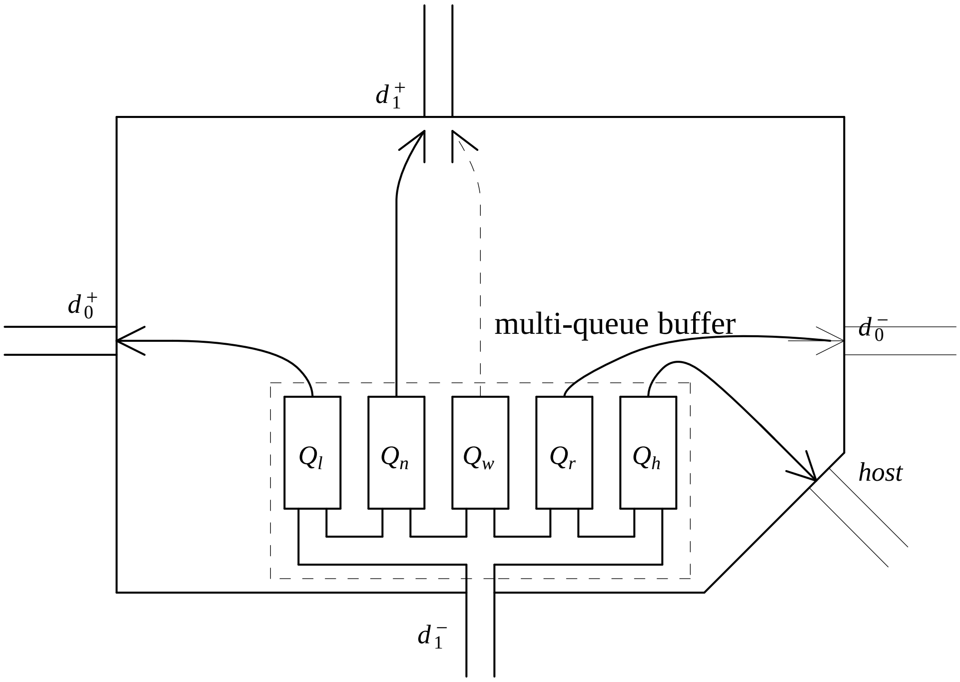

Figure 6.7: *Multi-Queue Buffer for Two-Dimensional Torus.*

d_0 output ports. In this case, the packet is appended to Q_l or Q_r (the queues for packets making a left or a right turn into the next dimension). Finally, if the packet is continuing in d_1, it may be a *wrapped* packet (one which has traversed the $N_{k-1} \rightarrow N_0$ or the $N_0 \rightarrow N_{k-1}$ links, depending upon the direction in which it is traveling) or *not-wrapped*. Q_w or Q_n are, respectively, the queues to which the wrapped and not-wrapped packets are appended.

Given the buffer organization above, deadlock-freedom can be guaranteed if Q_n is not allowed to occupy the entire buffer memory. The packets in Q_h are assured to eventually make progress, as the local node is assumed to be a packet sink. The packets in $Q_{l,r}$ will eventually move forward, since, if the current dimension is guaranteed make progress, then the succeeding dimensions are. Thus, the only concern is to prevent the packets in $Q_{n,w}$ from creating a cyclical dependency, and this is accomplished by preventing Q_n from occupying the entire buffer memory at an input port.

Statically allocating the buffer memory amongst the queues in the manner of SAMQ or SAFC buffers prevents a single queue from occupying the entire buffer memory. For the DAMQ buffer, *maximum usage flow control* prevents this from occurring. This is a flow control mechanism which blocks the transmission of packets which will be appended to a queue whose length is beyond a given threshold. Its operation is identical to the flow control performed for statically allocated buffer architectures, except that the fraction of buffer memory which a single queue is allowed to occupy is not necessarily $1/n$ for an $n \times n$ switch. The details of maximum usage flow control's operation are presented in Sec. 8.4.2.

The 5×5 switch architecture described above can be used to construct tori of any dimension n. Each processing node has $\lceil n/2 \rceil$ switches associated with it. Each switch forwards packets in two consecutive dimensions. The switch communicating in d_i and d_{i-1} has its host input port connected to the host output port of the switch handling $d_{i+2,i+1}$, and its host output port is connected to the host input port of the switch handling $d_{i-2,i-3}$. Thus, the switches associated with a node form a ring with their host input and output ports; the host processor injects packets into the switch handling dimension $d_{n-1,n-2}$ and receives packets from the switch which handles $d_{1,0}$ (if there are an odd number of dimensions, the last switch in the ring may only handle d_0, leaving two of five ports unused). This technique for handling high-dimensional tori is described in [Dall86].

6.3.2. Performance Evaluation: Dimensional Routing

Figure 6.8 presents results obtained from simulations of a 121-node, two-dimensional (radix equals eleven) torus with bidirectional communication links. For these simulations each node generates thirty-two-byte packets with a uniform distribution (every node is equally likely to be the destination of each packet transmitted), and the time between the creation of packets is random with a geometric distribution. The graphs in Fig. 6.8 show the average latency of packets traversing the network vs. the network throughput. For all torus network results in this chapter, network throughput is presented as the fraction of the bisection bandwidth utilized [Dall90b]. To determine the utilized bisection bandwidth, the simulator measures the throughput (bytes per cycle) per receiver and the number of hops packets traverse before reaching their destinations. The average number of bytes traversing links in the network in each cycle is the product of three terms: the average throughput per receiver, the number of receivers (nodes), and the average number of hops each packet traverses. Dividing this product by the total number of links in the system, yields the average throughput (bytes per cycle) *per link*. For a symmetric network, such as the torus, this number is also the fraction of the bisection bandwidth being utilized.

In the first graph in Fig. 6.8, results for switches with DAMQ and SAFC buffers are presented, with storage for five packets per input port (160 bytes). For the DAMQ buffer, a each queue within a buffer was limited to at most four thirty-two-byte packets. The second graph presents results for larger DAMQ and SAFC buffers; DAMQ buffers with storage for eight thirty-two-byte packets and SAFC buffers with storage for ten. In the case of the DAMQ buffer, results are presented

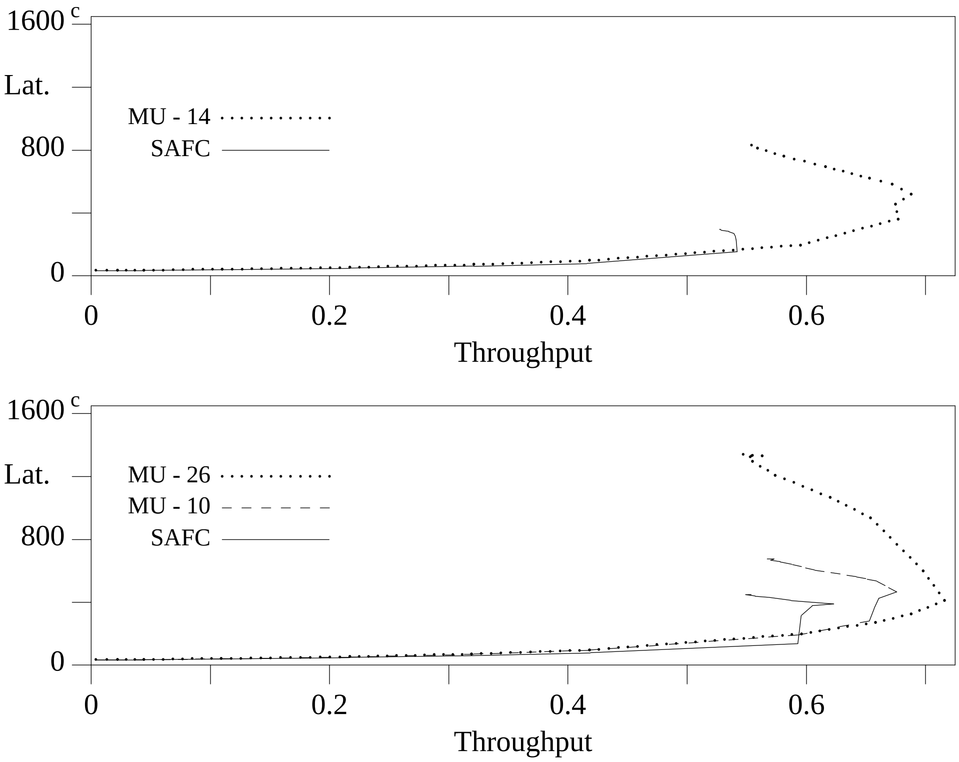

Figure 6.8: *Latency vs. Throughput, 121-Node 2-D Torus.* 11×11 torus network composed of 5×5 switches. Thirty-two byte packets. DAMQ buffers (maximum-usage flow control) and SAFC buffers. Packet destinations randomly chosen, uniform distribution. The top graph is the performance of 160-byte buffers (''MU-14'' is DAMQ buffers using maximum-usage flow control with a threshold of fourteen blocks for the DAMQ buffers). The bottom graph is the performance of 256-byte DAMQ buffers (thresholds of ten and twenty-six blocks) and 320-byte SAFC buffers.

for limiting each packet queue to three packets and to seven packets. In Fig. 6.9, results are given for the larger DAMQ and SAFC buffers in a 441-node torus (a 21×21 two-dimensional torus).

The graphs in Figs. 6.8 and 6.9 demonstrate the performance advantage of dynamic buffer allocation over static. Particularly for the smaller-sized buffers,

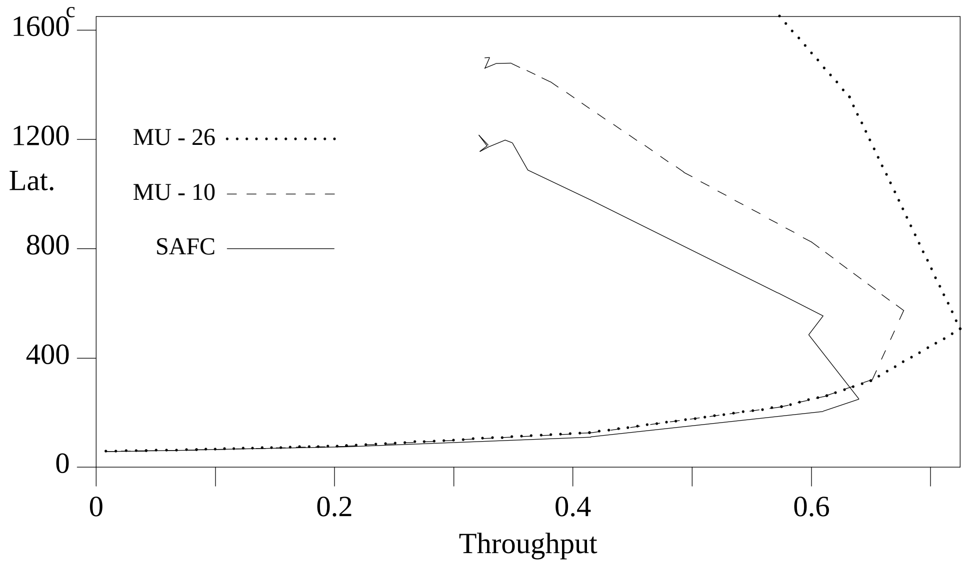

Figure 6.9: *Latency vs. Throughput, 441-Node 2-D Torus.* 21×21 torus network, composed of 5×5 switches. Thirty-two byte packets. Packet destinations randomly chosen, uniform distribution. 256-byte DAMQ buffers with maximum usage flow control (''MU-10'' and ''MU-26'' indicate maximum usage flow control using ten block and twenty-six block thresholds, respectively) and 320-byte SAFC buffers.

having the ability to dynamically allocate the buffer space allows a higher fraction of the available bandwidth to be utilized than does static allocation. This is not a surprising result — dimensional routing and the fact that the switches have a processor interface as one of their five ports both cause the switches to experience non-uniform traffic distributions, even under uniformly-distributed traffic conditions. The performance of a statically allocated buffer could be improved by providing an unequal distribution of buffer space to the queues (favoring those queues / ports which receive large amounts of traffic), but with small buffers this is not a viable option.

Two performance characteristics of torus networks are of interest. The first is that, the larger the network, the lower the bandwidth available to each processor. In Fig. 6.8, DAMQ buffers with 256 bytes per input port achieve a maximum throughput of 0.72. The radix twenty-one torus, with the same buffers, achieves a throughput of 0.73. The fraction of the bisection bandwidth utilized remains approximately the same as the network size increases, but there is a reduction in the bisection bandwidth *per node*. The reduction in bisection bandwidth per node is due to the fact that, for a two dimensional torus with n nodes, as the number of nodes in increased the bisection bandwidth grows as $O(n^{0.5})$.

The second performance characteristic of interest is that torus networks are *reactive*. That is to say, there is an applied load which results in a maximum throughput for the network. When that applied load is exceeded, then throughput is reduced, even though latency continues to increase. The reason that torus networks are reactive whereas omega networks are not is that the torus contains rings of switches — each dimension consists of switches connected in bidirectional rings. The ring topology causes congestion to experience positive feedback. When a link of the ring blocks due to congestion on the next switch, the back-pressure traverses the ring until it increases the degree of congestion at the original point of congestion. While implementing two virtual circuits per dimension breaks the cyclical resource dependency, it does not prevent congestion feedback from occurring.

Evidence for the existence of this positive feedback is provided in Figs. 6.10, 6.11, and 6.12. These figures are a visualization of the buffer utilization on each of the switches of a 121-node torus (256-byte DAMQ buffers at each input

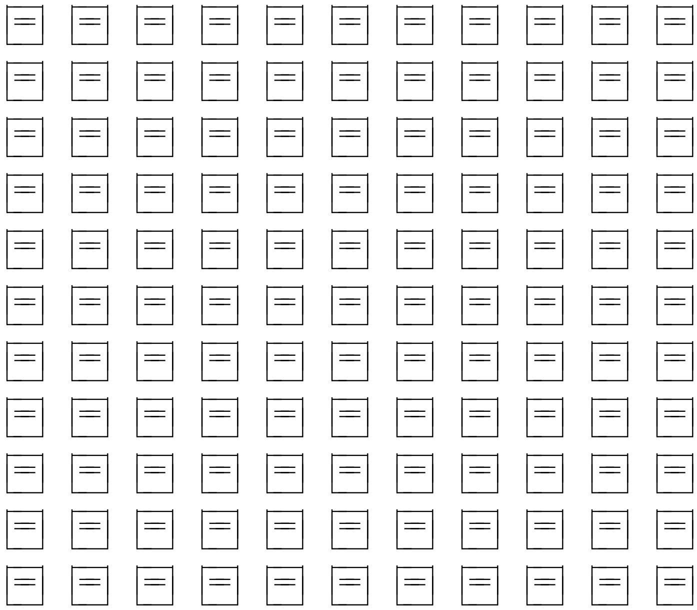

Figure 6.10: *Visualizing Buffer Utilization I.* Torus network operating just below saturation throughput. Differences in the average queue lengths in buffer at the host-interface input port. '=' indicates queues of ≈ equal length, 'p' indicates that Q_1^+ averages at least one block more than Q_1^-, 'P' indicates that the difference is eight or more blocks, and 'n' and 'N' indicate that Q_1^- has the greater average length.

port) operating just below, just in and well into the reactive range. The figures display the average number of blocks in each queue over the course of the simulation for the buffer which receives packets from the host processor on each switch. Since ten of every eleven packets entering an 11×11 torus make at least one hop in d_1, the majority of the host buffer is occupied by the queues Q_1^+ and Q_1^- (packets routed in dimension d_1 in the positive and negative directions). Under uniform traffic conditions, these queues should have the same average length over the course of a simulation. In the figures, the 11×11 matrix represents the torus, with each square of the matrix being a switch in the network. A switch is

N	=	n	N	p	N	N	N	N	N	p
N	p	n	N	p	N	N	N	N	N	p
N	p	N	N	p	n	N	N	n	N	p
N	p	n	N	p	n	n	n	n	N	p
N	P	n	n	P	=	n	n	n	N	p
n	P	=	n	P	p	n	n	=	n	P
n	P	=	n	P	p	n	=	p	n	P
n	P	p	n	P	p	=	p	p	n	P
=	P	p	=	P	P	p	p	p	=	P
p	P	P	=	P	P	p	p	P	p	P
p	P	P	=	P	P	p	p	P	p	P

Figure 6.11: *Visualizing Buffer Utilization II.* Torus network operating at saturation throughput.

marked with a ‘p’ if the average length of Q_1^+ is one buffer block or more greater than the average length of Q_1^-, and a ‘P’ if the difference is eight or more buffer blocks. Similarly, if Q_1^- is greater by one block or more, the switch is marked with an ‘n’, and if the difference is eight blocks or more, the switch is marked with an ‘N’. If the difference between the average queue lengths is less than one block, the switch is marked with an ‘=’.

Figure 6.10 shows the state of the host buffers for the network operating at just below saturation. As can be seen, the average queue lengths (averaged over the duration of the simulation) of Q_1^+ and Q_1^- are within one block of each other on every switch in the network. Fig. 6.11 presents the same data for the network operating just beyond saturation. As can be seen, the average queue lengths have begun to diverge as the applied load is increased. More significantly, the non-

P	N	=	P	N	P	P	N	p	P	N
p	N	N	P	N	P	P	N	=	P	N
P	N	N	P	N	P	P	N	p	p	N
P	N	p	P	N	P	P	N	p	P	N
P	N	P	P	N	P	P	N	P	P	N
P	N	P	P	N	P	P	N	P	P	N
P	P	P	P	N	P	P	N	P	P	N
P	P	P	P	N	P	P	N	P	P	N
P	P	P	P	N	P	P	N	P	P	N
P	P	P	P	N	P	P	N	P	P	N
P	P	P	P	N	P	P	N	P	P	N

Figure 6.12: *Visualizing Buffer Utilization III.* Torus network operating in the reactive region.

uniformity has begun to align itself within each ring in dimension d_1. Finally, Fig. 6.12 visualizes host buffer queue lengths when the network is operating well into the reactive region. It shows that, for most of the rings, every switch in the ring is dominated by packets attempting to traverse d_1 in the same direction. This bistable pattern of congestion strongly suggests the existence of positive feedback in the congestion of a ring of the torus.

In [Dall90b], Dally derives equations which model the performance of k-ary n-cubes which use wormhole routing. The switches in his model are buffer-less; the storage of packets within the network is not modeled. He determined that the maximum throughputs of 1K-node and 4K-node torus networks were 0.36 and 0.35, respectively (throughput measured as the fraction of capacity utilized). This is significantly lower than the throughput results presented in this dissertation for either the DAMQ or SAFC buffers — Dally acknowledges that buffering can and will improve network throughput. A more significant difference between the results presented in [Dall90b] and here is that Dally reported negligible reactivity, stating that the throughput of the network is proportional to the applied load until saturation, at which point it remains constant or drops slightly. This is in direct contrast to our results, where networks can lose 20%-50% of their available throughput due to reactivity. Dally states that the networks he modeled were non-reactive ‘‘because 1) the network is source queued, and 2) messages that encounter contention are blocked rather than aborted.’’ [Dall90b] We believe that the networks he modeled were only marginally reactive either because of the lack of buffering in the network or because his model did not incorporate back-pressure (since Dally’s model is intrinsically buffer-less, it is impossible to distinguish between the two possibilities).

6.3.3. Summary — Dimensional Routing

We have presented simulation results comparing the performance of torus networks (k-ary 2-cubes) constructed of switches implemented with DAMQ and SAFC buffers. These are store-and-forward buffers capable of virtual cut-through. Deadlock-free dimensional routing is implemented via multi-queue buffers. The DAMQ buffer dynamically allocates its buffer space among its multiple queues, and thus can only read from one queue at a time. The SAFC buffer statically allocates its buffer space. Static buffer allocation has two major impacts on the buffer architecture. First, this facilitates reading multiple packets from a single input port buffer simultaneously (see Sec. 3.3), and, second, it reduces the efficiency with which the memory is utilized. Our simulation results indicate that the superior buffer utilization provided by the DAMQ buffer has a greater impact upon network throughput than does the multiple simultaneous reads supported by the SAFC buffer, for the sizes of buffer memory examined.

Previous analyses of the performance of k-ary n-cubes [Dall90b, Agar91, Scot94] focused on modeling networks which use the *wormhole* routing protocol [Dall86]. Wormhole routing is identical to virtual cut-through, except that packets can be transmitted to switches which do not have enough buffer memory to store the entire packet. Typically, switches implemented to support wormhole routing have very little buffer space — the Torus Routing Chip [Dall86] is implemented with two four-byte buffers per input port. Our evaluation has shown that, as has been suggested [Dall90b], providing switches with buffers large enough to fully contain multiple packets (supporting virtual cut through, as opposed to wormhole routing) can significantly boost the saturation throughput of a network. Also, Dally claimed that, in a source-buffered torus network in which packets are blocked rather than discarded, reactivity is minimal or non-existent [Dall90b]. We found that torus networks with buffering for multiple packets at each switch are highly reactive.

6.4. Summary

This chapter presented an asynchronous communication network simulator. This simulator provides more functionality than does the simulator presented in Ch. 4. First, the switches being simulated can be connected in topologies other than an omega network; torus, mesh and hypercube topologies are currently implemented. Second, since asynchronous communication allows variable-length packets, parameters were added to the simulation to allow message length distributions and a maximum packet length to be specified. The third significant feature of this simulator not present in the synchronous network simulator is the ability to segregate the *senders* (packet sources) into groups, giving each group its own packet creation and destination assignment distributions and keeping separate performance statistics for each group.

The first issue which was addressed with this simulator was the performance of the DAMQ buffer in an asynchronous multistage interconnection network. Ch. 4 provided a thorough evaluation of the DAMQ buffer in a synchronous network, but the DAMQ buffer contains features which are only applicable to asynchronous networks. To this end we simulated multistage interconnection networks of various sizes connected in the omega topology, with both constant- and variable-length packets.

Under a uniform load of constant-length packets, the saturation throughput of the statically allocated buffer architectures (SAMQ and SAFC) improves relative to that of the dynamically allocated buffers (FIFO and DAMQ). This is due to the improved buffer utilization provided by virtual cut-through, which partially counter acts the poor buffer utilization caused by static buffer allocation. While the performance of the statically allocated buffers improves, it is still less than that of the DAMQ buffer for small buffer sizes — the SAFC buffer performance does not exceed that of the DAMQ buffer until the buffer memory is increased to one kilobyte.

The performance of statically allocated buffers decreases in the presence of variable-length packets. Statically allocated buffers lose a significant amount of their available buffer space to fragmentation when storing variable-length packets. The DAMQ buffer, on the other hand, minimizes the amount of buffer memory lost to fragmentation, and thus suffers a lesser reduction in its saturation throughput. Our simulations not only demonstrate the efficacy of the DAMQ buffer for asynchronous networks, they also show why the results of simulating synchronous networks provide an accurate prediction of the throughputs achievable by an asynchronous network transmitting constant-length packets.

We also made use of the asynchronous network simulator to examine the performance of the DAMQ buffer in the switches of a two-dimensional torus. We discuss how a multi-queue buffer can support the deadlock-free dimensional routing algorithm presented in [Dall87a] by associating packet queues with the virtual circuits in each dimension and using maximum usage flow control to prevent any one queue from occupying the entire buffer memory. This solution is analogous to the deadlock-free routing mechanism implemented on the Torus Routing Chip [Dall86], which placed two FIFO buffers at each input port to receive packets on the two virtual circuits in each dimension.

Having demonstrated that deadlock-freedom can be provided in a packet-switched network by using multi-queue buffers, the performances of SAFC and DAMQ buffers were compared. SAMQ buffers were not evaluated, since their performance will be no better than that of SAFC buffers. We examined 121-node and 441-node two-dimensional tori with a variety of buffer sizes and, for the DAMQ buffers, a variety of maximum usage flow control blocking thresholds. It was determined that switches with DAMQ buffers provide a significantly higher saturation throughput than do SAFC buffers — in fact, 256-byte DAMQ buffers outperform 320-byte SAFC buffers in the networks we simulated.

Of greater concern than the performance of specific buffer architectures are fundamental issues pertaining to the scalability of torus networks. Our simulations showed (a) that torus networks are highly reactive and that (b) for a torus of given dimensionality, when packet destinations are chosen from a uniform distribution of the possible destinations, the individual processors have access to less network bandwidth as the number of nodes in the network increases. The reactivity of the torus topology can be addressed by implementing flow control mechanisms which prevent the network from operating in the reactive region. The scalability problems of the torus topology must be addressed through communication locality and / or mechanisms such as *express cubes* [Dall91] which reduce the diameter of the communication network.

In this chapter, the asynchronous network simulator is demonstrated to be a powerful tool for exploring communication network performance. In Ch. 8, its ability to group senders is utilized in the exploration of the performance of hop-level flow control mechanisms.