# *Chapter Two*
# Previous Work

This chapter examines the work of previous researchers upon which our research is based. This dissertation examines buffer architectures and flow control mechanisms intended to support high throughput, low latency communication in scalable multicomputers. The current trend is to implement high-performance computer systems as large-scale multicomputers. While a bus-oriented shared memory systems can support the communication needs of systems comprising a small number (tens) of processors, large distributed systems must use messages to transmit data from processor to processor [Ston87]. Sharing information via messages eliminates the dependence upon a single bus for inter-processor communication, removing the limitation the shared bus imposes upon the size of a distributed system.

A multicomputer or multiprocessor whose size is not restricted by an architectural feature is said to be *scalable*. There have been a large number of computer systems built in recent years which are intended to be scalable: the Intel Delta and Paragon computers [Lill91], MIT's J-Machine [Dall87b], *T [Nikh92], and Alewife computers [Agar90], the Cray T3D [Oed], Thinking Machine's CM-5 [Leis92], the BBN Butterfly [Crow85] and Monarch [Rett90] multiprocessors, and others. While these networks utilize a variety of communication topologies and switching techniques, they have one feature in common; a switching network composed of small $n \times n$ switches. Sec. 2.1 presents research which indicates that switches with a small number of input and output ports result in higher throughput,

lower latency networks than switches with a large number of ports. We then discuss work relating to the size and location of the packet buffers on a switch. Sec. 2.3 presents work relating to congestion in communication networks. Sec. 2.4 presents a number of switch architectures which have been proposed, and discusses their strong and weak points. Finally, Sec. 2.5 discusses previous work relating to the use of linked lists to implement packet queues (a key architectural feature of the DAMQ buffer — see Ch. 3).

## 2.1. The Optimal Number of Ports

In ‘‘Performance of Buffered Delta Networks’’ [Kuma84], Kumar and Jump describe experiments modifying the structure of the switches of a delta (interconnection) network. One aspect they examined was the optimal size (number of ports) of a switch. They determined that there is a crossover point between the performance of networks of 4×4 and 2×2 switches as the total buffer space of the interconnection network is varied. They found that networks of 4×4 switches perform better than networks of 2×2 switches when the total buffer space is kept small. This result reflects the fact that a network of 4×4 switches has fewer switch-input-ports than a 2×2 network. With a smaller total buffer space, the 4×4 switch network concentrates that buffer space into fewer buffers and thus has longer queues. Since additional queue space provides diminishing returns, the greater length of the 4×4 switches’ queues is only of value when when the buffers are small. As one increases the total buffer space, eventually the fact that the network of 2×2 switches reduce the amount of output port contention at each switch becomes the dominating factor in network performance.

In his dissertation, Fujimoto [Fuji83] also explored the relationship between switch size and network performance. He used queueing network analysis to demonstrate that, given a fixed amount of bandwidth per switch, a small number of wide ports is preferable over a large number of narrow ports. This result was also reached by Dally [Dall90b], Agarwal [Agar91], and Goodman [Scot94].

## 2.2. Buffer Size and Location

The functional unit which to a great extent determines the structure of an $n \times n$ switch is the packet buffer. Dias and Jump [Dias81] were among the first investigate design tradeoffs for packet buffers in communication switches. They considered multi-stage Delta networks with first-in, first-out (FIFO) buffers at their input ports. They simulated and analyzed the networks and conclude that the buffers need only be large enough to store a small number of packets (four or less) for near-optimal network performance. Their results indicate that the saturation throughput of a network increases with the amount of buffering on each switch, but that most of the benefit of buffering can be achieved with small buffers (buffers that can store on the order of four packets located at each switch input port), and that the average packet latency increases linearly with buffer size. This result has been confirmed in a number of other researchers' works, including our own [Tami88b].

There is a significant body of work dealing with the physical location of buffers on a switch. As was mentioned above, Dias and Jump proposed placing FIFO packet buffers at the input ports of switches. Switches of this type have a fundamental problem with respect to performance; as was discussed in Ch. 1,

output port contention can significantly reduce the performance of switches with FIFO buffers at their input ports. An alternative to input port buffering is to implement a central buffer pool. In this switch architecture the buffer pool replaces the crossbar; packets arriving at the switch are all stored into the pool, and output ports transmitting packets read them from the buffer pool.

To explore the dynamics of a central buffer pool, Irland, in [Irla78], developed an analytical model of a packet switch with a finite buffer space. The goal of his work was to find an optimal buffer sharing policy (flow control) for a switch with a central buffer pool (as opposed to examining the optimal implementation of a central buffer pool). For his model Irland assumed a packet switch with $N$ ports, and a Poisson total arrival rate $\lambda$ which was the sum of the Poisson arrival rates $\lambda_n$ of packets destined for each of the output ports. The switch had space in it for $K$ packets, $K<\infty$, and the number of packets in the buffer destined for any single output was limited to $M$, $M \leq K$. The packets' service time $\mu$ varied exponentially. Finally, the switch modeled was a discarding switch, i.e. if a packet arrives and the buffer is full, or if there are already $M$ packets destined for the same output port that the arriving packet is destined for, the arriving packet is discarded. Thus, finding the optimal buffer sharing strategy entailed finding a value of $M$ such that the number of packets discarded by the switch is minimized.

Irland considered four flow control policies. In the first of these, the *Unrestricted Buffer Sharing* policy, the buffer is allocated on a first-come, first-serve basis until the buffer is completely full (i.e. $M=K$). The second of these, *No-sharing* (or fixed-partitioning), sets $M=K/N$. The third buffer sharing policy considered was *Square-root Sharing.* In this scheme $M=K/\sqrt{N}$. This figure was

derived from modeling a two-ported switch and determining the optimal value of $M$ when $\lambda_1=\lambda_2=1$ (i.e. at the maximum arrival rate). The idea behind this is that packet loses at low throughputs are trivial; the high throughputs are what the buffering scheme should be optimized for. Finally, there was the *Optimal Buffer Sharing* policy. In this scheme, the value of $M$ is adjusted to the optimal value based upon the current packet arrival rates. Actually implementing a buffer policy such as this is virtually impossible to do, but it is valuable as a measuring stick for the other three buffer policies.

Irland's analyses of the four buffer sharing policies were performed using a three-ported switch with buffer spaces of ten, twenty and thirty packet slots. In addition, he used two different types of traffic: balanced and unbalanced. For balanced traffic, the switch received traffic for each of the output ports at the same rate ($\lambda_1=\lambda_2=\lambda_3$), which was varied. For unbalanced traffic, $\lambda_2$ was held at 0.5, $\lambda_3$ was held at 0.7, and $\lambda_1$ was varied.

As was expected, Optimal Sharing had the lowest discarding rate and the highest throughput for every buffer size, traffic type and arrival rate (applied load). In addition, the throughput of the switch was an increasing function of the load, using the Optimal Sharing strategy. Of greater interest was the performance of Unrestricted Sharing, which performed marginally better than the other two sharing policies with balanced traffic, but whose performance fell off drastically when the traffic was unbalanced. Specifically, the Unrestricted Sharing policy displayed a higher mean delay and a lower throughput than either of the other two policies (No-sharing and Square-root Sharing) under unbalanced traffic whose arrival rate (as the sum of the rates for each of the output ports) was $\approx 1$ and over.

The reason for this, as Irland explains, is that the output port which is receiving the majority of the packets cannot operate at greater than 100% of its capacity. Thus, if it is receiving packets faster than it can transmit them, in the Unrestricted Sharing scheme the buffer will fill up with packets destined for the congested output port. This in turn prevents packets destined for the less congested ports from arriving, causing those ports to be idle. This is a phenomena Fujimoto [Fuji83] refers to as buffer hogging — it is the root cause of *tree saturation* [Pfis85a].

Other results of interest are that the Square-root Sharing policy significantly outperforms the No-sharing policy in terms of the packet discarding probability, and the the No-sharing policy displays the lowest mean packet delay (latency). The fact that the No-sharing policy does better in terms of latency comes as no surprise; in a single-switch analysis, the latency of a packet corresponds to how many packets destined for the same output port were in the buffer when the packet arrived. Since the No-sharing policy causes the switch to discard many more packets than the Square-root Sharing policy, it stands to reason that the No-sharing policy will display a lower latency. If the analysis were extended to a full communications network and, upon being discarded, packets were re-transmitted from their source, the mean latency of the network using the Square-root Sharing policy might be lower than that of the No-sharing policy.

Irland's work focused on buffer utilization. The relevance of this in evaluating the buffering scheme of a packet switch is significant. Irland's results indicated that optimal performance is achieved when a switch is implemented with a central buffer pool simultaneously accessible by all of the input and output ports.

In addition, use of the buffer pool by each *output port* should be limited to less than a fixed percentage of the total buffer space (Irland suggests $1/\sqrt{N}$ as an appropriate fraction).

Karol, et. al., in [Karo87], discuss the relative merits of locating the buffers at the input ports versus the output ports. Their switch model consisted of a switch with $N$ input and output ports connected internally by a crossbar. The packet buffers were located at either the input or the output ports; when located at the output ports, packets from multiple input ports could be written into the buffers simultaneously. Packets arrived at each input port with inter-arrival times governed by Bernoulli processes, with an equal probability of being destined for any of the output ports. The buffers themselves were infinitely large, so flow control was not considered.

Their results showed that buffering packets at the output ports significantly shortened the average queue length. In addition, as the number of I/O ports approached infinity, the maximum throughput of a switch with buffers at its output ports approached 95%, while that of a switch with buffers at the input ports never reached 60% throughput, according to their analytical model. The reason for the superior performance of output port buffering is that it reduces the effect of output port contention. For each output port, if there are any packets on the switch destined for it, they will be located in the buffer associated with that output port and will thus never be blocked by packets destined for other output ports.

Thus, queueing packets at the output ports is preferable over queueing at the inputs ports *if packets from all of the input ports can be queued simultaneously*. To do this, Karol et. al. say that the switch must run ‘‘$N$ times as fast as the input and

output trunks.'' In other words, the internal bandwidth of the switch must be as great or greater than the sum of the bandwidths of the input ports for it to be possible to move packets from each input port to the same output port buffer at the same time. The simplest way to do this would be to operate the switch $N$ times faster than the links. However, communication links can operate at the same clock speeds as the silicon components [Scot94]. To be able to support a bandwidth which is the sum of the bandwidths of the input ports, then, the switch must have an internal data path which is $N$ times *wider* than the input ports.

There are significant difficulties with the implementation of both central buffer pools and output port buffering in single-chip VLSI switches (as is discussed in Sec. 3.2). Thus, we found only a single switch in the literature that uses central or output port buffering on its own (McMillen's centrally-buffered switch [McMi86a] — see Sec. 2.4.3). IBM's Vulcan Switch [Stun94] uses a central buffer pool, but also incorporates buffers at the input ports which can store incoming packets. Similarly, the Post Office Chip [Stev86, Davi92] combines an off-chip central buffer pool with input port buffers. Dias and Jump, in [Dias81], looked at placing buffers at the crosspoints of a switch crossbar, creating a separate buffer for each input port $\times$ output port pair. This switch is identical to the SAFC switch discussed in Ch. 3 and in[Tami88b]; placing buffers at the crossbar's crosspoints is logically identical to placing $n$ buffers at each the $n$ input ports, associating each buffer with an output port. McMillen patented two switch implementations whose buffers are located other than at the input port: in [McMi86b] he describes a switch similar to the SAFC switch, and in [McMi86a] he describes a centrally-buffered switch.

## 2.3. Network Contention and Flow Control

In the previous subsections we have alluded to flow control. Flow control is a method used to regulate traffic in the network. It prevents packets from running over each other and controls how fast each advances through the network [Dall90a]. Flow control often involves throttling the flow of data through one or more communication links. Flow control mechanisms are usually included with a network implementation in order to prevent the loss of data due to overflowing packet buffers. This subsection gives a brief overview of some of the previous work in the area of flow control; a more thorough treatment is presented in Ch. 8.

The BBN Butterfly [Crow85] and Monarch [Rett90] multiprocessor systems use *discarding flow control*: when multiple packets contend for the output port of a switch, all but one of the packets is discarded. If buffer memory is provided, packets are discarded only after all of the buffer memory has been utilized. Discarded packets are not permanently lost; the transmitting node retains a copy of all packets it transmits, and will retransmit a packet if it fails to receive an acknowledgement (or does receive a negative acknowledgement, depending upon the implementation). This is an undesirable mechanism in many communication networks because it wastes network bandwidth by requiring packets to traverse links multiple times when they encounter congestion. The vast majority of the tightly-coupled distributed systems in existence today utilize *blocking flow control*: the Intel Paragon [Lill91], the TMC CM-5 [Leis92], the IBM SP-1 [Stun94], Dally's J-Machine [Dall87b], Seitz's Mosaic [Lutz84], the DASH multicomputer at Stanford [Leno92], the Cray T3D [Oed], and others. Under blocking flow control,

when multiple packets contend for a single output port, all but one of the packets are stored in a local buffer. When this buffer becomes full, instead of allowing packets to arrive and then dropping them, the buffer *blocks* transmissions — neighboring switches connected to the full buffer are prevented from transmitting to that buffer. With blocking flow control, packets do not traverse the same link multiple times, but they can experience high network latencies if they encounter congestion in the network. Blocking and discarding flow control mechanisms both prevent the loss of data in the network, but neither promotes high performance in the presence of non-uniform traffic.

A specific form of non-uniform traffic which designers of multistage interconnection networks have explored is a *hot spot* [Pfis85a]. A hot spot is a particular destination node (memory bank) which receives a greater fraction of the packets than the other destination nodes. When the applied load of packets to this hot spot is greater than the bandwidth of the link connecting the network to the hot spot, then congestion will occur.

Pfistor and Norton discovered hot spots in their examination of the behavior of multistage interconnection networks in which the processes are competing for access to global locks or semaphores [Pfis85a]. They discovered that not only are accesses to hot spots particularly slow, but the existence of a hot spot will increase the latency of all of the memory accesses in the system. The reason for this is a phenomena which they dubbed *tree saturation.* Tree saturation is the result of the applied load to a hot spot being greater than the bandwidth to that memory bank. When this occurs, the buffers of the switch immediately preceding the memory bank become full. This in turn prevents the switches preceding the now full switch

from transmitting packets to that switch, and their buffers become full. This eventually forms a tree of saturated switches leading from the hot spot back to all of the processors. When the congestion reaches the senders, then the senders experience a dramatic reduction in the throughput at which they can insert packets into the network.

Along with the discovery of tree saturation, Pfister and Norton developed an algorithm for determining the point at which tree saturation will occur in a given system. Given that

$R$ is the maximum network throughput, per processor ($0 \leq R \leq 1$),

$h$ is the fraction of memory references directed at the hot spot, and

$p$ is the total number of processors, with an equal number of memory banks,

then

$$R = \frac{1}{1 + h(p - 1)}$$

Note that, as either the hot spot percentage or the number of processors increases, the throughput per processor decreases. This is a serious limiting factor in the scalability of multiprocessors.

Pfister and Norton claim that hot spot contention is a major drawback to any multiple-memory bank shared-memory system. It can also be a problem in a message passing multicomputer, where an inordinate number of messages are being sent to a single processor. Thus, finding a solution to this problem will significantly improve the performance and scalability of multiple processor systems. The solution that Pfister and Norton examine is implementing the network with combining switches. When two packets containing references to the

same memory location are queued in the same buffer at the same time, a combining switch will merge the two packets into a single packet to be forwarded to the memory bank. When the reply from the memory reaches the switch, it is split back into two separate packets to be sent to each of the processors which generated the requests. Unfortunately, Pfister and Norton's research indicated that combining switches cost from six to thirty-two times as much to implement as non-combining switches. Of course, simply replicating the interconnection network six times will not necessarily increase the system's performance, because the memory bank bandwidth is still the limiting factor, but if the memory accesses were segregated according to whether or not they were to semaphores, it is possible that the improvement in performance would be greater than that exhibited by combining switches. Phister and Norton discuss this option, but without quantitatively analyzing it concluded that combining switches were a more efficient solution. Given their extreme expense, and the fact that they are only effective in situations where a single memory location is being contested for, we disagree with this assessment.

In [Scot90], Scott and Sohi describe a different approach to preventing hot spots from degrading network performance. A global feedback flow control scheme is proposed for multistage interconnection networks (MINs) operating with a synchronous communication protocol (see Ch. 4). In their scheme, the buffers which feed destination nodes (i.e., the packet buffers in the last switching stage) are monitored. Since the authors assume that the buffers are located at the output ports of the switches, each buffer in the last switching stage is associated with a single destination. When the number of packets in a monitored buffer reaches a *hot*

threshold, the destination associated with that buffer is ‘‘hot’’. Similarly, when the queue length falls below a *not hot* threshold the destination becomes ‘‘cold’’. These changes in destination state are transmitted to all senders over a dedicated flow control feedback network. Congestion is controlled by preventing senders from transmitting packets to hot modules. In Ch. 8, this flow control mechanism is analyzed in detail; we conclude that, while it is effective in addressing hot spots in medium-scale networks (networks consisting of fifty to two hundred and fifty processing nodes), it is not scalable and is thus inappropriate for the systems we envision.

Dias and Kumar [Dias89] have proposed a packet switching flow control mechanism which our evaluations indicate is effective under a number of non-uniform traffic conditions, including hot spots. Their scheme, which we call *destination-based* flow control (Dias and Kumar did not name it) allows only a single packet to any single destination within a buffer at any point time. If a packet arrives at a buffer that is already holding a packet destined for the same address, the packet is discarded, and the previous switch (as opposed to the original source of the packet) re-queues the packet at the tail of the FIFO buffer from which it came. Ch. 8 presents this scheme in detail, discusses the simulation model of the scheme which was used in our evaluations, and provides detailed analysis of why this flow control mechanism performs as it does.

## 2.4. Switch Architectures

The earlier sections of this chapter present previous work in the context of the individual functional units which comprise a communication switch architecture. This section discusses work which presents switch, network, or multicomputer architectures *in toto*. In the discussions below we attempt to analyze the impact the buffer and flow control architectures have on the performance of the systems presented.

### 2.4.1. The Post Office Switch

The Post Office [Stev86, Davi92] is a single-chip communication coprocessor (communications controller) designed to operate in a hexagonal-torus multicomputer. The chip performs topology-dependent dynamic routing. It depends upon virtual cut-through to achieve low-latency communication. There is only enough memory at the six input ports of the switch to receive a packet while it is routed and the switch is arbitrated — when packets cannot be cut through the switch, they are stored in off-chip memory until an output port is available.

Off-chip buffering has a single advantage over on-chip buffering; the input ports occupy less silicon area. This allows the Post Office chip to contain six input and output ports on a single chip, as well as a processor interface (the equivalent of a seventh input and output port) and hardware to handle the packetization and depacketization of messages. The disadvantages of off-chip buffering are associated with the fact that the port to the buffer memory can potentially be a bandwidth bottleneck. Packets have a high probability of being cut through a switch only when the network is lightly utilized. Under conditions of high packet

throughput, packets have a low probability of experiencing virtual cut-through. This means that, when the system is attempting to utilize most or all of the ports of a switch (which is necessary to achieve high system throughput), every input port will be attempting to write packets to and every output port will be attempting to read packets from the off chip packet buffer — simultaneously. If the bandwidth of the port to the buffer memory is not greater than the sum of the bandwidths of the switch ports attempting to access it, then the network throughput will be reduced.

### 2.4.2. The Torus Routing Chip

The Torus Routing Chip [Dall86] employs a routing scheme in which almost no buffering is performed at all. It uses *wormhole routing* to forward packets. Wormhole routing is a technique whereby packets are forwarded through a switch in which there is not enough buffer space to store the packet in its entirety. The packet is cut through the switch in the manner of virtual cut-through [Kerm79], but if the packet is blocked, then only a fraction of the packet is stored on any one switch. In the case of the Torus Routing Chip, only four bytes of a packet can be stored on any one switch. Thus, the packet is strung through the network as though it were a worm crawling from the source to the destination node.

The Torus Routing Chip is a 3×3 switch intended for use in multi-dimensional torus networks. One pair of ports is the processor interface, the other two transport packets in two dimensions of the torus. If a torus of dimension greater than two is desired, multiple Torus Routing Chips are linked by connecting the host output port of one to the host input port of the next to form a higher-dimension switch.

A major contribution of this work is the introduction of a class of deadlock-free routing algorithms for torus networks termed *dimensional routing* [Dall87a]. The cause of deadlock is cyclical resource dependencies. To prevent inter-dimensional cycles, packets are routed in strict dimensional order. E.g. for an $n$-dimension network, packets are routed in dimension $n-1$ first, then $n-2$, etc. To prevent dependency cycles within the rings which comprise the dimensions of a torus, two virtual channels are defined within each link — $V_0$ and $V_1$. When a packet is introduced to a dimension, it travels on the $V_0$ channels. If the packet traverses the wrap-around link (the link connected *node* $r-1$ to *node* 0 in radix $r$ torus), the packet is moved to the $V_1$ channels. By multiplexing each communication link between the two virtual channels and by providing separate buffers at each input port and each output port for them, there are no intra-dimensional cycles either, and deadlock is prevented.

There are four major advantages to using a buffering scheme like that of the Torus Routing Chip. First, it is extremely simple, which means that it is easy and inexpensive to implement. Second, it is small — the buffer requires very little silicon to implement. In the case of the Message Driven Processor [Dall87b], this allows the packet buffers and switching hardware to be placed on the same chip as the host processor. Third, it can operate at high frequencies, allowing it to support high-bandwidth communications. Finally, a system implemented with wormhole routing does not have a hardwired maximum packet length.

A disadvantage of the Torus Routing Chip is the fact that a single packet can occupy many channels simultaneously. This results in a positive-feedback situation; the available bandwidth of the system is inversely proportional to the

relative throughput of traffic traveling upon it. Since having buffers which can store multiple packets does not preclude the use of wormhole routing, and since it is known that the use of packet buffers which can store a small number of packets significantly enhances the performance of communication networks, there appears to be little point in having buffers as small as those of the Torus Routing Chip.

### 2.4.3. The McMillen Switch Architectures

McMillen patented two switch architectures [McMi86a, McMi86b]. The first of these is an $N \times M$ switch using a central buffer. The buffer is a multiport memory with $N+M$ ports. When they arrive at the switch's input ports, packets are routed and placed into the central buffer. The routing tag for the packet is stored in an arbitration memory. The arbitration logic randomly selects routing tags from this memory and forwards the selected packet out the appropriate output port. Central buffering schemes have several problems. These are discussed in detail in Section 3.2, but we will mention the primary disadvantage, which is the size and performance of multiported memories. For a switch with four input and output ports, a central buffering scheme as described by McMillen would require eight-ported memories. Memory cells of this size require large amounts of silicon and operate much more slowly than do one- or two-ported memories. These factors negate any performance benefits which might be gained from the use of a central buffer, and, as discussed in Section 3.2, is is not entirely clear that there are any performance benefits to negate.

McMillen's second patent deals with a buffering scheme similar to the SAFC buffer described in Section 3.2. Each input port has $M$ queues, where each queue

is associated with one of $M$ output ports. Packets arriving at an input port are routed and then stored in the queue associated with the output port to which they are destined. The ramifications of this architecture are discussed in Section 3.2. McMillen also presents an SAFC-style buffer architecture in [McMi80] (the buffer is not given a name in this paper).

## 2.5. Buffer Organization via Linked Lists

As will be discussed in Chs. 3 and 7, the DAMQ buffer uses linked lists to maintain the queues of packets stored within the buffer. Other researchers and engineers have also implemented linked lists to maintain packet queues. As was mentioned previously in this chapter, Fujimoto discusses the implementation of a centralized buffer pool [Fuji83]. Fujimoto organized the packets stored in the central buffer pool into packet queues for each of the output ports by using linked lists of buffer locations. Linked lists are also used by the Intel 82596 LAN coprocessor for efficient flexible storage utilization [Inte89]. However, with the 82596, the pointers are stored with the data in main memory and the linked list data structure is prepared and managed by the general-purpose host processor. The coprocessor simply follows the prepared linked list to obtain the data for transmission or to store an arriving packet (frame). While this organization is sufficient for interfacing to local-area networks, it does not support the high throughput and low latency required for multiprocessor and multicomputer interconnection networks.