Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches

Yuval Tamir, *Member, IEEE,* and Gregory L. Frazier, *Student Member, IEEE*

***Abstract*—Small $n \times n$ switches are key components of interconnection networks used in multiprocessors and multicomputers. The architecture of these $n \times n$ switches, particularly their internal buffers, is critical for achieving high-throughput low-latency communication with cost-effective implementations. We discuss several buffer structures and compare them in terms of implementation complexity, inter-switch handshaking requirements, and their ability to deal with variations in traffic patterns and message lengths. We present a new design of buffers that provide non-FIFO message handling and efficient storage allocation for variable size packets using linked lists managed by a simple on-chip controller. We evaluate the new buffer design by comparing it to several alternative designs in the context of a multistage interconnection network. Our modeling and simulations show that the new buffer outperforms alternative buffers and can thus be used to improve the performance of a wide variety of systems currently using less efficient buffers.**

***Index Terms*— Communication coprocessors, communication switches, interconnection networks, multicomputers, multi-queue buffers, multistage networks, VLSI systems.**

I. Introduction

HIGH-BANDWIDTH low-latency communication between processors is critical to the ability of multiprocessors and multicomputers to achieve high performance by exploiting parallelism. Multiprocessors with a large number of nodes (e.g., greater than 64) use multistage interconnection networks composed of small $n \times n$ switches (typical, $2 \leq n \leq 10$) for communication [2], [8], [17]. Similarly, communication through point-to-point dedicated links in multicomputers [4], [21], [26] relies on communication coprocessors with a small number of ports [3], [22] that function as small $n \times n$ switches with $n - 1$ ports connected to other nodes, and one bidirectional port connected to the local application processor. The design of high-performance small $n \times n$ switches is thus of critical importance to the success of multiprocessor and multicomputer systems.

This paper deals with the design and implementation of a small $n \times n$ VLSI communication switch, focusing on the organization of its internal buffers. The function of the switch is to take packets arriving at its input ports and route them to its output ports. As long as only one packet at a time arrives for a given output port, there will be no conflict, and the packets are routed with minimum latency. Unfortunately, as the throughput goes up, so does the probability of conflict. When two packets destined for the same output port arrive at different input ports of a switch at approximately the same time, they cannot both be forwarded immediately. Only one packet can be transmitted from an output port at a time, and hence one of the two packets is stored at the node for later transmission. The maximum throughput at which the switch can operate depends directly on how efficient the switch is at storing the conflicting packets and forwarding them when the appropriate output port is no longer busy.

An ideal switch has infinite buffer space, but will only buffer (delay) a packet as long as the output port to which the packet is destined is busy. Such a switch can handle n incoming packets while transmitting n packets and can receive and forward the first byte of a packet in a single cycle [12]. In a real implementation, buffers are finite and have a finite bandwidth. This can result in conflicts due to attempted simultaneous accesses to shared buffers and in messages that cannot be received due to lack of buffer space. Packets ready to be transmitted may be blocked behind packets waiting for their output port to free up, and significant delays may be introduced by complex memory allocation schemes required to handle variable length packets.

The research described in this paper is part of the UCLA ComCoBB (Communication Coprocessor Building-Block) project, whose focus is the design and implementation of a single-chip high-performance communication coprocessor for VLSI multicomputer. The ComCoBB chip is, in part, a small $n \times n$ switch, and the problem of designing an efficient buffering scheme for it had to be faced early on in the project. We have developed a new type of buffer for small $n \times n$ switches, called a *dynamically allocated multi-queue* (DAMQ) buffer [23], which will be described and evaluated in this paper. While this buffer was originally developed for use in a multicomputer communication coprocessor, it is equally useful for multistage networks and it is in that context (multistage networks) that the buffer will be evaluated.

In the next section we will discuss the key issues in the design and implementation of $n \times n$ switches. Several alternative switch architectures will be described and the choice of the DAMQ buffer as the key building block of the switch will be explained. As a reference point for evaluating practical switch architectures, we will also describe a theoretical "ideal" switch which achieves many of the desirable characteristics of an

Manuscript received June 15, 1990; revised November 5, 1991. This work was supported by the SDIO Innovative Science and Technology Office, managed by the Office of Naval Research, contracted to the Jet Propulsion Laboratory under task plan 80-2984, and by Hughes Aircraft Company and the State of California MICRO program. G. L. Frazier is supported by a DARPA/NASA Research Assistantship in Parallel Processing administered by the Institute for Advanced Computer Studies, University of Maryland.

The authors are with the Computer Science Department, University of California, Los Angeles, CA 90024.

IEEE Log Number 9105058.

$n \times n$ switch by ignoring the constraints of the implementation technology. The design and micro architecture of the DAMQ switch (a switch which uses DAMQ buffers) will be described in Section III. This description will include the timing of the buffer in the context of the ComCoBB chip and the use of virtual circuits [1], [7] and virtual cut through routing [12]. In Section IV the DAMQ switch is evaluated in the context of a multistage interconnection network by comparing it to three alternative practical switches as well as the theoretical "ideal" switch using Markov models and event-driven simulations. One of the results described in Section IV is that a multistage interconnection network implemented with 4 × 4 DAMQ switches can achieve a maximum throughput which is *forty percent* higher than a network of FIFO (first-in-first-out) switches with the same amount of storage (storage for four packets) in each input buffer.

II. Designing a Switch for a Packet Switching Network

An $n \times n$ switch should be able to accept simultaneous arrival of packets from all of the input ports, while at the same time transmitting packets through all of the output ports. Packets that cannot be immediately forwarded to an output port need to be buffered within the switch. There are three basic conditions where buffering is necessary: 1) the output port through which the packet needs to be routed is blocked by the next stage of the network, 2) two packets destined for the same output port arrive simultaneously at different input ports but the output port can accept only one packet at a time, and 3) the packet needs to be held while the router in the switch determines the output port through which the packet is to be routed.

In order to maximize throughput and minimize latency, *virtual cut-through* [12] should be supported—the switch should not have to wait for a complete packet to arrive before beginning to forward it through an available output port. Efficient utilization of the available storage in the switch for buffering packets implies that as long as there is any storage space available, it should be possible to receive packets from any input port. Furthermore, if variable length packets are used, wasted memory due to internal and/or external fragmentation must be minimized. In order to minimize unnecessary delays, the switch should be organized so that forwarding a packet to an output port will not be delayed by having to wait for the transmission of packets destined for other output ports (more on this last point later in this section).

Buffering may be done by shared central buffers, independent buffers at each output port, or independent buffers at each input port. Complete sharing of available storage by all communication ports results in more efficient storage utilization than static partitioning of the buffer storage between ports. Hence, ideally it would be best to use central buffers where all the available storage can be dynamically allocated to arriving packets from any input port. In the rest of this paper a switch that uses such buffers is called a *centrally-buffered, dynamically-allocated* (CBDA) switch.

Unfortunately, there are fundamental difficulties in producing efficient high-performance implementations of a switch with shared central buffers. In order to achieve high performance, multiple high-bandwidth communication ports must be able to access the central buffers simultaneously. In the worst case, the bandwidth of the interconnection between the buffer pool and the ports must be equal to the sum of the bandwidths of all the ports. Furthermore, for an $n \times n$ switch, the buffers must have $2 \times n$ ports in order to support n simultaneous reads with n simultaneous writes. Multiport memory is undesirable since its implementation is expensive (in area) and leads to poor performance (long access times).

In a switch with shared central buffers it is possible to achieve the required bandwidth without using multiport memory by making the buffer memory and the interconnection (e.g., a bus) to the buffer pool sufficiently wide [22]. However, the need to "assemble" bytes into wide words before transmission increases communication latency (especially in lightly loaded networks) and some of the available bandwidth is wasted when transmitting blocks smaller than the width of the bus. With variable length packets, it is difficult to implement control circuitry that can quickly (within one or two cycles) make memory allocation decisions and minimize internal and external fragmentation.

In addition to implementation difficulties, shared central buffers may also result in some performance problems. Specifically, previous studies have shown that with complete sharing a single congested output port may "hog" most of the available storage in a centralized buffer pool, thus impeding all other communication through the switch [10], [7], [18]. This, in turn, can cause the neighbors, which cannot transmit packets to the full switch's node, to have their buffer fill up, thus converting a single "hogged" buffer into a system-wide problem. The need to limit the buffer space used by any single port increases the implementation complexity [7], [18] and prevents the use of the entire buffer for one path, even when such use would actually increase performance.

The above considerations limit the choice for efficient practical switch implementations to independent buffers at each output port or independent buffers at each input port. If FIFO buffers are used, it has been shown that the mean queue length of systems with output port buffering is shorter than the mean queue length of equivalent systems with input port buffering [11]. This implies that a switch with input buffering requires larger buffers in order to achieve the same probability of buffer overflow. The reason for this is that with buffers at the input ports, packets destined for output ports which are idle may be queued behind packets whose output port is busy, and thus cannot be transmitted. The problem with implementing output port buffering is that, in order to be able to handle simultaneous packet arrivals, the buffers must have as many write ports as there are input ports to the switch. Implementing buffers with multiple write ports increases their size and reduces their performance. Furthermore, if more than one write can occur at a time there is again, as was the case with centralized buffers, a difficult problem of allocating the buffer resources efficiently for variable size packets.

The remaining option is to implement buffering at the input ports. The advantage of input port buffering is that only one packet at a time arrives at the input port so that the buffer

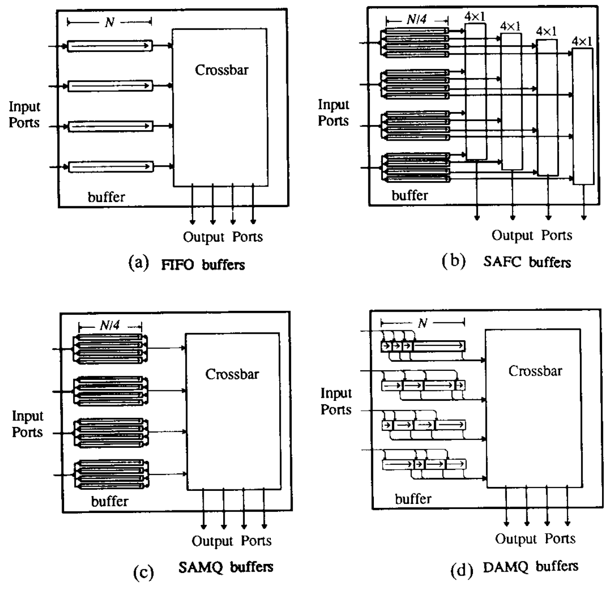


Fig. 1. Alternative designs of switches with input port buffers.

needs only a single write port. Furthermore, if the buffer is managed as a FIFO queue, it is very easy to deal with variable length packets and avoid the memory management problems mentioned above. For these reasons some existing switches use FIFO queues at the input ports [3], [19]. As discussed above, the problem with FIFO buffers at the input ports is that packets may be blocked unnecessarily—a single packet at the head of the queue whose destination output port is busy can block all other packets in that queue from being transmitted even if their destination output ports are idle. Our design attempts to capture the advantages of input FIFO queues but avoids this critical disadvantage.

In the process of designing an efficient buffer for communication switches we have experimented with several types of buffers. The first of these was a FIFO buffer with a single write port and a single read port which allows packets to be forwarded in a first-in, first-out manner. A simple switch with four input and four output ports using FIFO input buffers is shown in Fig. 1(a). The buffers are connected to the output ports by a 4×4 crossbar. It should be noted that the dual-ported storage cells used in the FIFO buffer are needed for virtual cut through and must be used for all buffer types [24].

As mentioned earlier, with FIFO input buffers, output ports may be idle even though there are packets in the switch waiting to be transmitted through those ports. In order to utilize the output ports more efficiently, and thus increase the switch's throughput, packets must be segregated according to the output port to which they have been routed. This can be done using separate FIFO buffers for each of the output ports at each of the input ports [13], [15]. In the case of a four-port switch, this amounts to 16 separate buffers. Since there are multiple buffers at the same input port, a simple 4×4 crossbar will not accommodate all of the possible ways in which packets can be transmitted. Instead, this scheme requires a 16×4 crossbar or, as we have shown in Fig. 1(b), four 4×1 crossbars. We call this switch a *statically allocated, fully connected* (SAFC) switch, and the buffers are SAFC buffers. This name stems from the fact that the input ports have separate lines to each of the output ports (fully connected), and the storage in each input buffer is *statically* partitioned between queues to the output ports.

There are several problems with this design. First, it incurs a large amount of overhead. Four separate switches must be controlled, as opposed to a single-crossbar. While it is much simpler to control a 4×1 switch than it is to control a 4×4 crossbar, using four of them will require replicating the same hardware four times. In addition, each input port will require four separate buffers and buffer controllers. In a VLSI implementation, with chip area at a premium, replicating control hardware is not an efficient use of a scarce resource.

Another problem is that the utilization of the available storage cells in the SAFC switch is not as good as in the FIFO switch. The available buffer space at each input port is *statically* partitioned so that, for a 4×4 switch, only one quarter of the input buffer space is available as potential storage for any given packet. This is in contrast to the FIFO buffers where the entire storage at the input port is available for all arriving packets. Thus, if traffic is not completely uniform, the FIFO buffer will "adapt" to it better than the SAFC buffers. With the SAFC buffers, packets may be rejected by an input port due to lack of buffer space, even though there are some empty buffers at that port for other output ports.

Another problem with SAFC buffers is the complexity of efficiently implementing flow control with *blocking switches*. In such switches [8], the output port of a switch is allowed to transmit only if there is space in the buffer of the destination switch. When the buffer of a switch fills up, the output port(s) of neighbor switch(es) must be blocked—they are notified that they cannot transmit any more messages until some buffer space frees up. With four separate buffers at each input port, information about each of these buffers must be conveyed to the corresponding output port. This is four times the amount of information that is necessary for the flow control of a FIFO buffer. More importantly, if an output port is notified that there is a full buffer in the input port to which it is directly connected, it must "preroute" packets to determine which of the buffers the packet is to be stored in before transmitting it. This pre-routing means that each switch makes the routing decision for the *next* switch in the path of the packet instead of keeping routing decisions local to the switch. While prerouting is possible, it increases the complexity of the routing hardware and causes routing decisions to be made based on information that may be out of date. An alternative flow control mechanism is to forward the packet to the next stage regardless of the state of its buffers and discard the packet if it turns out that there is no space for it. Such a scheme avoids prerouting but requires unacknowledged packets, which may be discarded, to be retained for possible retransmission. The acknowledgment and retransmission can be done either on a stage by stage basis or end-to-end between sender and receiver. In both cases additional hardware is required to store and manage unacknowledged packets. Stage by stage acknowledgment, requires all the switches to be more complex. With end-to-end acknowledgment there is a need to route the acknowledgment signal through multiple hops across the network.

It should be noted that prerouting is necessary not only for flow control but also in order to determine where to store the packet as it arrives at the input port. Without prerouting, incoming packets would have to be stored in a "staging buffer" until the routing decision is made. This again will add to communication latency as well as circuit complexity.

The SAFC buffer can be simplified by implementing the four separate buffers at each input port as a single buffer whose space is divided into four separate queues. This does not reduce the rate at which the buffers can receive packets, since there is only a single input port supplying all four queues. However, it does reduce the number of packets which can be read from the queues associated with an input port (assuming that the buffer has a single read port and a single write port). This switch is shown in Fig. 1(c). It is called a *statically allocated multi-queue* (SAMQ) switch and the buffers are called SAMQ buffers. The space for the output ports is allocated statically, but it is implemented as multiple queues within a single buffer. Since only one packet can be read from a buffer at a time, the interconnection network is a 4×4 crossbar. This eliminates some of the overhead associated with the SAFC switch. However, the problems caused by the need to preroute packets and the inefficiency of statically partitioning the available buffer storage are the same as in the SAFC switch.

What is needed is a buffer which can access the packets destined for each of the output ports separately, but which can apply its free space to any packet. This is the buffer which we have designed, and which we call the *dynamically allocated, multi-queue* buffer (DAMQ). Dynamically allocated, because the space within the buffer is not statically partitioned between the output ports, but is allocated on the basis of each packet received. Multi-queue, because within each buffer there are separate FIFO queues of packets destined for each output port [Fig. 1(d)].

III. Dynamically Allocated Multi-Queue Buffers

In this section we describe the design and micro architecture of the dynamically allocated, multi-queue buffer, as it is used in the ComCoBB chip. It should be noted that an almost identical design can be used for DAMQ buffers in a switch of a multistage interconnection network. The design presented uses similar building blocks to those used in a single chip 2 μm CMOS VLSI layout of a DAMQ buffer [6]. The ability to achieve the bandwidth and latency mentioned in this section has been verified by SPICE circuit simulations of key building blocks [6].

In order to understand the design decisions in the DAMQ buffer, it is necessary to be familiar with a few details regarding the ComCoBB chip. The communications coprocessor being designed in the ComCoBB project has four input ports, four output ports, and a processor interface, all connected via a 5×5 crossbar switch. Each port is autonomous, and independently handles receiving and transmitting packets, so that all the ports can be active at the same time. Each input port has associated with it a buffer and a router, and input and output ports are paired such that there are two unidirectional links between each pair of neighboring processing nodes. The links are eight bits wide and transfer data at a raw bandwidth of at least 40 Mbytes per second. The input port uses a packet-level synchronizer that selects the clock phase used to latch in the data in order to minimize the probability of synchronization failures [24]. The packets in the ComCoBB system are of variable length, from 1 to 32 bytes long, and messages can be made up of multiple packets.

A. Buffer Organization

Multiple queues of packets are maintained within a DAMQ buffer in linked lists. In order to manage linked lists of variable size packets, the buffer is partitioned into 8 byte blocks. Each packet occupies from one to four blocks (the set of blocks which hold a packet is referred to as that packet's *slot* within the *buffer*). For each buffer block there is a pointer register, which points to the next block in the list (*Pointer Registers*, in Fig. 2). The pointers for the linked list are stored in a separate storage array so that they can be accessed simultaneously with accessing the data in the buffer. There are five linked lists in each buffer: a list of packets destined for each of the three output ports with which the input port is not paired, a list of packets destined for the processor interface, and a list of free (currently unused) buffer blocks.

Linked lists are also used by the Intel 82596 LAN coprocessor for efficient flexible storage utilization [9]. 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.

When a packet arrives at an input port, a block is removed from the free list and used to store the first 8 bytes of the packet. The block is then linked to the tail of the list for the output port to which the packet is routed. When a packet is transmitted through the crossbar switch from the input buffer to an output port, the blocks it occupies are returned to the free list. In order to manage the linked lists, each buffer has five *head* and *tail* registers (Fig. 2). The head register points to the first block of the first packet of its linked lists, and the tail register points to the last block of the last packet in the list. Except for the buffer storage array and the control finite state machines, all the hardware required to implement dynamic buffer allocation and multiple queues is contained within the box marked "B" in Fig. 2. The functional blocks within "A" are needed for low latency handling of virtual circuits and variable length packets. The hardware in "A" is independent of the DAMQ buffer, and would accompany any other buffer configuration.

B. Packet Reception and Transmission

The actions required to receive, forward, and transmit a

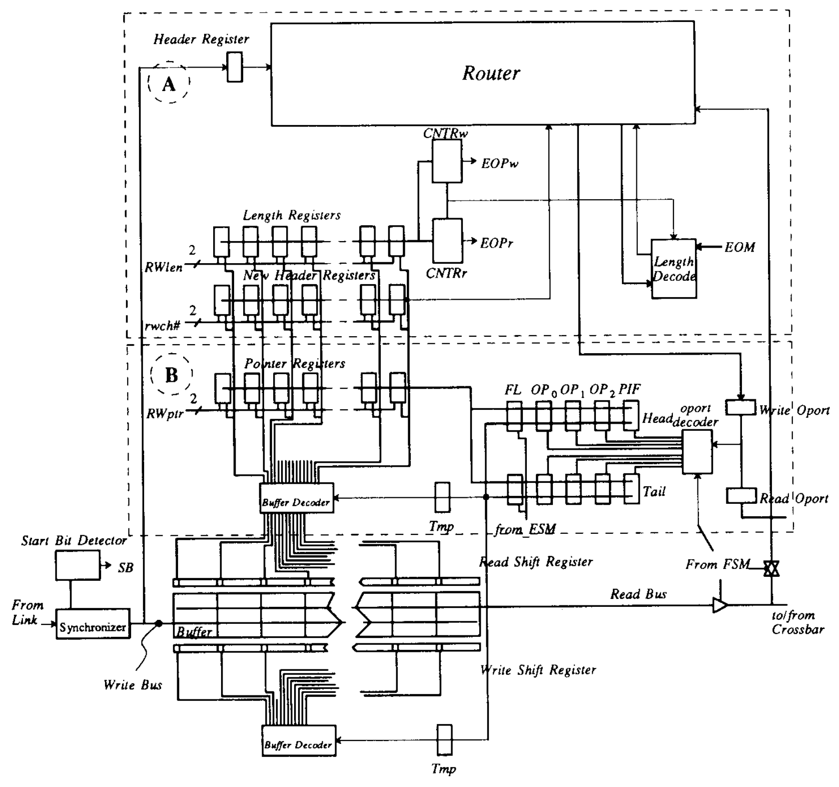


Fig. 2. The dynamically-allocated multi-queue buffer, as used in the ComCoBB chip.

packet depend on the packet format and basic protocol of the ComCoBB system. Packets consist of a *header byte*, a *length byte* if the packet is the first packet of a message, and then from 1 to 32 bytes of data. The ComCoBB uses a form of *virtual circuits* [1], [18] to route packets through the network [25]. Each physical link is logically divided into multiple *virtual channels*. Before data are transmitted, a path of virtual channels is established from sender to receiver. While a packet is transmitted through a link, its header byte specifies the virtual channel number used by the packet. In most cases, routing consists of a single read from a small table at the input port. When a packet is forwarded to an output port, its header byte may be changed to reflect the fact that it will use a different virtual channel on the outgoing link than it had used on the incoming link [25].

The *router* (shown as a "black box" in Fig. 2) uses the header byte to determine the packet's output port and new header (and length, if this is not the first packet in the message). Each packet is preceded by a "start-bit," which is used for synchronization [24]. The header byte is transmitted in the clock cycle immediately following the start-bit, and the rest of the packet is transmitted at a rate of one byte per clock cycle following that.

1) Packet Reception: Because each byte of the packet must pass through the synchronizer before entering the buffer, there is a full clock cycle delay between the signaling of packet arrival (*SB* in Fig. 2) and the actual arrival of the header byte, with the rest of the bytes of the packet following in the succeeding clock cycles. While the router is dealing with the header byte, the first byte(s) of the packet are stored in the block which is at the head of the free block list. As the packet is being stored in the first free block, the router stores the packet's length in a length register associated with that block (and into the write counter), and notifies the buffer controller which of the output ports the packet is destined for. In addition, the router creates a new header byte for the packet and stores it in another register associated with the packet's first block.

Once the buffer controller is notified of the linked list into which the packet should be placed, it sets the pointer register of the block currently pointed to by the tail register of that linked list to point to the first block of the current packet. It then sets the tail register to point to the packet's first block. When the transmission reaches 8 bytes, and the first block is filled, the next block in the free list (not necessarily adjacent within the buffer) is used to store the next 8 bytes. The same sequence as given above is used to place the packet's second block at the end of the queue, and to point the tail of the list at that block. The end of the packet (*EOP*) is detected when the write counter reaches zero.

2) Packet Transmission and Virtual Cut Through: The crossbar is controlled by a central arbiter which determines which buffers are to be connected to which output ports. It makes this decision based upon data it receives from each of

TABLE I
VIRTUAL CUT THROUGH IN FOUR CLOCK CYCLES

Cycle	Phase	Action
0		The start bit arrives in either phase 0 or phase 1. The start bit detector notifies the synchronizer of the proper phase in which to receive packets and notifies the FSM controlling the buffer that a packet is arriving.
1		The header byte arrives in the synchronizer is either phase 0 or phase 1, but is not yet available.
2	0	The synchronizer releases the header byte, which is latched by the header register, for use by the router. The rest of the packet will be released from the synchronizer a byte at a time, at phase 0 of each clock cycle.
	1	The router determines the output port the packet is to be sent using a local table, and sends this information to both the arbiter (for access to the crossbar) and the controlling FSM's. The router also generates a new header for the packet, which is stored in a register associated with the packet's slot.
3	0	The byte specifying the length of the message is released from the synchronizer. It is loaded into the router, using the header byte to index a table contained within the router.
	1	Latch the result of the crossbar arbitration. The packet's length is passed through the length decoder, and latched into the length register associated with the packet's slot and into the write counter.
4*	0	The synchronizer releases the first byte of the packet itself, which is stored in the buffer. If the packet is only a single byte long, the write counter signals EOP. The new packet header is transmitted across the crossbar to be latched at the output port, and the output port generates a start-bit.
	1	The output port latches the new packet header. The bit in the write address shift register is shifted up a location.
5	0	The output port transmits the header byte to the next switch. The packet's length is sent through the crossbar, and is also loaded into the read counter. The second byte of the packet is written into the buffer slot.
	1	The output port latches the packet length. The bit in the write address shift register is shifted up a location. An on-bit is loaded into the read address shift register, at the beginning of the packet's slot, prepared to read the packet's first byte for transmission across the crossbar on the next clock cycle.

*The new start bit is generated here; thus, four cycle turn-around time.

the buffers, so that a buffer is never connected to an output port for which it does not have any data. When a buffer is connected to an output port through the crossbar, it uses the head register of that output port's linked list to locate the first block to be transmitted. The first byte transmitted is the new header byte (the start bit is generated automatically by the output port). While the length of the packet is loaded into the read counter, the head register of that list is set to the value stored in the pointer register associated with the first block. The packet is transmitted until the read counter reaches zero, using the pointer registers associated with each block to find the next block to be transmitted. The list's head register always points to the next block to be transmitted, or to the first block in the free list if its linked list is empty.

With virtual cut through [12] the switch begins to forward a packet before it has completely received it. The amount of time a packet is delayed by the switch using virtual cut through depends on the availability of the output port and the speed with which the packet is routed. This delay is independent of the packet's length. We refer to this latency as the "turn-around" time: the time from the arrival of the start bit until the time that the switch transmits the start bit for the same packet to the next switch. Since the buffers can handle simultaneous reads and writes and the header and length registers are stored in a separate memory, the turn-around time can be as low as four cycles (see Table I).

One of the critical factors that facilitates the short-turn-around is the fact that, when a linked list is empty, its head register is set to point to the first block of the free list. The only time a packet is "cut through" the buffer is when the linked list to the appropriate output port is empty and that output port is currently idle. When a packet arrives at an input port and it can be cut through, the head register for the appropriate linked list is already pointing to the correct block. Thus, the process of receiving can be overlapped with the process of transmitting, resulting in a fast cut through.

3) Buffer Implementation: Our buffer design is driven by the high bandwidth of the ports. The links between ComCoBB chips are 8 bits wide and transfer data at a raw bandwidth of at least 40 megabytes per second (one byte per clock cycle, with 2 μm CMOS implementation) [6]. This high rate of transfer is achieved by using packet-level synchronized communication [24]. The clock phase used to latch the incoming bytes is determined by the synchronizer based on the packet start bit. With a small maximum packet size (32 bytes), the crystal-controlled local clock will not "drift" to the point of causing a synchronization failure during the reception of the rest of the packet [24]. The buffer pool is an $8 \times n$ array of dual ported static memory cells, where n is a multiple of eight (since the buffer must contain an integer number of blocks). Two 8-bit buses traverse the memory array: one carries data from the synchronizer (*write bus*), and one transmits data to the crossbar (*read bus*). Based on the simulation results discussed in Section IV and a maximum packet size of 32 bytes, we expect the size of the buffer to be between 64 and 128 bytes [6].

In order to achieve high transfer rates, address decode time is eliminated by using shift registers to address the storage cells instead of traditional addressing mechanisms [24]. Along both sides of the static cell array are a series of 8-bit shift registers, one shift register for each buffer block (Fig. 2). There are separate shift registers for reading from and writing to the buffer, making the two operations completely independent. For either reading or writing, there is never more than a single shift register enabled at a time, and never more than a single bit of that shift register which is "on." It is the "on" bit of the enabled shift register which addresses the buffer, enabling the

8 static cells associated with it to sequentially read/write their data. To write into the buffer, the "on" bit is set to the first byte of the initial buffer block in which we will receive the packet, and then each cycle a byte of the packet is written into the buffer, and the bit is shifted to receive the next byte. When the last byte of a block is used, the shift register associated with that block is disabled, and the shift register associated with the next block is enabled, with its "on" bit pointing to the first byte of the block [24].

As mentioned earlier, in order to efficiently handle variable length packets, the buffer is partitioned into 8-byte blocks. Support for the linked list organization is provided by a pointer register which is associated with each block. The value of the pointer register is the number of the next block in the list. One important design parameter is the size of the blocks. At one extreme, a block can be the size of the largest possible packet (32 bytes). This choice would result in inefficient utilization of the buffer memory since small packets (e.g., four bytes) would use up an entire block, wasting large amounts (e.g., 28 out of 32 bytes) memory. On the other hand, the blocks should not be too small since there is some overhead associated with each block. For each block there is a pointer register (for the linked lists) as well as a length register and a new header register (to facilitate fast virtual cut through) because any block can be the first block of a packet. Thus, smaller blocks require more chip area for the same amount of buffer space. Smaller blocks also require more processing by the receiving and transmitting finite state machines for pointer manipulation. 8-byte were chosen as a compromise between the overhead of small blocks and the internal fragmentation in large blocks.

The buffers are locally controlled, to allow the ports to operate concurrently and independently. Each input port has three finite state machines associated with it, each FSM handling a separate facet of the buffer management [6]. The first is the *buffer manager*, which handles receiving new packets and assigning them to free buffers. Second, there is the router, which does the routing and updates the routing information. Finally, there is the *transmission manager*, which, when notified by the arbiter that the buffer has been connected to an output port, transmits the packets and returns the freed blocks to the free list. Because these state machines exist as separate entities, interacting via registers and a few shared signals, each buffer can both receive and transmit packets simultaneously at the highest bandwidth possible. The FSM's maintain the organization within the buffer via the head and tail registers of the five linked lists and the pointer registers associated with each block. The FSM's are synchronized so that they will not attempt to read/write from/to the same register or use the same bus simultaneously, and they will never read and write from/to the same byte of memory at the same time.

IV. Buffer Performance Evaluation

In our performance evaluation of the different buffer types we considered both *discarding switches*, which discard packets that attempt to enter a full buffer [2], and *blocking switches* which block the transmitter from sending to a full buffer [7], [8]. In order to evaluate the DAMQ buffer, we compared the performance of DAMQ switches to the performance of FIFO, SAMQ, and SAFC switches and to the performance of hypothetical CBDA switches (see Section II). Markov models were used to evaluate 2×2 discarding switches. For 4×4 switches, Markov modeling is difficult due to the large number of states. For example, for a 4×4 switch with storage for six packets at each input port, the model has more than three million states. Hence, the evaluation of both discarding and blocking switches was done using event-driven simulation.

As discussed in Section II, there are fundamental difficulties in producing efficient high-performance implementations of switches with shared central (CBDA) buffers. We include these switches in our evaluation studies not as a practical implementation option but as hypothetical "idealized" switches whose performance is a yardstick against which the performance of practical switches can be measured. The CBDA switches are "ideal" only in the sense that they provide the most efficient utilization of buffer storage as well as non-FIFO access to stored packets. If traffic is not uniform (some destinations receive more packets than others), CBDA buffers are susceptible to buffer hogging (see Section II) and may actually result in lower performance than switches with other buffer organizations. Since most of the evaluation in the paper is based on uniform traffic, the CBDA switches do provide a basis of comparison with "idealized" performance characteristics.

It should be noted that an additional significant implementation problem with shared central buffers is the need for complex flow control. In our idealized simulations, when there are fewer buffer slots than inputs ports (so that not all the input ports can receive packets), the input ports which do not have packets pending for them are blocked first, and the rest are prioritized according to the length of time the packets had spent in the preceding switch. In an actual network implementation, the information about pending packets in other switches would not be available to the switch. However, this is not of concern to us since the CBDA switch is not being considered as an implementable switch.

A. Evaluation of 2×2 Discarding Switches Using Markov Models

We have evaluated the performance of individual 2×2 discarding switches using Markov models. An entire network was not modeled due of the intractable number of states which would result. Several simplifying assumptions were made: 1) fixed length packets, 2) uniform distribution of packet destinations, 3) when there is contention for an output port, the input port that is allowed to transmit is chosen randomly, and 4) synchronous store-and-forward operation of the switch so that during each *stage cycle* [27] packets either completely arrive or completely depart. Since we assume a uniform packet size, the packet slots are made up of a constant number of blocks, and will therefore be used as the unit of buffer storage.

The model used to generate the Markov state transition graph for each switch allows the switch to simultaneously receive and transmit messages during each stage cycle. A packet is discarded if and only if it arrives at a full buffer

TABLE II
THE PERFORMANCE OF A SINGLE 2 × 2 DISCARDING SWITCH. RESULTS OBTAINED USING A MARKOV MODEL. SHOWN IS THE PERCENTAGE OF DISCARDED PACKETS FOR VARIOUS APPLIED INPUT TRAFFIC RATES (THE FRACTION OF THE RAW INPUT PORT CAPACITY)

Buffer Type	Buffer Size per Port	Percentage of Discarded Packets Versus Applied Input Traffic Rate							
		0.25	0.50	0.75	0.80	0.85	0.90	0.95	0.99
FIFO	1	1.7	7.1	15.5	17.4	19.3	21.2	23.1	24.6
	2	0+	1.2	8.7	11.4	14.5	17.8	21.3	24.2
	3	0+	0.2	6.1	9.2	13.0	17.0	21.0	24.2
	4	0+	0+	4.7	8.1	12.3	16.7	21.0	24.2
	5	0+	0+	3.8	7.5	12.0	16.7	21.0	24.2
	6	0+	0+	3.2	7.1	11.9	16.6	21.0	24.2
SAMQ	2	0.9	4.7	11.3	12.9	14.5	16.1	17.8	19.1
	4	0+	0.3	3.0	4.2	5.5	7.1	8.9	10.5
	6	0+	0+	0.9	1.5	2.4	3.7	5.4	7.1
SAFC	2	0.8	3.8	9.1	10.5	11.9	13.4	15.0	16.3
	4	0+	0.2	2.0	2.8	3.8	5.1	6.6	8.1
	6	0+	0+	0.5	0.9	1.5	2.4	3.8	5.2
DAMQ	2	0+	0.6	4.8	6.4	8.3	10.5	12.9	15.0
	3	0+	0+	1.4	2.4	3.9	5.8	8.3	10.6
	4	0+	0+	0.4	0.9	1.8	3.3	5.6	8.1
	5	0+	0+	0.1	0.4	0.9	2.0	3.9	6.5
	6	0+	0+	0+	0.1	0.4	1.2	2.8	5.4
CBDA	2	0+	0+	1.8	3.0	4.6	6.7	9.3	11.8
	3	0+	0+	0.2	0.5	1.2	2.6	4.9	7.5
	4	0+	0+	0+	0.1	0.3	1.1	2.9	5.4
	5	0+	0+	0+	0+	0.1	0.4	1.8	4.1
	6	0+	0+	0+	0+	0+	0.2	1.1	3.3

which is not currently transmitting a packet. This corresponds to the hardware described earlier, where the buffer supports simultaneous read and write operations. The operation of the switch is equivalent to a switch that alternately sends and receives packets, where the state of the switch is defined by the packets in its buffers *after* packet transmission but *before* packet reception. With this model, a packet is discarded when it arrives at a full buffer. Thus, if one packet arrives at the switch, the probability that it will be discarded is

$$P(discard) = \sum_{i \in I} P(i) + 0.5 \sum_{j \in J} P(j)$$

where I is the set of states in which both input ports are full and J is the set of states in which one of the input ports is full. Packets arrive at each input port of the switch with a probability equal to the applied traffic rate and with equal probability of exiting either output port. The transition probabilities between states is calculated in two stages: packet arrival puts the switch into an intermediate state, and then arbitration and transmission moves the switch to the next state. The product of these probabilities is the probability of transferring from one state to the next via a particular intermediate state. The sum of all such transfers is the total probability of that state transition.

All four practical switches as well as the CBDA switch were evaluated at varying applied traffic rates and different buffer sizes. The applied traffic rate corresponds directly to the probability of a packet arriving at an input port, i.e., for a switch operating with 70% applied input traffic rate each input port has a probability of 0.70 of having a packet arrive at each long clock cycle. From our model we could determine the probability that a given packet arriving at a switch will be discarded for a given level of traffic. The results are presented in Table II. Since the SAMQ and SAFC switches statically allocate buffer space to each of the output ports, they can only have an even number of packet slots in each buffer.

As shown in Table II, the switch with DAMQ buffers performs better (lower probability of discarding) than any of the other practical switches with the same amount of storage at any rate of traffic (except for the SAFC switch with 0.99 applied input traffic rate). It should be noted that a DAMQ switch with space for three packets per input buffer discards as few or fewer packets than the FIFO switch with space for up to six packets for all traffic rates. Furthermore, the DAMQ switch performs *significantly better* than the FIFO switch for high traffic rates. The savings in chip area due to this dramatic decrease in storage requirements is several times greater than the area for the extra control circuitry needed for the DAMQ buffer (ten head/tail registers, one pointer register per block, and more complex FSM's) [6].

For light traffic and only two slots per buffer, the FIFO switch performs better than the SAMQ and SAFC switches. Under these conditions the probability of discarding is determined by available storage and the FIFO buffer, having a single pool of slots instead of statically partitioned storage, delivers better performance. This effect is overshadowed by the unnecessary blocking of packets due to FIFO handling when the traffic rate is high or when there are more than four slots per buffer. In general, the SAMQ buffer performs almost as well as the SAFC buffer, indicating that the additional throughput provided by fully connecting the inputs with the outputs does not provide a significant boost in performance for synchronous uniform traffic.

The benefits of non-FIFO access to the buffers are demonstrated by the fact that the FIFO buffers perform significantly worse than the three other buffer types for traffic rates above 80% and a wide range of buffer sizes. Furthermore, the performance of the FIFO buffers cannot be improved by simply increasing the buffer size—for example, with a traffic rate of 90%, increasing the FIFO buffer size from three to six slots does not significantly reduce the discarding rate. On the other hand, SAMQ, SAFC, and DAMQ switches with four slot buffers have significantly lower discarding rate than the FIFO switches with six slot buffers. Thus, it is not always possible to trade off implementation and control complexity for increased storage. FIFO switches perform poorly at high traffic rates *regardless of their buffer size.*

We have previously described the benefits of the DAMQ buffer over the SAMQ and SAFC buffers in the areas of implementation, flow control, and routing. The performance advantage of the DAMQ compared with the SAMQ and SAFC buffers is based on more efficient use of storage. As shown in Table II, the DAMQ switch consistently performs significantly better (lower discarding rate) than a SAMQ or SAFC switch with equal size buffers. This effect becomes more pronounced as the partitioning of SAMQ and SAFC buffer increases (i.e., for $n \times n$ switches with larger n).

B. Switch Evaluation Using Omega Network Simulation

We have simulated communication on a 64×64 Omega network [14] constructed from three stages of 4×4 switches. The network connects 64 processors (message generators) to 64 memory modules (message receivers). In our simulation we assumed synchronized, store-and-forward message transmissions, where packets are transmitted/received instantaneously once every *stage cycle* [27]. Fixed length packets are assumed. Both blocking and discarding protocols were used for packet flow control. Most of our network simulations were done for uniformly distributed traffic, where packets have an equal probability of being sent to any one of the possible destinations. We have also studied "hot-spot" traffic [16], in which a disproportionate percentage of the messages from each source are sent to a single common destination.

An important issue is the design of communication switches is the scheme used to arbitrate between multiple packets which require conflicting resources in order to be forwarded. The switches have to arbitrate between multiple buffers with packets destined for the same output port. The SAMQ and DAMQ switches also need to arbitrate between packets from the same buffer which were routed to different output ports. The arbitration scheme used in the evaluation involved examining the buffers of a switch one at a time, transmitting packets from the longest unblocked queue is the buffer. To support fairness, the order in which buffers are examined is not fixed—a modified round-robin priority scheme is used. In successive arbitration "rounds" each buffer in turn is the first buffer to be examined. If a nonempty buffer has the top priority but is unable to transmit (for example, due to a full buffer at the destination node), the buffer retains its top priority for the next round. To maintain fairness within the buffers, a stale count is used on the queues [22] to determine which queues within a buffer have held packets for a long period of time and should therefore get top priority.

1) Networks of Discarding Switches: The results of simulating a network of discarding switches [2] under uniform traffic are presented in Table III. The table shows the percentage of packets which are discarded for various rates of introducing packets to the network. The applied input traffic rates are expressed as the fraction of the network's raw input port capacity. For each applied input traffic rate R, with a resulting discarding rate D, the throughput of the network is $R \times (1 - D)$.

The results in Table III show that, for a given buffer size and wide range of applied input traffic rates, the DAMQ switch network has significantly lower discarding rates than networks with FIFO, SAMQ, and SAFC switches. Furthermore, the maximum achievable throughput is significantly higher with the DAMQ switch. At low applied input traffic rates, the FIFO switch performed *better* than the SAMQ and SAFC switches due to the four-way static partitioning of the buffer space in the SAMQ and SAFC buffers. However, for high traffic rates the advantages of non-FIFO handling of packets dominate, and both SAMQ and SAFC switches achieve lower discarding rate and higher maximum throughput than the FIFO switches. In all cases, the hypothetical CBDA switch performs better than the other switches, thus demonstrating the value of complete sharing of storage and arbitrary random access to any packet buffered in the switch.

2) Networks of Blocking Switches: A multistage interconnection network composed of blocking switches can be characterized by the relationship between packet latency and throughput. In general, as throughput increases, so does the latency. For low throughputs, before the network approaches saturation, latency grows very slowly with increasing throughput. As the throughput approaches saturation, the latency increases rapidly. Near saturation, small increases in throughput are accompanied by large changes in latency. This relationship between latency and throughput is shown in Fig. 3 and elsewhere [5], [16].

We have simulated 64×64 Omega networks of blocking switches using all five switch types and several buffer sizes under uniform traffic with a wide range of traffic loads. The results of our simulations are shown in Table IV. The throughput is reported as the fraction of the aggregate raw link bandwidth. This is the fraction of the maximum network throughput that would be achieved if all the network links were transmitting at all times (this would be possible if there were no conflicts in any of the switches). The latency is the number of stage cycles from the moment the packet is created to its delivery at the destination. The minimum latency through the Omega network is three stage cycles. At each sender, the interval to the next packet creation is calculated from the time the current packet enters the network.

Table IV shows that for low throughput rates all the switches achieve nearly the same latencies. Furthermore, for particular buffer type, as long as the throughput is well below saturation,

TABLE III
THE PERFORMANCE OF A 64 × 64 OMEGA NETWORK COMPOSED OF 4 × 4 DISCARDING SWITCHES. RESULTS OBTAINED FROM SIMULATION. UNIFORM TRAFFIC. SHOWN IS THE PERCENTAGE OF PACKETS DISCARDED FOR VARIOUS APPLIED INPUT TRAFFIC RATES (THE FRACTION OF THE NETWORK'S RAW INPUT PORT CAPACITY)

Buffer Type	Buffer Size per Port	Percentage of Packets Discarded Versus Applied Input Traffic Rate								Maximum Throughput
		0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	
FIFO	1	1.5	5.8	12.1	19.6	27.0	33.9	40.3	45.8	0.45
	2	0+	0.2	1.5	4.9	11.2	19.6	28.0	35.7	0.52
	3	0	0+	0.2	1.3	5.2	13.4	22.3	31.1	0.55
	4	0	0+	0+	0.4	2.5	10.3	18.6	27.2	0.57
	8	0	0	0	0+	0.2	5.3	13.6	24.0	0.61
SAMQ	4	0.4	1.9	4.6	8.4	13.2	18.6	23.9	29.1	0.61
	8	0+	0+	0.1	0.4	1.2	3.1	6.2	10.5	0.78
SAFC	4	0.4	1.5	3.6	6.4	9.9	14.2	18.6	23.2	0.67
	8	0	0+	0.1	0.3	0.8	2.0	3.9	6.9	0.84
DAMQ	2	0+	0.1	0.4	1.8	5.0	10.7	17.3	24.5	0.63
	3	0	0+	0+	0.1	0.7	3.0	7.2	13.3	0.72
	4	0	0	0+	0+	0.1	0.7	3.9	9.6	0.78
	8	0	0	0	0	0	0+	0+	0.7	0.88
CBDA	1	0+	0.2	1.1	4.4	10.5	18.7	26.8	34.5	0.53
	2	0	0	0	0+	0.1	1.3	4.7	10.9	0.73
	3	0	0	0	0	0+	0.1	0.8	3.5	0.82
	4	0	0	0	0	0	0+	0.1	1.1	0.86
	8	0	0	0	0	0	0	0	0+	0.93

the buffer size does not have a significant impact on the latency. For any given buffer size the DAMQ switch network can achieve significantly higher maximum throughput than networks with FIFO, SAMQ, or SAFC switches. For example, with a buffer size of four packet slots per input, the maximum throughput of a network composed of DAMQ switches is at least 30% higher than a network composed of any of the other three practical switches. As shown in the Table IV and in Fig. 3, for these same switches, at a throughput of 0.50, the DAMQ network resutls in lower latency than with the other three practical networks. This difference in latency is due to the fact that the other switches are at or near their saturation throughputs. At lower throughputs (below 0.40) the average latencies with all the networks are very close to each other. Hence, the major advantage of the DAMQ switches is their ability to provide good performance for high throughput rates.

As shown in Fig. 3, the latency of the network of SAMQ and SAFC switches with four packet slots per input buffer does not increase significantly near saturation as it does with the other switches. The reason for this is that with four packet slots per buffer, each queue has only one buffer slot. Hence, once a packet enters a switch, it is forwarded the next time its queue gets priority from the arbiter. With the other buffer types and with larger SAMQ and SAFC buffers, the packet may be queued behind several other packets waiting for the same output port.

The results in Table IV are similar to the results of modeling and simulating discarding switches—in all cases the benefits of non-FIFO handling of packets are in better performance under high network loads. At low throughputs, the FIFO switches perform as well as the SAMQ, SAFC, and DAMQ switches. However, compared to FIFO buffers, the multi-queue buffers can provide higher maximum throughput as well

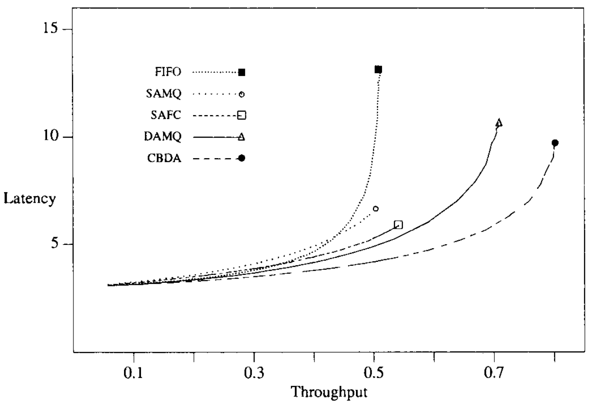


Fig. 3. The performance of a 64 × 64 Omega network composed of 4 × 4 blocking switches with four packets slots per input port buffer. Results obtained from simulation. Uniform traffic.

as lower latency at high throughputs. Increasing the size of FIFO buffers beyond three or four slots results in very small increases in the maximum network throughput [5]. On the other hand, with buffers that provide non-FIFO handling of packets, the maximum network throughput can be increased far beyond what is achievable with FIFO buffers. This result is seen in Fig. 4 which shows the diminishing returns obtained from increasing the size of FIFO buffers. This figure also shows that the DAMQ buffers with only three packets slots results in significantly better performance than a FIFO buffer with eight slots. Hence, beyond buffer space for two or three packets, it is more beneficial to allocate hardware resources to the more complex control of the DAMQ buffer than to use these resources for additional buffer memory.

As buffer size is increased, efficient utilization of buffer stor-

TABLE IV
THE PERFORMANCE OF A 64 × 64 OMEGA NETWORK COMPOSED OF 4 × 4 BLOCKING SWITCHES. RESULTS OBTAINED FROM SIMULATION. UNIFORM TRAFFIC. SHOWN ARE THE AVERAGE LATENCIES OF PACKETS THROUGH THE NETWORK FOR VARIOUS NETWORK THROUGHPUTS

Buffer Type	Buffer Size per Port	Average Latency Versus Network Throughput 0.10	0.20	0.30	0.40	0.50	saturated	Saturation Throughput
FIFO	1	3.67	5.51	Sat.	Sat.	Sat.	8.89	0.24
	2	3.14	3.39	3.88	5.41	Sat.	7.95	0.44
	3	3.15	3.38	3.81	4.82	Sat.	10.60	0.48
	4	3.14	3.38	3.79	4.65	9.34	13.14	0.51
	5	3.14	3.38	3.79	4.62	8.59	15.65	0.53
	6	3.15	3.34	3.79	4.63	7.78	17.87	0.55
	8	3.14	3.38	3.79	4.60	6.90	23.03	0.57
	12	3.15	3.38	3.79	4.61	6.78	33.00	0.59
SAMQ	4	3.24	3.58	4.09	4.90	6.57	6.68	0.50
	8	3.14	3.36	3.68	4.07	4.95	9.39	0.71
	12	3.15	3.36	3.68	4.16	4.91	13.00	0.78
SAFC	4	3.22	3.50	3.88	4.42	5.28	5.88	0.54
	8	3.13	3.29	3.51	3.80	4.21	7.53	0.75
	12	3.13	3.29	3.50	3.79	4.20	9.80	0.82
DAMQ	2	3.14	3.36	3.74	4.48	Sat.	7.19	0.50
	3	3.14	3.36	3.68	4.17	5.00	8.81	0.63
	4	3.14	3.36	3.68	4.16	4.91	10.66	0.71
	5	3.15	3.36	3.68	4.16	4.90	12.81	0.76
	6	3.14	3.36	3.68	4.16	4.90	14.85	0.80
	8	3.14	3.36	3.68	4.17	4.89	19.10	0.84
	12	3.14	3.36	3.68	4.16	4.92	29.15	0.90
CBDA	1	3.24	3.53	4.64	Sat.	Sat.	6.63	0.33
	2	3.13	3.30	3.50	3.81	4.35	6.31	0.59
	3	3.13	3.29	3.51	3.79	4.20	7.75	0.73
	4	3.13	3.29	3.50	3.80	4.19	9.71	0.80
	5	3.13	3.29	3.50	3.80	4.20	11.40	0.84
	6	3.13	3.29	3.51	3.79	4.20	13.84	0.86
	8	3.13	3.29	3.51	3.79	4.20	18.07	0.90
	12	3.13	3.29	3.51	3.79	4.21	26.07	0.94

age becomes less cirtical. Hence, the performance advantages of DAMQ buffers over SAMQ and SAFC buffers decreases with increasing buffer size. With sufficiently large buffers, the performance of SAFC buffers is expected to be *better* than the performance of DAMQ buffers since they allow multiple packets to be sent from a single input simultaneously. For example, for network throughput under 0.50, a 12 slot SAFC buffer achieves slightly *lower* latency than a 12 slot DAMQ buffer. However, it should be noted that the DAMQ buffer with only six packet slots achieves camparable performance. Based on the implementation described in [6], the additional control for DAMQ buffer will require significantly less chip area than the storage for a single 32-byte packet. Hence, there is a high cost, in terms of wasted chip area, for using SAFC buffers. Furthermore, as discussed in Section II, the SAFC buffer requires more complex flow control as well as prerouting, which are likely to involve even more circuitry and chip area.

For some applications, the handling of nonuniform traffic by the network may be a critical factor in determining system performance [16]. Of particular interest is "hot-spot" traffic, where a particular ("hot") destination receives a higher percentage of the packets than any other destination. Table V presents the results of simulating networks composed of the various switches with hot-spot traffic [16]. 5% of the packets from all senders are sent to the "hot" destination, while the destinations

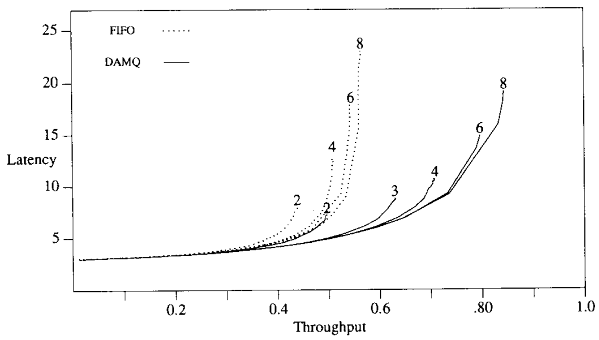


Fig. 4. The performance of a 64 × 64 Omega network composed of 4 × 4 FIFO or DAMQ blocking switches with several sizes of input buffers. Results obtained from simulation. Uniform traffic. The labels on the curves indicate the number of packet slots per buffer.

of the rest of the packets are uniformly distributed among all the receivers. The results demonstrate that with hot-spot traffic, the buffer type does not matter. Below saturation, the switches display almost equal latencies, just as with uniform traffic. However, unlike the situation with uniform traffic the switches all reach saturation at the same throughput of around 0.24.

With hot-spot traffic (Table V), the saturation throughput for all switches is significantly lower than with uniform traffic (Table IV). This is caused by the increased probability of contention within each switch for the output port on the path

TABLE V
THE PERFORMANCE OF A 64 × 64 OMEGA NETWORK COMPOSED OF 4 × 4 BLOCKING SWITCHES WITH BUFFER SPACE OF FOUR PACKET SLOTS PER INPUT PORT. 5% HOT-SPOT TRAFFIC. SHOWN ARE THE LATENCIES OF PACKETS THROUGH THE NETWORK FOR VARIOUS NETWORK THROUGHPUTS. RESULTS OBTAINED FROM SIMULATION

Buffer Type	Average Latency Versus Network Throughput					Saturation Throughput
	0.05	0.10	0.15	0.20	saturated	
FIFO	3.07	3.17	3.32	3.81	23.58	0.24
SAMQ	3.12	3.27	3.48	3.88	10.92	0.24
SAFC	3.11	3.25	3.43	3.78	10.53	0.24
DAMQ	3.07	3.16	3.30	3.67	25.20	0.24
CBDA	3.10	3.15	3.25	3.55	16.96	0.24

to the hot-spot. With FIFO buffers, when there is contention for an output port only one packet from one of the contending input ports is forwarded. All the other contending input ports are idle. Thus, within a short period of time, all of the switches which are on the path to the hot-spot have a high probability of having packets destined for the hot-spot at the head of their buffers and of having their buffers completely full. Pfister and Norton [16] call this effect "tree saturation." Since there is a path from every sender (processor) to each receiver (memory bank), when a hot-spot tree saturates, the traffic backs up to block every single sender. For all buffer types the network saturates at a throughput t that satisfies the equation $t(1 - h) + thp = 1$, where h is the fraction of packets destined for the hot-spot and p is the number of inputs/outputs of the network [16].

Tree saturation occurs with DAMQ switches just as with FIFO switches. The DAMQ switches forward all of the non-hot-spot traffic, but cannot forward hot-spot traffic since the bottleneck is the contention for output ports. This causes the buffers to fill up with hot-spot traffic. Once that happens, each DAMQ buffer is dominated by the queue to the output port on the path to the hot-spot and the network becomes saturated just like a network with FIFO switches. With the SAMQ and SAFC switches buffer space cannot become completely occupied by hot-spot traffic since it is statically partitioned. However, the blocked hot-spot traffic at the inputs to the network quickly block all non-hot-spot traffic attempting to enter the network, thus leading to saturation at the same levels as the other switches.

Since the buffer organization does not help in reducing the impact of tree saturation, other solutions must be found. One possiblility is to follow the design of the IBM RP3 multiprocessor [17] and use two separate networks: one for general traffic and the second, a combining network [8], for hot-spot traffic caused by synchronization traffic, such as accesses to semaphores. In a system such as this, the hot-spot traffic would not interfere with the uniform memory accesses, so significant performance gains would be made by using the DAMQ buffer instead of the FIFO buffer in the general traffic network. If the hot-spot traffic originates from only a subset of the nodes, an alternative scheme for limiting tree saturation is the use of feedback from the destinations to "throttle" the hot-spot traffic and the source [20]. In this case, the use of DAMQ switches instead of FIFO switches will still improve network performance for the general traffic.

V. SUMMARY AND CONCLUSIONS

The potential of large multiprocessors and multicomputers to achieve high performance can only be realized if they are provided with high-throughput low-latency communication. Fast small $n \times n$ switches with routing and buffering capabilities are critical components for achieving high-speed communication. The organization of the buffers in the $n \times n$ switches is one of the most important factors in determining their performance.

The architecture of $n \times n$ switches should allow them to efficiently utilize their buffer memory as well as the raw bandwidth of their ports. The architecture is constrained by the requirement that it must be amenable to high-performance cost-effective VLSI implementation. Since both the datapath and control of the switch must operate at high clock rates, complexity must be limited. Based on these considerations, we have developed a new type of buffer, called a *dynamically allocated multi-queue* buffer, for use in $n \times n$ switches. This buffer supports forwarding of packets in non-FIFO order and provides efficient handling of variable length packets. A critical advantage of DAMQ buffers is that flow control is simpler than with other multi-queue buffers and there is no need to "preroute" packets as with the other multi-queue buffers.

We have described the micro architecture of a DAMQ buffer and its controller in the context of the ComCoBB communication coprocessor for multicomputers. The DAMQ buffer can be efficiently implemented in VLSI to support packet transmission and reception at the rate of one byte per clock cycle with high clock rates. With a "hardwired" linked list manager and a fast routing mechanism, the ComCoBB chip will support virtual cut through of messages with a latency of four cycles.

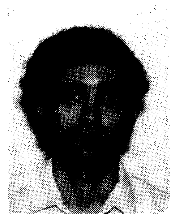
We have evaluated the DAMQ buffer by comparing its performance with that of three alternative practical buffers in the context of a synchronous store-and-forward multistage interconnection network. Both discarding and blocking switches were considered. The DAMQ buffer provides two key features: non-FIFO handling of packets and dynamic partitioning of buffer storage. We have shown that with hot-spot traffic that involves all the senders in the network, these features do not improve performance since they do not prevent tree saturation. For uniform traffic, our modeling and simulations show that these features result in large performance improvements over conventional FIFO buffers and/or static storage partitioning. The DAMQ buffer provides significantly lower latencies and higher maximum throughput than other practical buffer organizations with the same total buffer storage capacity.

Acknowledgment

Our work was aided and enhanced by other members of the ComCoBB group: H.-C. Chi, T. Frazier, and Y. Turner.

References

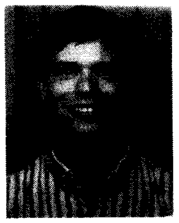
[1] D. Bertsekas and R. Gallager, *Data Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1987.
[2] W. Crowther, J. Goodhue, R. Gurwitz, R. Rettberg, and R. Thomas, "The butterfly parallel processor," *IEEE Comput. Architecture Newsletter,* pp. 18–45, Sept./Dec. 1985.
[3] W.J. Dally and C.L. Seitz, "The Torus routing chip," *Distributed Comput.,* vol. 1, no. 4, pp. 187–196, Oct. 1986.
[4] W.J. Dally, "Fine-grain message passing concurrent computers," in *Proc. Third Conf. Hypercube Concurrent Comput.,* vol. 1, Pasadena, CA, pp. 2–12, Jan. 1988.
[5] D.M. Dias and J.R. Jump, "Packet switching interconnection networks for modular systems," *IEEE Comput. Mag.,* vol. 14, no. 12, pp. 43–53, Dec. 1981.
[6] G.L. Frazier and Y. Tamir, "The design and implementation of a multi-queue buffer for VLSI communication switches," in *Proc. Int. Conf. Comput. Design,* Cambridge, MA, Oct. 1989, pp. 466–471.
[7] R.M. Fujimoto, "VLSI communication components for multicomputer networks," CS Division Rep. UCB/CSD 83/136, Univ. of California, Berkeley, CA 1983.
[8] A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer—Designing an MIMD shared memory parallel computer," *IEEE Trans. Comput.,* vol. C-32, pp. 175–189, Feb. 1983.
[9] Intel Corporation, *82596 User's Manual,* 1989.
[10] M.I. Irland, "Buffer management in a packet switch," *IEEE Trans. Commun.,* vol. COM-26, pp. 328–337, Mar. 1978.
[11] M.J. Karol, M.G. Hluchyj, and S.P. Morgan, "Input vs. output queueing on a space-division packet switch," in *Proc. IEEE Global Telecommun. Conf.,* Houston, TX, Dec. 1986, pp. 659–665.
[12] P. Kermani and L. Kleinrock, "Virtual cut through: A new computer communication switching technique," *Comput. Networks,* vol. 3, no. 4, pp. 267–286, Sept. 1979.
[13] M. Kumar and J.R. Jump, "Performance enhancement in buffered Delta networks using crossbar switches and multiple links," *J. Parallel Distributed Comput.,* vol. 1, no. 1, pp. 81–103, 1984.
[14] D.H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.,* vol. C-24, pp. 1145–1155, Dec. 1975.
[15] R.J. McMillen and J. Siegel, "The hybrid cube network," in *Proc. Distributed Data Acquisition, Comput, Contr. Symp.,* Dec. 1980, pp. 11–22.
[16] G.F. Pfister and V.A. Norton, "Hot spot contention and combining in multistage interconnection networks," *IEEE Trans. Comput.,* vol. C-34, pp. 943–948, Oct. 1985.
[17] G.F. Pfister, W.C. Brantley, D.A. George, S.L. Harvey, W.J. Kleinfelder, K.P. McAuliffe, E.A. Melton, V.A. Norton, and J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): Introduction and architecture," in *Proc. 1985 Int. Conf. Parallel Processing,* Aug. 1985, pp. 764–771.
[18] D.A. Reed and R.M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing.* Cambridge, MA: MIT Press, 1987.
[19] Y. Rimoni, I. Zisman, R. Ginosar, and U. Weiser, "Communication element for the versatile multicomputer," in *Proc. 15th IEEE Conf. in Israel,* Apr. 1987.
[20] S.L. Scott and G.S. Sohi, "The use of feedback in multiprocessors and its application to tree saturation control," *IEEE Trans. Parallel Distributed Syst.,* vol. 1, pp. 385–398, Oct. 1990.
[21] C.L. Seitz, "The Cosmic Cube," *Commun. ACM,* vol. 28, no. 1, pp. 22–33, Jan. 1985.
[22] K.S. Stevens, S.V. Robinson, and A.L. Davis, "The post office—Communication support for distributed ensemble architectures," in *Proc. 6th Conf. Distributed Comput. Syst.,* Cambridge, MA, May 1986, pp. 160–166.
[23] Y. Tamir and G.L. Frazier, "High-performance multi-queue buffers for VLSI communication switches," in *Proc. 15th Annu. Int. Symp. Comput. Architecture,* Honolulu, HI, May 1988, pp. 343–354.
[24] Y. Tamir and J.C. Cho, "Design and implementation of high-speed asynchronous communication ports for VLSI multicomputer nodes," in *Proc. Int. Symp. Circuits and Syst.,* Espoo, Finland, June 1988, pp. 805–809.
[25] Y. Tamir and Y.F. Turner, "High-performance adaptive routing in multicomputers using dynamic virtual circuits," in *Proc. 6th Distributed Memory Comput. Conf.,* Portland, OR Apr. 1991, pp. 404–411.
[26] C. Whitby-Strevens, "The Transputer," in *Proc. 12th Annu. Symp. Comput. Architecture,* Boston, MA, June 1985, pp. 292–300.
[27] H. Yoon, K.Y. Lee, and M.T. Liu, "Performance analysis of multi-buffered packet-switching networks in multiprocessor systems," *IEEE Trans. Comput.,* vol. 39, pp. 319–327, Mar. 1990.



Yuval Tamir (S'78–M'85) received the B.S.E.E. degree from the University of Iowa, Iowa City, in 1979, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 1981 and 1985, respectively.

Since 1985 he has been on the faculty of the Computer Science Department at the University of California, Los Angeles, where he is currently an Assistant Professor. He established and is currently directing the Computer Science VLSI Systems Laboratory. His research interests include scalable parallel architectures, fault-tolerant computing, and VLSI systems.

Dr. Tamir is a member of the IEEE Computer Society and the Association for Computing Machinery.



Gregory L. Frazier (S'87) received the S.B. degree in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, MA, in 1986.

He is currently a Ph.D. student in the Computer Science Department at the University of California at Los Angeles. His research is in the area of architectural support for inter-processor communication in multicomputers and multiprocessors. He received a DARPA/NASA Research Assistanceship in Parallel Processing in 1991, a Chancellor's Fellowship at UCLA in 1986.

Mr. Frazier is a student member of the Association for Computing Machinery and the IEEE Computer Society.