UNIVERSITY OF CALIFORNIA

Los Angeles

Buffering and Flow Control in Communication Switches for Scalable Multicomputers

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Computer Science

by

Gregory Lee Frazier

1995

The dissertation of Gregory Lee Frazier is approved.

Kirby A. Baker

Henry Samueli

Milos D. Ercegovac

David Rennels

Yuval Tamir, Committee Chair

University of California, Los Angeles

1995

This dissertation is dedicated to Verra Morgan, the oil that
keeps the wheels of the U.C.L.A. Computer Science Department turning,
to my daughters, Mikaela and Joanna, whose love has redefined my world,
and my wife, Tiffany, whose excellence sets the mark I try to match.

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

I must acknowledge the large number of people who helped make this dissertation a reality. Primary among them is my advisor, Professor Yuval Tamir, whose instruction and guidance were invaluable. Also important to my work was the advise and support of my cohorts in the UCLA VLSI Systems Laboratory: Marc Tremblay, Yoshio Turner, G Janakiraman, Sanjay Jain, Tiffany Frazier and Hsin-Chou Chi. Whether the problem was coding, finding a reference, formatting a document or simply moral support, these were the people I turned to. Not only my own research, but the activities of the UCLA Computer Science Department as a whole would have ground to a halt without the efforts of Verra, Roberta, Jackie, Judy and the rest of the staff. Finally, I would like to acknowledge my family, whose love powered me through the long years of graduate school: my wife Tiffany, my daughters Mikaela and Joanna, my mother Sue Frazier and my sister Melissa (who has graciously waited for me to finish my dissertation before completing hers).

VITA

February 15, 1964	Born in Ann Arbor, MI
1986	B.S. Computer Science and Engineering Massachusetts Institute of Technology Cambridge, MA
1986-1987	Chancellor's Fellow University of California Los Angeles, CA
1987-1990	Research Assistant, Research Associate University of California Los Angeles, CA
1991	M.S. Computer Science University of California Los Angeles, CA
1991-1993	NASA/DARPA Research Assistantship in Parallel Processing
1993-1994	Teaching Associate University of California Los Angeles, CA
1994-	Senior Software Engineer SAIC McLean, Virginia

PUBLICATIONS AND PRESENTATIONS

1. G. L. Frazier and Y. Tamir, ‘‘The Design and Implementation of a Multi-Queue Buffer for VLSI Communication Switches,’’ *International Conference on Computer Design*, Cambridge, MA, pp. 466-471 (October, 1989).

2. Y. Tamir and G. L. Frazier, ‘‘Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches,’’ *IEEE Transactions on Computers* **41**(6), pp. 725-737 (June 1992).

3. Y. Tamir and G. L. Frazier, ‘‘Hardware Support for High-Priority Traffic in VLSI Communication Switches,’’ *Journal of Parallel and Distributed Computing* **14**(4), pp. 402-416 (April 1992).

4. Y. Tamir and G. L. Frazier, ‘‘Support for High-Priority Traffic in VLSI Communication Switches,’’ *9th Real-Time Systems Symposium*, Huntsville, AL, pp. 191-200 (December 1988).

5. Y. Tamir and G. L. Frazier, ‘‘High-Performance Multi-Queue Buffers for VLSI Communication Switches,’’ *15th Annual International Symposium on Computer Architecture*, Honolulu, Hawaii, pp. 343-354 (May 1988).

ABSTRACT OF THE DISSERTATION

Buffering and Flow Control in Communication Switches

for Scalable Multicomputers

by

Gregory Lee Frazier

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1995

Professor Yuval Tamir, Chair

Small $n \times n$ switches are key components of the communication networks of multicomputers. For fine-grained distributed tasks to efficiently execute on large multicomputers, high throughput, low latency communication must be supported. The architecture of these switches, particularly their internal buffers, is critical for achieving high performance communication with cost-effective implementations.

The focus of the work in this dissertation is the design of a new buffer, the *dynamically allocated, multi-queue* (DAMQ) buffer. This buffer maintains multiple queues of packets and allows the queue heads to be accessed in any order rather than restricting access to first in, first out (FIFO) order for the entire buffer. This reduces the effect of output port contention, giving a network of switches with DAMQ buffers (DAMQ switches) a significantly higher bandwidth and a lower average latency at given throughputs than a network of FIFO switches.

The DAMQ buffer is easily enhanced to support low latency delivery of high-priority packets even under heavy network load. This is important for distributed

real-time systems where a subset of the packet transmissions must happen within a bounded time.

The implementation complexity and performance characteristics of several non-FIFO buffer structures are evaluated. Extensive simulation and analysis are used in the evaluation of the DAMQ buffer and alternative buffers. A VLSI implementation of the DAMQ buffer as a component for communication switches is presented. Detailed circuit simulations are used to demonstrate the clear superiority of the DAMQ buffer over conventional FIFO buffers.

A methodology for evaluating flow control mechanisms is presented. A *congestion benchmark suite* is used to ‘‘stress’’ selected flow control mechanisms with a variety of congestion patterns. It is shown that hop-level flow control schemes in conjunction with multi-queue buffers can be used to optimize network performance under uniform and non-uniform traffic conditions.

Chapter One
Introduction

A computer program can be viewed as a set of instructions or operations which must be executed in order to complete a task. A primary goal of computer architecture is to design computers which will execute programs in the shortest time possible. Two possibilities for increasing the speed of a computer are to increase the speed with which each instruction is executed or to increase the number of instructions which can be executed simultaneously. Recent trends in VLSI technology have shifted the focus of high performance computing from the former possibility to the latter. A large scale distributed computer, consisting of hundreds or thousands of independent processing nodes, has the potential to bring many times the processing power of a supercomputer to bear upon the execution of a program for a fraction of the supercomputer's cost.

A distributed program is a program implemented as set of processes, where each process can be executed in parallel on the separate nodes of a distributed computer. For these processes to cooperate in the solution of a problem they must communicate with each other, either by transmitting data from one processor to another via messages or by accessing shared memory. Distributed systems which use the former communication model are often referred to as *multicomputers*, while shared memory machines are *multiprocessors.* The more processes a task can be divided into, the more steps toward the task's completion that can potentially be executed simultaneously, and, in theory, the faster the task will complete. However, as the process granularity becomes very fine, the latency of the process-

to-process or process-to-memory (depending upon the architecture) communication begins to contribute significantly to the task's execution time. In addition, the available bandwidth of the communication network may be less than the desired throughput of communication. These two issues — high communication latencies and finite network bandwidth — limit the degree to which a task can be profitably distributed. Implementing high bandwidth, low latency communication networks is thus critical to the ability of multiprocessors and multicomputers to achieve high performance by exploiting parallelism.

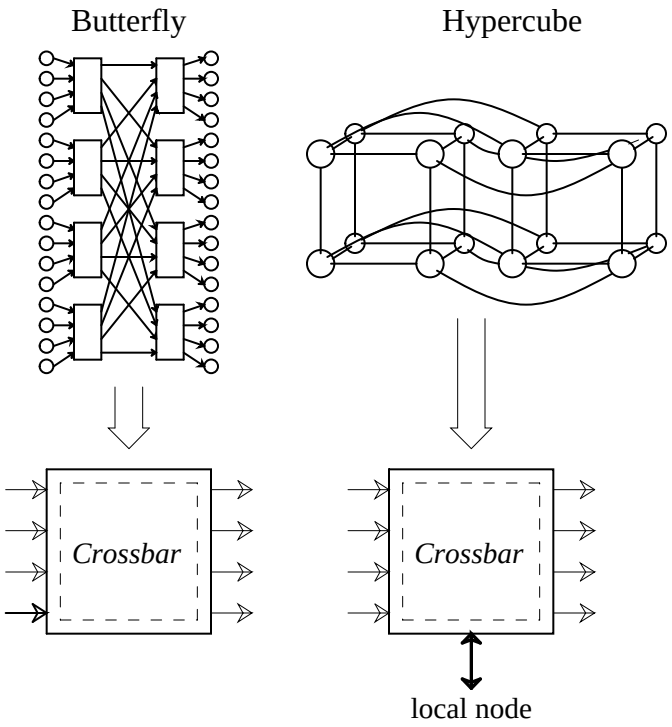


Figure 1.1: *Switches in Multiprocessors and Multicomputers.* Small $n \times n$ switches are the central component of the communication networks of both multiprocessors and multicomputers.

Multiprocessors with a large number of nodes (e.g. greater than 64) use multistage interconnection networks to connect processors to memory [Crow85,

Gott83, Rett90, Pfis85b]. These networks are typically composed of a large number of small $n \times n$ switches (typically, $2 \leq n \leq 10$) (Fig. 1.1). Memory references are formed into packets to be transmitted across these networks. Multicomputers, on the other hand, use point-to-point links dedicated to nearest neighbor communication [Seit85, Whit85, Dall87b]. If a process wishes to send a message to a node which is not an immediate neighbor, the packet(s) which constitute the message must pass through intermediate nodes to reach their destination. To accelerate its performance, each computing node relies on a communication coprocessor with a small number of ports [Dall86, Stev86] that function as a small $n \times n$ switches with $n-1$ ports connected to other nodes, and one bidirectional port connected to the local application processor (Fig. 1.1). The design of high performance small $n \times n$ switches is thus of critical importance to the development of multiprocessor and multicomputer systems. Since many of these $n \times n$ switches are needed in a large system, there is strong motivation to implement each switch as a single VLSI chip.

1.1. Latency and Output Port Contention

A switch's job 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 a 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 must be stored at the node for later transmission. The maximum throughput at which the switch can operate depends directly on how efficiently the switch can store the conflicting packets and forward them when the appropriate output port is no longer busy.

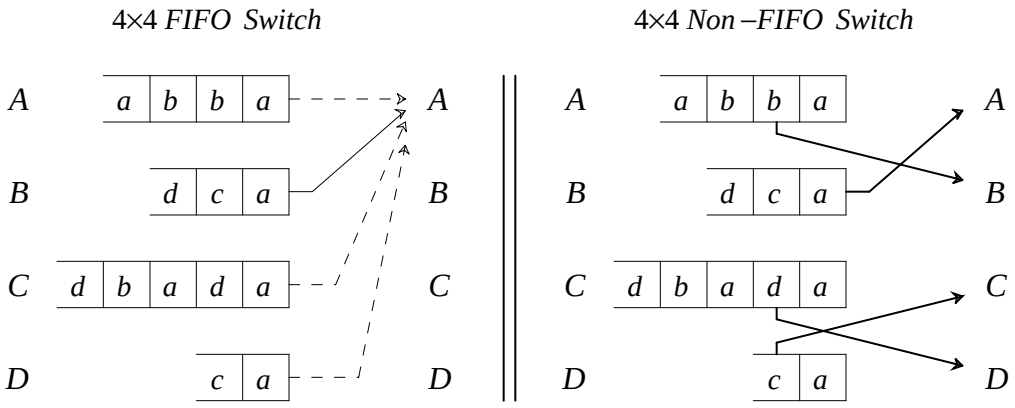


Figure 1.2: *Output port contention.* In the FIFO switch, packets destined for output port A block packets destined for the other output ports. In the other switch, non-FIFO buffer access improves buffer and output port utilization.

Communication switches which use first in, first out (FIFO) buffers located at the input ports to store packets [Rimo87] do so because it is a simple scheme to implement. Space allocation in a FIFO buffer is simple, even for variable size packets, and only two registers are required to maintain buffer organization (pointers to the front and rear of the queue, with a ‘‘circular’’ memory array). This simplicity, however, has a price — FIFO buffers are very inefficient. Their inefficiency stems from the fact that, in a FIFO buffer, only the ‘‘oldest’’ packet in the buffer is available for transmission at any given point in time. This makes the switch susceptible to *output port contention.* Output port contention is the situation in which two or more buffers have packets destined for the same output port

(Figure 1.2). Since each output port can only transmit one packet at a time, one or more of the buffers cannot transmit its packet, and because in a FIFO buffer only the first packet in the queue is available for transmission, those buffers must remain idle while the other transmits its packet. In addition, the buffers which are idle may have within them packets destined for other output ports, which may also be idle. In this situation, the FIFO property of the buffers is limiting the throughput of the switch; available bandwidth could potentially be better utilized by using non-FIFO buffers.

The significance of this is magnified when dealing with real-time systems. Multiprocessor and multicomputer systems that are connected to input/output devices which interact with the outside world occasionally require particularly fast communication with different parts of the system. The broadcasting of certain types of system-wide events, such as the initiation of global system rollback, may also require preferential handling by the network. Real-time requirements due to interaction with I/O devices and system-wide exception handling thus lead to the need to support special purpose *high-priority* traffic whose maximum latency is significantly lower than the maximum latency for general traffic. In a switch composed of FIFO buffers it is likely that, upon entering a switch, a high-priority packet will be trapped behind a ‘‘normal’’ packet, and will thus be delayed. A switch composed of non-FIFO buffers, on the other hand, would be able to transmit the high-priority packet as soon as its destination output port is free, allowing the high-priority packet to bypass the normal packets.

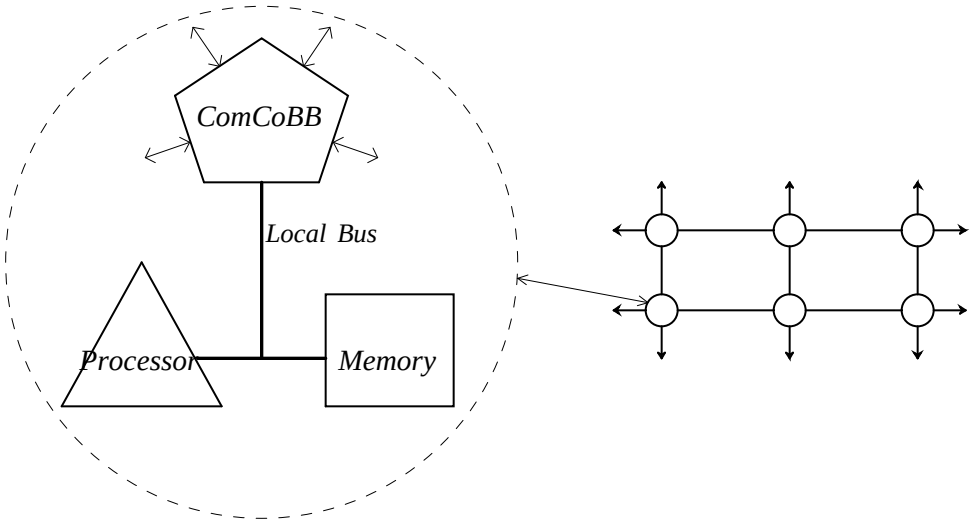


Figure 1.3: *The ComCoBB Chip.* The ComCoBB Chip is a communication coprocessor whose intent is to ‘‘hide’’ the communication network from the processing nodes of a multicomputer.

1.2. The Dynamically Allocated, Multi-Queue Buffer

The results reported in this dissertation were produced as part of the UCLA ComCoBB project. The goal of the ComCoBB (**Com**munication **Co**processor **B**uilding-**B**lock) project is to design and implement a single-chip high-performance communication coprocessor for use in VLSI multicomputer systems. In a multicomputer node consisting of a CPU, local memory and various co-processors, the ComCoBB chip handles forwarding all packets which do not originate from or are not destined to the node, as well as handling the network interface for all packets / messages which are to or from the node (Figure 1.3). Thus, the ComCoBB chip is, in part, a small $n \times n$ switch. It quickly became evident that one of the keys to designing a high performance packet switch was to design an efficient buffering scheme for the switch. The focus of this dissertation is the

design and implementation of the *dynamically allocated, multi-queue* (DAMQ) buffer, the internal buffer of a small $n \times n$ VLSI communication switch.

The key to the high performance of the DAMQ buffer is its non-FIFO behavior. The DAMQ buffer maintains an internal organization of packets via linked lists. The lists correspond to the destination ports of the packets and the priority with which the packets are forwarded. This allows the buffer to be accessed ‘‘randomly’’, based upon the destination and priority of the packets within it. The use of linked lists, however, maintains the FIFO ordering of the packets of the same priority between each input port / output port pair, thus supporting routing schemes such as virtual circuits which depend upon packet ordering.

The DAMQ buffer supports features other than non-FIFO switching which contribute to its high performance. With *shift register addressing,* [Tami88a] the DAMQ buffer can transfer data into and out of a large buffer memory at very high bandwidths. This addressing scheme also efficiently supports implementation of a dual ported buffer memory, which is in turn needed to support *virtual cut-through* [Kerm79]. Virtual cut-through is the ability to begin to forward a packet as soon as it has been routed, i.e. before it has been completely received. This has the potential to significantly reduce the latency of packets traveling across a lightly loaded network. Finally, the DAMQ buffer can support variable size packets; the design discussed in this thesis supports packets of one to thirty-two bytes in length. Supporting variable length packets reduces the average packet latency, increases the system throughput and is essential in a high-performance system running fine-grained distributed tasks.

1.3. The Scope of this Work

In this dissertation we present the architecture of new buffer, the dynamically allocated multi-queue buffer. We demonstrate how the effect of output port contention on switch throughput is reduced by organizing the packets within a buffer into multiple queues and associating the queues with each of the switch's output ports, thus allowing the packets to be accessed based upon where they are routed. This scheme is extended by associating the queues not only with output ports but also with packet priority levels. This extension allows the DAMQ buffer to support distributed real-time computing by providing ''special'' service to high-priority packets.

A comprehensive analysis of the DAMQ buffer is provided by (a) exploring the implementation considerations of the DAMQ buffer versus both FIFO buffers and alternate non-FIFO buffer architectures and (b) by extensively analyzing and simulating switches implemented with these buffers. The simulations are in the context of a multistage interconnection network, and demonstrate the efficacy of the DAMQ buffer for both a general purpose multicomputer, where the average latency per throughput and the maximum network throughput are the key performance measurements, and for a distributed real-time system, where the maximum latency of high-priority (real-time) packets for a given network throughput is the key measurement.

An implementation of a DAMQ buffer is presented, and it is shown that the complex control required for a DAMQ buffer does not reduce its raw bandwidth relative to a simpler FIFO buffer. Further, the implementation reveals that the DAMQ buffer control logic does not require significantly more silicon area to

implement than the FIFO buffer control logic.

Finally, this dissertation explores communication flow control mechanisms for scalable multicomputers. We argue the necessity of implementing hop-level flow control for large scale multicomputers. Several hop-level flow control mechanisms are evaluated, both on the basis of their cost of implementation and on the basis of the degree to which they impact communication network performance under a variety of data traffic conditions. We found that no single hop-level flow control mechanism that we explored out-performed all of the other mechanisms in every traffic pattern that we evaluated. However, it is demonstrated that, due to the fact that it dynamically allocates its buffer space on a per-packet basis and maintains its internal organization via linked lists, the DAMQ buffer can support a wide variety of flow control mechanisms.

In the next chapter, we review previously published work in the areas of communication buffer design, switch design, flow control and support for real-time multicomputer / multiprocessor communication. In Ch. 3, the architecture of the DAMQ buffer is presented. The body of the chapter examines switch and buffer architecture tradeoffs, evaluating them on the basis of their ability to promote high performance communication. The tradeoff evaluations result in the specification of several non-FIFO buffer architectures, including the DAMQ buffer. Finally, details of the DAMQ buffer architecture such as the implementation of multiple queues of packets within the buffer as linked lists and support for variable length packets are presented.

In Ch. 4, the DAMQ switch is evaluated in the context of a multi-stage interconnection network by comparing it to alternative switch architectures,

including a switch composed of FIFO buffers. The comparisons are made with the assumption that the ‘‘raw bandwidth’’ (the instantaneous bandwidth for each input and output port) is limited by the rate at which the link can be driven, as opposed to the rate at which a buffer can be accessed, and thus the raw bandwidth is equal for switches using the different buffer architectures. The comparisons were performed using Markov analysis and event-driven simulations of a synchronous communication network. These evaluations demonstrate the performance gains achievable by using the DAMQ buffer.

Chapter 5 examines the idea of utilizing the non-FIFO property of the DAMQ buffer to support multiple classes of packets. By dedicating a packet queue to ‘‘high-priority’’ packets, a network can guarantee a low maximum latency for critical network traffic. The simulator presented in Ch. 4 is used to evaluate the high-priority queue.

Chapter 6 evaluates the performance of the DAMQ buffer in the context of an asynchronous communication network. Since the DAMQ buffer design discussed in this dissertation has features which cannot be used in a synchronous protocol — support for variable-length packets and virtual cut-through — it is important to evaluate the performance in an asynchronous environment. The omega network simulation results in this chapter support the results and conclusions of Ch. 4. Also in this chapter, it is shown that the DAMQ buffer can be used to implement deadlock-free dimensional routing for the torus network topology, similar to that presented in [Dall86]. An enhancement to this routing mechanism is proposed and evaluated via simulation of a two-dimensional square torus network.

A VLSI implementation of the DAMQ buffer is described in Ch. 7. The

DAMQ buffer is implemented as a functional unit independent of any single switch architecture. This single-chip DAMQ buffer (the *DAMQ Chip*) is designed for use as the communication interface for a multicomputer node. A comparison of the silicon area required to implement the DAMQ buffer versus a comparable FIFO buffer is presented, and circuit analysis demonstrates that the complex control logic of the DAMQ buffer does not limit the clock frequency or reduce the buffer's raw bandwidth.

The evaluations of communication network performance presented in Chs. 4-6 focus on a buffer's ability to forward packets as the key to network performance, keeping other network features constant. Yet, there are other communication network features which not only affect network performance, but also interact with the buffers in such a way as to enhance the performance of some buffer architectures over others. Of particular interest is the flow control mechanism employed by the network. Ch. 8 discusses previous work in the area of flow control for packet switching communication networks. It establishes hop-level flow control as an integral component of a communication network for a scalable distributed computer, and presents some alternative hop-level flow control mechanisms. The relationship between the flow control mechanisms and the buffer architecture is discussed, and their impact upon communication network performance is measured via simulation. Then the complexity / cost of implementing these mechanisms is examined, and the chapter ends with a summary of hop-level flow control.

Chapter 9 summarizes the dissertation. It reprises the contributions claimed by this dissertation and reviews the analyses and conclusions of each chapter. It is

argued that the DAMQ buffer is an appropriate choice when implementing a communication switch intended for large scale multicomputer and multiprocessor packet switching networks. The chapter concludes by presenting potential topics of future research.

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 λ which was the sum of the Poisson arrival rates λ_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 μ 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, λ_2 was held at 0.5, λ_3 was held at 0.7, and λ_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 ≈ 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.

Chapter Three
The DAMQ Buffer Architecture

When designing a switch, the goal is to enhance the performance of the communication network. A network's performance is measured by its saturation throughput and the average latency of packets traversing the network at a given throughput. Translating the network performance goals into the design of an $n \times n$ communication switch is not trivial. There are a number of fundamental tradeoffs to evaluate, including how many ports the switch should have, whether or not to buffer packets in their entirety at intermediate nodes, and where to buffer the packets if one is doing so.

This chapter describes the process by which we arrived at the DAMQ buffer architecture. The following sections evaluate the features a communication switch should have to support high performance packet switching. The first sections focus on the basic structure of a switch. This includes the location, size, and fundamental organization of the packet buffers. Then, having arrived at a basic architecture, the additional characteristics that a switch should have to support high-performance communication are enumerated. The result is a specification of the DAMQ buffer architecture, as well as the architectures of feasible alternative buffers — succeeding chapters compare the performance of networks implemented with DAMQ buffered switches to the performance of networks implemented with these alternate buffer architectures.

3.1. Switch Size

An issue fundamental to the architecture of a switch (and to the architecture of the communication network that the switch will implement) is the number of input and output ports the switch will have. Different internal switch organizations may be preferable depending on the number of ports. For example, as discussed below, packet buffering may be done in a central buffer pool. In order for the switch to be able to utilize all of its ports simultaneously, the bandwidth of a central buffer pool must be greater than or equal to $2n$ times the bandwidth of a port, where n is the number of input / output port pairs on the switch. While this may be feasible on a 2×2 switch, and may even result in an efficient implementation, it is far less reasonable on a 6×6 switch. Further, the complexity of arbitrating between up to $2n$ simultaneous requests to the buffer grows significantly as n increases. Finally, the relative performance of various buffer architectures changes with the number of ports. For example, the bandwidth of a switch with infinite FIFO buffers at its input ports drops from 0.75 to 0.65524 as the number of ports increases from two to four.

As was discussed in Ch. 2, many researchers have independently concluded that switches with a small number of input and output ports ($3 \le n \le 6$) result in better network performance than do switches with a large number of ports. Kumar and Jump, comparing the performance of multistage interconnection networks in [Kuma84], showed that networks of 4×4 switches outperform networks of 2×2 switches when the total network buffer space is restricted.

3.2. Buffer Location

Packet buffers can be located at the input ports (i.e. the buffer space can be statically partitioned among the input ports), it can be centrally located (complete sharing), the buffers can be placed at the output ports of the switch, or a combination of the above may be used.

Complete sharing of available storage by all communication ports results in more efficient storage utilization than a static partitioning of the buffer storage between the ports. This suggests that the optimal switch organization is a central buffer pool, where all free memory is available for allocation to any packet arriving at any input port. In fact, in Ch. 4, switches with central buffer pools are used as ‘‘pseudo-ideal’’ switches (these are referred to as CBDA switches — centrally-buffered, dynamically allocated). However, there are fundamental difficulties with producing efficient high-performance implementations of a switch with shared central buffers.

A major difficult is that, in order to achieve high performance, multiple high-bandwidth communication ports must be able to access packet 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 of the input and output ports. There are three ways to implement such a buffer. First, it can have a single read and a single write port whose widths are equal to the width of the input and output ports, but which operate n times faster than the communication links. Given the high rates which are achievable for inter-chip communication [Rett90], this is not a realistic option. A second possibility is to keep the buffer ports’ width the same as that of the communication ports, but to

implement multiple read and write ports to the buffer. For an $n \times n$ switch, the buffer must have $2 \times n$ ports in order to support n simultaneous reads with n simultaneous writes. Multiport memory is undesirable in a VLSI implementation because its implementation is expensive (in silicon area) and leads to poor performance (long access times). The third method to support $2 \times n$ simultaneous accesses is to implement one write port and one read port, each n times wider than the communication ports, and multiplex these among the input and output ports [Stev86, Stun94]. Since the output ports must wait their turn to access the central buffer pool, the use of a central buffer with a single, wide, multiplexed read port can result in lost bandwidth and increased transmission latencies — particularly for switches with many ports and when transmitting variable-length packets (or packets whose length in flits is not a multiple of the number of ports).

To ameliorate the problem of access to a central buffer pool, switches which utilize central buffer pools implement them in combination with buffers at the input ports [Stev86, Davi92, Stun94]. Incoming packets are initially stored in the buffer located at the input port while the switch attempts to cut the packet through. The idea is to utilize virtual cut-through to avoid incurring the latency associated with storing packets to and retrieving packets from the central buffer pool. If cut-through is not possible, the input port buffer acts as a staging buffer, concatenating the incoming data into wide words to store into the multiplexed central buffer pool. Unfortunately, virtual cut-through is achieved at low network throughputs — conditions under which the buffers of a switch are not heavily utilized. Under conditions of high throughput where efficient buffer utilization is desired, virtual cut-through will not be possible, and packets passing through the switch will most

likely be stored into and retrieved from the central buffer pool.

In addition to implementation difficulties and switch-local performance problems, shared central buffers can also cause global performance problems. Previous studies have shown that, with complete sharing, a single congested output port may ‘‘hog’’ the available storage in a centralized buffer pool, impeding all other communication through the switch [Irla78, Fuji83, Reed87]. This, in turn, can cause neighboring switches, which cannot transmit packets to the full switch, to have their buffer fill up, converting a single ‘‘hogged’’ buffer into a system-wide problem.

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 [Karo86], due to the elimination of head-of-queue blocking. This implies that a switch with input buffering requires larger buffers in order to achieve the same probability of buffer overflow. 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. Further, as was the case with central buffers, there is the need for staging buffers at the input ports; incoming packets must be stored some place while they are being routed, and the incoming flits must be concatenated into wide words for storage in the output port buffers.

The remaining option is to implement buffering at the input ports. A primary

advantage of input port buffering is that at most one packet will be being written to it at any given point in time. Furthermore, if the buffer is managed as a FIFO queue, it is very easy to deal with variable length packets. For these reasons most current multicomputer communication switches include buffers associated with the input ports [Dall86, Rimo87, Stev86, Davi92, Oed, Stun94, Leis92, Dall87b, Lill91].

3.3. Buffer Size and Structure

The amount of buffer space to include on a communication switch depends upon a number of factors, not the least of which is the switching paradigm being used — specifically, whether one is supporting *packet switching* vs. *wormhole switching* [Dall86].

The Torus Routing Chip [Dall86] takes an extreme position with regards to the amount of buffer space to include on a switch — each input port has two four-byte buffers associated with it, which is not enough storage for a packet. When a packet is blocked (cannot be cut through a switch), its data is stored in the buffers of multiple switches, and occupies those buffers (and communication channels) until the packet can be forwarded. The Cray T3D takes a more moderate approach. While [Oed] does not specify the amount of buffer space available on each switch, it does say that the buffers can hold small packets while larger packets must be wormhole routed. It has been shown that providing buffering for approximately four packets at each input port dramatically improves network throughput [Tami92a, Kuma84, Ahma89]. This dissertation does not evaluate wormhole routing, but rather evaluates buffers which can store a small number of

packets in their entirety.

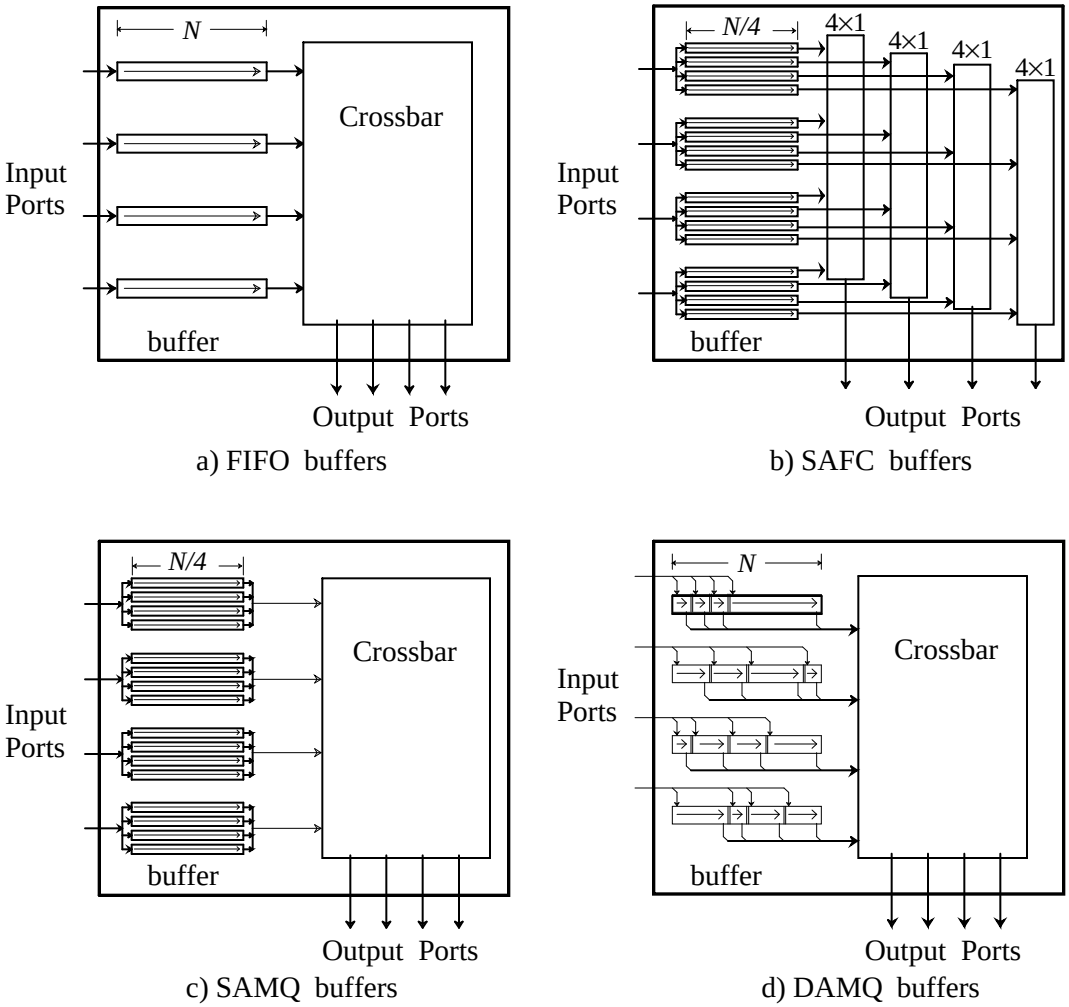


Figure 3.1: Alternative designs of switches with input port buffers.

A buffer which can store multiple packets requires a structure to order these packets. A FIFO buffer with a single write port and a single read port offers the most simple structure, by storing consecutive flits in consecutive buffer memory locations. A communication switch with four input and four output ports using FIFO input buffers is shown in Fig. 3.1a. 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 of the buffer types we examined.

As mentioned in Ch. 1, 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 [Kuma84, McMi80]. In the case of a four-port switch, this amounts to sixteen separate buffers. We refer to this type of buffer as being *statically allocated,* as the buffer space associated with each input port is statically allocated to the output ports. We explored two statically allocated buffer architectures: *statically allocated, multi-queue* (SAMQ) and *statically allocated, fully-connected* (SAFC). The difference between these architectures is that the SAMQ buffer has a single read port shared by all of the queues at an input port, where as the SAFC buffer has separate read ports for each queue. Since multiple packets can be read at the same time from an SAFC buffer, a 4×4 crossbar will not accommodate all of the possible ways in which packets can be transmitted from an SAFC switch. Instead, this scheme requires a 16×4 crossbar or, as we have shown in Fig. 3.1b, four 4×1 crossbars.

There are several problems with static buffer allocation. First, SAMQ and SAFC buffers do not utilize their storage as efficiently as does the FIFO buffer. Since the available buffer space at each input port is *statically* partitioned, in a 4×4 switch, only one quarter of the input buffer space is available to a given packet. Thus, with static buffer allocation, packets may be rejected by an input port due to

a lack of buffer space, even if there is storage at that input port. This is in contrast to the FIFO buffer, where the entire storage at the input port is available for any arriving packet.

Another problem with statically allocated buffers is the difficulty of efficiently implementing *blocking flow control*. In blocking switches [Gott83], the output port of a switch is allowed to transmit only if there is space in the buffer of the destination switch to store the packet. When the buffer of a switch fills up, the output port of the neighboring switch must be blocked from transmitting to avoid data loss due to buffer overflow. In this case, the switch whose buffer is full notifies the neighboring switch that it cannot transmit any more messages until some buffer space is freed up. A 4×4 switch with four queues implemented as separate buffers at each input port must convey information about each of these queues to the corresponding neighbor. This is four times the amount of information that is necessary for a 4×4 switch implemented with FIFO buffers.

More importantly, if a neighboring switch is notified that there is a full queue in the next switch, it must *pre-route* packets to determine which queue of the destination buffer packets are to be stored in before transmitting them. Performing 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 pre-routing is possible, it increases the complexity of the routing hardware and makes dynamic routing problematic. 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 pre-routing but requires that buffers not delete packets as they transmit them, but

instead wait until an acknowledgement is received before reusing the buffer memory. The acknowledgement 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. Ch. 8 presents a detailed discussion of flow control mechanisms.

It should be noted that pre-routing 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. If incoming packets do not have the identity of the queue to which they are to be appended already associated with them as they arrive at the input port, they have to be stored in a staging buffer while the routing decision is made.

What is desired is a buffer which can access the packets destined for each of the output ports separately (i.e. a non-FIFO buffer), but which can apply its free space to any incoming packet, thus avoiding the poor buffer utilization and the pre-routing associated with static buffer allocation. This is the buffer which we have designed, and which we call the *dynamically allocated, multi-queue* (DAMQ) buffer. Dynamically allocated, because the space within the buffer is not statically partitioned among the output ports, but is allocated on the basis of each packet received. Multi-queue, because within each buffer there are separate FIFO queues holding packets destined for each output port (Fig. 3.1d). This buffer does not require pre-routing, because whether or not a packet can be received does not depend upon the direction of its future hops.

3.4. Buffer Features

In the previous sections we determined that a buffer should reside at the input ports of a switch, maintain FIFO queues of packets for each of the output ports, and dynamically allocate its buffer space on a per-packet basis. This provides a basic structure for the DAMQ buffer but does not cover some important issues in designing a communication buffer for a multicomputer.

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 (i.e. full utilization of all ports). This implies that the buffer must be dual ported to allow a read and a write to occur simultaneously. Also, a switch should only buffer a packet as long as necessary, i.e. it should not wait for the entire packet to arrive before transmitting it — virtual cut-through [Kerm79] must be supported. The DAMQ buffer has a dual-ported buffer memory and logically separate reception and forwarding control circuitry so that it can simultaneously receive and transmit packets (or perform virtual cut-through).

Since the DAMQ buffer allows multiple queues to share a single physical memory, it requires a mechanism to maintain packet order. The DAMQ Buffer Chip uses linked lists to implement the packet queues — each packet in the buffer is pointed to by the previous packet in its queue and points to the next. In our implementation of the DAMQ buffer (Ch. 7), the links lists are implemented by dividing the buffer memory into fixed size *buffer blocks*. Each block has a *pointer register* associated with it which points to the next block in the linked list. Each packet queue has a *head* and a *tail register* which point to the first and last block of the queue.

Buffer blocks are also key to supporting variable-length packets in the DAMQ buffer (supporting variable-length packets can dramatically improve the efficiency of a multicomputer communication network [Dall88]). A buffer supports variable-length packets by providing fast memory allocation and minimizing the memory wasted due to internal and / or external fragmentation. FIFO buffers implemented as circular memory arrays ‘‘automatically’’ minimize fragmentation — the free memory is always contiguous within the buffer. If the queues of a statically allocated non-FIFO buffer are implemented as individual FIFO buffers, the queues minimize fragmentation within themselves, but fragmentation in the buffer as a whole is exacerbated by the static buffer allocation. Since the DAMQ buffer dynamically allocates its buffer space, it requires a more sophisticated mechanism to manage memory allocation and to minimize fragmentation. The buffer blocks used to implement the DAMQ buffer’s linked lists also support variable-length packets. Fixed sized buffer blocks eliminate external fragmentation, and internal fragmentation is minimized by having the buffer blocks (eight bytes) be smaller than the maximum packet size (thirty-two bytes). Thus, the linked lists not only link together the packets which comprise a queue, but also the multiple buffer blocks which comprise a single packet.

Technically, the buffer blocks and linked list structure of the DAMQ buffer are artifacts of our implementation (Ch. 7); there are other ways to implement dynamically allocated multi-queue buffers. However, in the following chapters (4, 5, 6 and 8), the testbed used to evaluate network features includes eight-byte buffer blocks in its behavioral model of the DAMQ buffer. The buffer blocks were included because they impact the behavior of the DAMQ buffer with respect to

memory fragmentation when forwarding variable-length packets.

3.5. Supporting Multiple Classes of Packets

Section 1.1 discussed the desirability of supporting real-time communication. The idea is to allow a subset of the packets generated by the system to be designated ‘‘high priority’’, and to provide preferential treatment for these packets within the network, reducing both the average and the variance of the high-priority packet latencies. In Ch. 5, it is demonstrated that non-FIFO handling of packets is not sufficient to significantly reduce the ratio of worst-case to average latencies.

The DAMQ buffer achieves low average latencies by employing multiple packet queues to prevent packets which are destined for different output ports from blocking each other. An extension of this idea is to use an additional queue at each input port dedicated to high-priority packets. This allows high-priority packets to be routed through the switch as soon as they arrive in preference to any normal packets waiting to be transmitted out of the same output port.

This scheme requires that there be a means to identify critical data packets. We assume that the sender of a packet can mark it ‘‘high-priority’’ by setting a dedicated bit in the packet’s header byte. The goal of implementing a communication network with DAMQ buffers which have the high-priority packet queue is to continue to provide low average latency for all traffic while also providing relatively low worst-case latencies to a small percentage of high-priority packets [Tami88c].

Since the DAMQ buffer implements its packet queues as linked lists, adding an additional queue to the buffer requires little more than another head and tail

register. Since adding a single queue to store the high-priority packets in a buffer means that a queue may hold packets to different output ports, this scheme also requires that the *header registers* (the registers associated with each buffer block which store the packet headers) have two extra bits to store a output port identifier for the packet. Thus, support for multiple packet classes can be easily incorporated into the DAMQ buffer architecture.

Chapter Four

Buffer Performance Evaluation with Synchronous Networks

This chapter presents evaluations of the performance of communication networks composed of switches implemented with the DAMQ buffer. The goal is to determine the performance characteristics of the DAMQ buffer and evaluate various design parameters associated with this buffer. In order to make this evaluation, the DAMQ buffer's performance is compared to the performance of several other buffer types: FIFO buffers, two statically allocated non-FIFO buffer architectures (SAMQ and SAFC buffers), and a central buffer pool whose space is dynamically allocated to incoming packets.

Chapter 3 argued that the use of FIFO buffers reduces network performance due to the susceptibility of FIFO buffers to output port contention and that non-FIFO buffer architectures which utilize static buffer allocation inefficiently utilize their available buffer space. Further, a strong case as made for the difficulty of implementing statically allocated non-FIFO buffers (the need for pre-routing packets, etc.) and of implementing central buffer pools. Nevertheless, there are a number of reasons for comparing the performance of these particular buffers to the performance of DAMQ buffers. First, they facilitate the evaluation of specific architectural features of the DAMQ buffer. For instance, by including the SAMQ buffer in the comparisons, the impact of dynamic buffer allocation on the performance of the DAMQ buffer can be isolated. Second, some of thse buffer organizations (specifically, SAFC) may be worthy of consideration in their own

right. Finally, while the CBDA buffer presents impossible implementation requirements (as it is simulated in this chapter — see Sec. 4.1), it serves as a ‘‘pseudo-optimal’’ buffer architecture under uniform traffic conditions, providing an estimate of the available communication network performance which the other buffer architectures fail to tap.

The next section discusses the difference between *synchronous* and *asynchronous* communication networks and discusses why we present results from evaluations of synchronous networks in this chapter. Sec. 4.2 describes the Markov model used to evaluate single 2×2 switches, presents the results obtained from the model and examines these results. The following section describes the event-driven simulator used to measure the performance of synchronous multistage interconnection networks. This section describes the switch behavioral model used in the simulator, including the algorithms for discarding and blocking flow control and for arbitrating the crossbar of each switch. Secs. 4.4 and 4.5 present and discuss the results obtained by simulating networks of discarding and blocking switches, respectively, which are transmitting uniformly distributed traffic. Sec. 4.6 examines the performance of networks of blocking switches when communication traffic is not uniform. The chapter concludes with a summary of the results and analyses presented in this chapter.

4.1. Synchronous and Asynchronous Networks

This chapter and the following use analysis and simulations of a *synchronous* communication network in order to compare the performance of different buffer architectures. However, Chs. 6 and 8 perform their evaluations in the context of an *asynchronous* network. In the context of a multicomputer communication network, the terms *synchronous* and *asynchronous* do not refer to the presence or absence of a clock, but rather to the synchrony (or lack thereof) with which packets are transmitted.

In a synchronous network, time is globally divided into *stage cycles* [Yoon90]. A stage cycle is the time required for a packet to travel one hop in the network. Each switch of a synchronous network may transmit packets at each stage cycle boundary, with at most one packet traversing a link each stage cycle. In an asynchronous network, on the other hand, each switch operates independently of its neighbors and is capable of transmitting or receiving a packet during any clock cycle that an output or input port is available. Packets require multiple clock cycles to be transmitted received (as opposed to a single stage cycle), and the number of cycles is dependent upon the size of the packet.

The use of a synchronous network has several implications. First, virtual cut-through [Kerm79] is not supported. As was discussed in previous chapters, virtual cut-through can dramatically reduce communication latency in lightly-utilized communication networks. However, in a synchronous network, packets can only travel one hop in a stage cycle. Using a synchronous network also eliminates the possibility of transmitting variable length packets; since transmission service time is constant (one link per stage cycle per packet), it is not meaningful to consider

variable-length packets.

Asynchronous networks are used by most of the academic and commercial multicomputers built in recent years: the Intel Delta and Paragon [Lill91], MIT's J-Machine [Dall87b], *T [Nikh92], Alewife [Agar90], the Cray T3D [Oed], and others. Exceptions include the BBN Butterfly [Crow85] and Monarch [Rett90] machines, which use a synchronous networks. Further, the DAMQ buffer architecture was intended for use in an asynchronous network, with features specifically designed to support virtual cut-through and variable-length packets. However, simulations of synchronous communication networks have been widely used in research on packet switching networks [Dias81, Kuri89, Lang88, Dias89, Yoon90, Kuma84, Pfis85b, Lee86, Yew86, Kuma86, Knig89, Scot90, Tami88b, Tami92a, Tami92b]. For this reason and because a synchronous switch is simpler to model, this chapter evaluates the DAMQ buffer in the context of a synchronous network (as does Ch. 5, which examines the ability of various buffer architectures including the DAMQ buffer to support real-time communication).

4.2. 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 to 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* [Yoon90] packets either completely arrive or

completely depart. Since we assume a uniform packet size, the amount of buffer memory occupied by each packet is a constant. Therefore, the *packet slot* (the amount of memory occupied by a packet) is used as the unit of buffer storage in this analysis.

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 which is not currently transmitting a packet. This corresponds to the architecture described in Ch. 3, 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, when a 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, J is the set of states in which one of the input ports is full, and $P(k)$ is the probability of being in state k after any given stage cycle. Packets arrive at each input port of the switch with a probability equal to the applied traffic rate and with equal probability of being destined for 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 Tab. 4.1. 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 Tab. 4.1, 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. The single exception to this is the SAFC switch with six packet slots per buffer, operating with a 0.99 applied input traffic rate. This result demonstrates that, as the size of the buffer grows, the ability to dynamically allocate buffer space becomes less important. When large buffers make dynamic buffer allocation less relevant, the additional connectivity of the SAFC switch gives it the performance edge over the DAMQ buffer.

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. Further, 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

Buffer Type	Pkts per Port	Percentage of discarded packets vs. 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

Table 4.1: *Results of Markov Analysis.* A single 2×2 discarding switch. The percentage of discarded packets for a variety of buffer sizes (given as packets per input port) for various applied input traffic rates (the fraction of the raw input port capacity).

for the extra control circuitry needed for the DAMQ buffer [Fraz89] (Ch. 7).

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 packet 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 2×2 switches.

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. This is due to there being a theoretical limit to the throughput obtainable from FIFO buffers [Irla78]. Under uniform traffic conditions, a 2×2 switch implemented with FIFO buffers cannot support throughputs over 0.75 (consider that there are only two packets available for transmission on any given stage cycle, and each packet is equally likely to be destined for either output port). 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 Tab. 4.1, the DAMQ switch performs significantly better (lower discarding rate) than a SAMQ or SAFC switch with small buffers.

This relatively simple analysis of a single discarding switch has exposed two buffering fundamentals. The first is the impact of the performance restriction imposed by FIFO buffering. Even with a small number of packet slots, switches implemented with multi-queue buffers can exceed the theoretical maximum throughput of switches implemented with FIFO buffers — and the maximum throughput FIFO switches under uniformly distributed traffic decreases as the number of ports increases (0.65542 for a 4×4 FIFO). The second is the importance of dynamic buffer allocation when there are a small number of packet slots per input port.

4.3. The Simulator

We used a general purpose simulation tool written in Modula2 called *SIMON* [Swop86] to construct the simulator. *SIMON* provides facilities for the creation of *objects* (co-routines) and an event queue which allows the objects to pass messages to each other. The simulator uses three types of objects to create a communication network: *senders, switches* and *receivers*. We simulate a clocked system; each object in the system is synchronous, and the clock is global. As explained in Sec. 4.1, the network being synchronous is independent of the fact that the individual objects with the network are synchronous — the asynchronous network simulator described in Sec. 6.1 also uses a global clock.

Network traffic is stochastically generated. Each sender injects packets into the network using geometrically distributed inter-message delays. The destination address of each message is chosen randomly — in Secs. 4.4 and 4.5, a uniform distribution of the possible destination addresses is used. For Sec. 4.6, which explores network performance in the presence of a hot spot, each packet generated has probability h of being addressed to the hot spot, probability $1-h$ of having its address chosen from the uniform distribution. The simulator follows a set structure in performing the simulations. First, the simulations are run until x packets (in total) are received (reach their destinations). This number must be set high enough that, by the end of the x packets, the system is stabilized. At this point, the statistics-gathering variables are reset and the simulations continue to execute until n more packets are received. Finally, the simulation ends and the statistics are printed to a file. This chapter presents the results of simulating 64×64 omega networks [Lawr75] constructed from three stages of 4×4 switches. The network connects sixty-four processors (message generators) to sixty-four memory modules (message receivers).

The next section discusses results obtained using *discarding flow control*, while the two sections after it present the results of simulations of networks which perform *blocking flow control*. Under discarding flow control, decisions as to whether or not to transmit a packet are made without consideration of the state of the destination buffer. If there is not buffer space available to receive an incoming packet, then the packet is discarded. In this simulator, a switch discards a packet by transmitting it back to its original sender. Each sender object has a queue for packets which have been discarded — while a sender can generate a new packet

while this queue is not empty, it cannot transmit a new packet while there are packets in the discarded queue (i.e. senders give priority to packets which have been discarded).

Under blocking flow control, when a buffer becomes filled with packets, it signals this state to the neighboring switch(es) which transmit to that buffer. Senders / switches do not transmit packets to a switch which has asserted the blocking signal. Only after the signal is dropped can transmissions recommence. In the network simulated for this dissertation, if a full buffer transmits a packet on stage cycle i (and thus becomes unfull), the neighboring switch will be able to transmit to the once-full buffer on stage cycle $i+1$.

An important issue in 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 (they are multi-queue buffers which can only read queue at a time). 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 in 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 non-empty 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 [Stev86] to determine which queues within a buffer have held packets for a long period of time and should therefore get top priority. This is not an arbitration scheme which one could efficiently implement in a communication switch. Tamir and Chi present a detailed discussion of arbitration schemes for small $n \times n$ switches [Tami93].

Network performance is measured by the maximum throughput achieved and by the average packet latency at a given throughput. In a synchronous network, throughput is measured as the fraction of the network bandwidth utilized, which is equal to the average number of packets reaching the destination nodes per stage cycle divided by the number of destination nodes (i.e. $0 \leq thpt \leq 1$). Latency is calculated in *stage cycles* and is measured from the time of a packet's creation until it reaches its destination.

4.4. Evaluation of Discarding Switches via Event-Driven Simulation

Section 4.2 discussed results obtained from an analytical model of a single discarding 2×2 switch. This section extends the evaluation by simulating a network of discarding switches.

The results of simulating a network of discarding switches under uniform traffic are presented in Table 4.2. 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 $R \cdot (1-D)$.

The results in Tab. 4.2 show that, for a given buffer size and wide range of

Buffer Type	Pkts per Port	Percentage of Packets Discarded vs. Applied Input Traffic Rate								Max. Thpt.
		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

Table 4.2: *Discarding Percentage vs. Throughput.* 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).

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, for the buffer sizes examined. 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, non-FIFO handling of packets allows the switches implemented with SAMQ and SAFC buffers to operate at throughputs above the FIFO switch's theoretical maximum. 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 when dealing with uniformly-distributed traffic.

4.5. Evaluation of Blocking Switches via Event-Driven Simulation

The previous section described the results of simulating networks of discarding switches. This section explores the performance of networks using blocking flow control. The performance of 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 (i.e. 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. 4.1 and elsewhere [Dias81, Pfis85a].

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 Tab. 4.3. The throughput is reported as the fraction of the aggregate raw link bandwidth. This is the fraction

Buffer Type	Pkts per Port	Average latency vs. network throughput						Sat. Thpt
		0.10	0.20	0.30	0.40	0.50	sat.	
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
	4	3.14	3.38	3.79	4.65	9.34	13.14	0.51
	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
	4	3.14	3.36	3.68	4.16	4.91	10.66	0.71
	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
	4	3.13	3.29	3.50	3.80	4.19	9.71	0.80
	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

Table 4.3: *Latency vs. Throughput, Blocking Switches.* The performance of a 64×64 omega network composed of 4×4 blocking switches. Results obtained from simulation. Uniform traffic. Shown are the latencies of packets through the network for various network throughputs.

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 4.3 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, the buffer size does not have a significant impact on the latency. For the buffer sizes shown in the table, 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. If one were to increase the buffer sizes, as one approaches infinity, the performance of switches implemented with SAMQ and DAMQ buffers would be equal, and SAFC buffers better; static versus dynamic buffer allocation is irrelevant in extremely large buffers.

As shown in Tab. 4.3 and in Fig. 4.1, for these same switches, at a throughput of 0.50, the DAMQ network results 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.

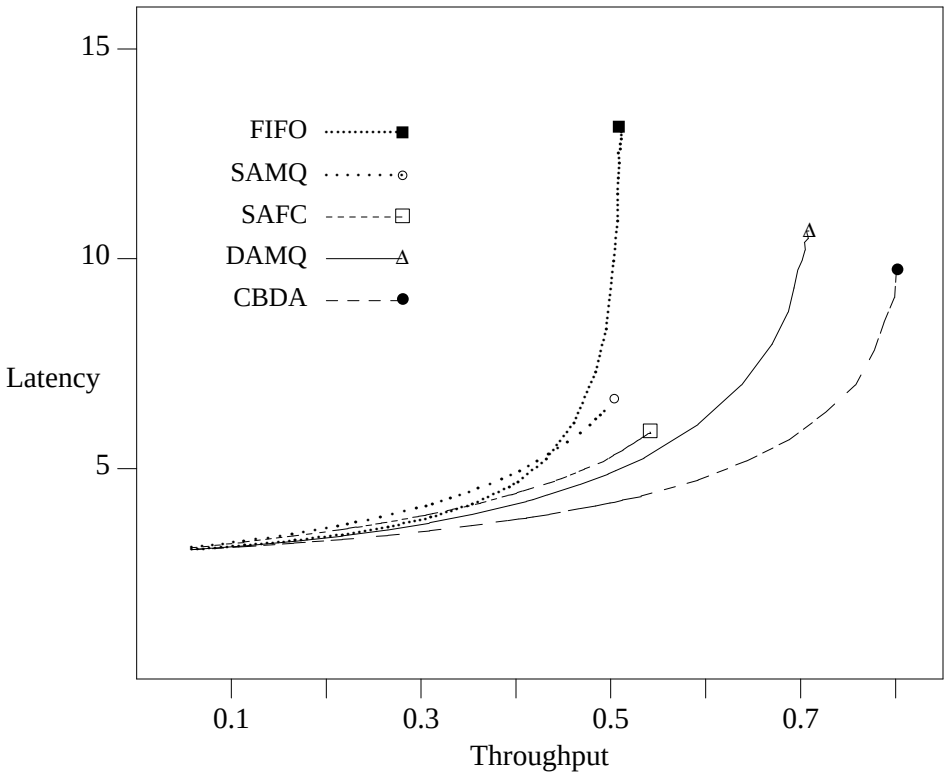


Figure 4.1: *Latency vs. Throughput.* The performance of a 64×64 omega network composed of 4×4 blocking switches with four packet slots per input port buffer. Results obtained from simulation. Uniform traffic.

Hence, the major advantages of the DAMQ buffer are (a) its ability to support high throughput communication and (b) its ability to provide low-latency communication at throughput rates which are approaching saturation.

As shown in Fig. 4.1, 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 allocated to four queues 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 (if not sooner). 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.

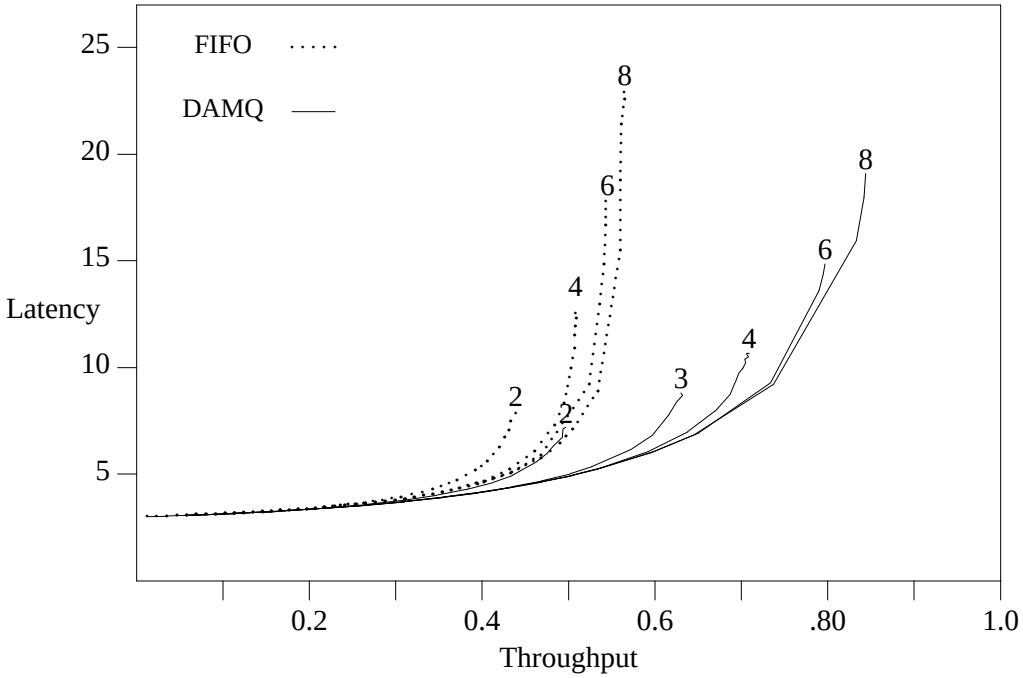


Figure 4.2: *Blocking Switches with Varying Buffer Sizes.* The performance of a 64×64 omega network composed of 4×4 FIFO or DAMQ blocking switches with several sizes of input buffers. Results obtained from simulation. Uniform traffic.

The results in Tab. 4.3 are similar to the results of the Markov model and the simulations of discarding switches — in all cases the benefit derived from non-FIFO handling of packets is the improved network performance under high network loads. At low throughputs, the FIFO switches perform as well as the SAMQ, SAFC, and DAMQ switches. However, multi-queue buffers can provide higher maximum throughput as well as lower latency at ‘‘moderate’’ throughputs.

This problem cannot be resolved by increasing the size of the FIFO buffers — increases beyond three or four slots results in very small increases in the maximum network throughput (as the work of ourselves and others [Dias81] has shown).

Multi-queue buffers, on the other hand, increase the maximum throughput far beyond what is achievable with FIFO buffers. This result is seen in Fig. 4.2, which shows the diminishing returns obtained from increasing the size of FIFO buffers. This figure also shows that implementing DAMQ buffers with only three packets slots results in significantly better performance than a FIFO buffer with eight slots. Hence, beyond minimal buffer space (two or three packet slots), it is more beneficial to allocate hardware resources to the more complex control of the DAMQ buffer than to additional buffer memory.

4.6. Non-Uniform (Hot Spot) Traffic

For some applications, the handling of non-uniform traffic by the network may be a critical factor in determining system performance [Pfis85a]. Of particular interest is ‘‘hot-spot’’ traffic, where a particular (‘‘hot’’) destination receives a higher percentage of the packets than any other destination. Tab. 4.4 presents the results of simulating networks composed of the various switches with hot-spot traffic [Pfis85a]. Five percent of the packets from all senders are sent to the ‘‘hot’’ destination, while the destinations 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

(≈ 0.24).

Buffer Type	Average latency vs. 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

Table 4.4: *Blocking Switches with Hot Spot Traffic.* A 64×64 omega network composed of 4×4 blocking switches. Four packet slots per input port. Five percent hot-spot traffic. Shown are the latencies of packets through the network for various network throughputs. Results obtained from simulation.

With hot-spot traffic (Table 4.4), the saturation throughput for all switches is significantly lower than with uniform traffic (Table 4.3). This is caused by the increased probability of contention within each switch for the output port on the path 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 [Pfis85a] 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 [Pfis85a].

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 bandwidth of the single link connecting the network to the hot destination. 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 possibility is to follow the design of the IBM RP3 multiprocessor [Pfis85b] and use two separate networks: one for general traffic and the second, a combining network [Gott83], 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 flow control to ‘‘throttle’’ the hot-spot traffic and the source [Scot90, Dias89]. In Ch. 8, we discuss the possibility of a flow control scheme to minimize the impact on the rest of the communication traffic in the network caused by hot spots and other forms of congestion.

4.7. Summary and Conclusions

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 both of these features are critical for supporting high throughput, low latency communication.

Also, we examined buffer performance in communication networks experiencing hot spot contention. We have shown that with hot-spot traffic that involves all the senders in the network, multi-queue buffering and dynamic buffer allocation 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.

Chapter Five
Supporting High-Priority Traffic

In Sec. 3.5, we discuss uses for the queues of a multiqueue buffer other than simply increasing the saturation throughput of the network or reducing the average latency. There are situations where one may desire ‘‘especially good’’ service for packets of a particular class. Examples given in Sec. 3.5 include messages between kernel processes whose latency impacts a large number of user processes, messages controlling remote i/o devices which may have particular timing constraints (real-time communication), and flow control messages which must be able to bypass the network congestion which it is attempting to resolve. This chapter evaluates the ability of the DAMQ buffer to provide support for ‘‘high-priority packets’’ which require ‘‘especially good service’’.

There are two aspects to the ‘‘especially good’’ service needed by high-priority packets. First, the latency of packets in this class must be low — significantly lower than the latency of ‘‘normal’’ packets. Second, the latency of packets in this class must be predictable (especially when being used for real-time applications). To be predictable, packet in this class must be oblivious to the throughput of ‘‘normal’’ packets in the network *and* have a low variance in latency. In other words, given a percentage of packets being marked high-priority, the average latency of these packets should remain relatively constant as the throughput of normal packets is varied, and the worst-case latency of the high-priority packets should remain near the average latency as the throughput of normal packets is varied.

In Sec. 4.5, we presented figures showing average latency vs. throughput for multistage interconnection networks. This performance metric does not present the information necessary in order to evaluate the effectiveness of multi-queue support for high-priority packets. Presenting the average and maximum latencies of high-priority packets for a simulation, however, is inconclusive. The maximum latency experienced by a given simulation is an increasing random variable; the longer the simulation runs, the closer the maximum latency will approach its theoretical bound (if such a bound exists).

This chapter utilizes the same simulator that was used in Ch. 4, but the 99^{th} *percentile latency* is used as the metric of a network's support for high-priority traffic (in contrast to *average latency*). The 99^{th} percentile latency of packets for a given simulation is the lowest latency of the slowest one percent of the packets. While the maximum packet latency of a simulation continues to climb long after the simulation has reached steady state, the 99^{th} percentile latency of the simulation remains relatively constant.

The problem of high worst-case latency through a network is demonstrated by the results of simulations of a 64×64 Omega network composed of switches with FIFO input port buffers. As shown in Fig. 5.1, the maximum latency through the network may be many times larger than the average latency. Furthermore, if the buffer size is increased from four packets slots to eight packet slots, the ratio of maximum latency to average latency is not improved, nor is the ration of 99^{th} percentile latency to average latency. Hence, increasing the size of the FIFO buffers is not a solution to the problem of very large worst-case latency through the network.

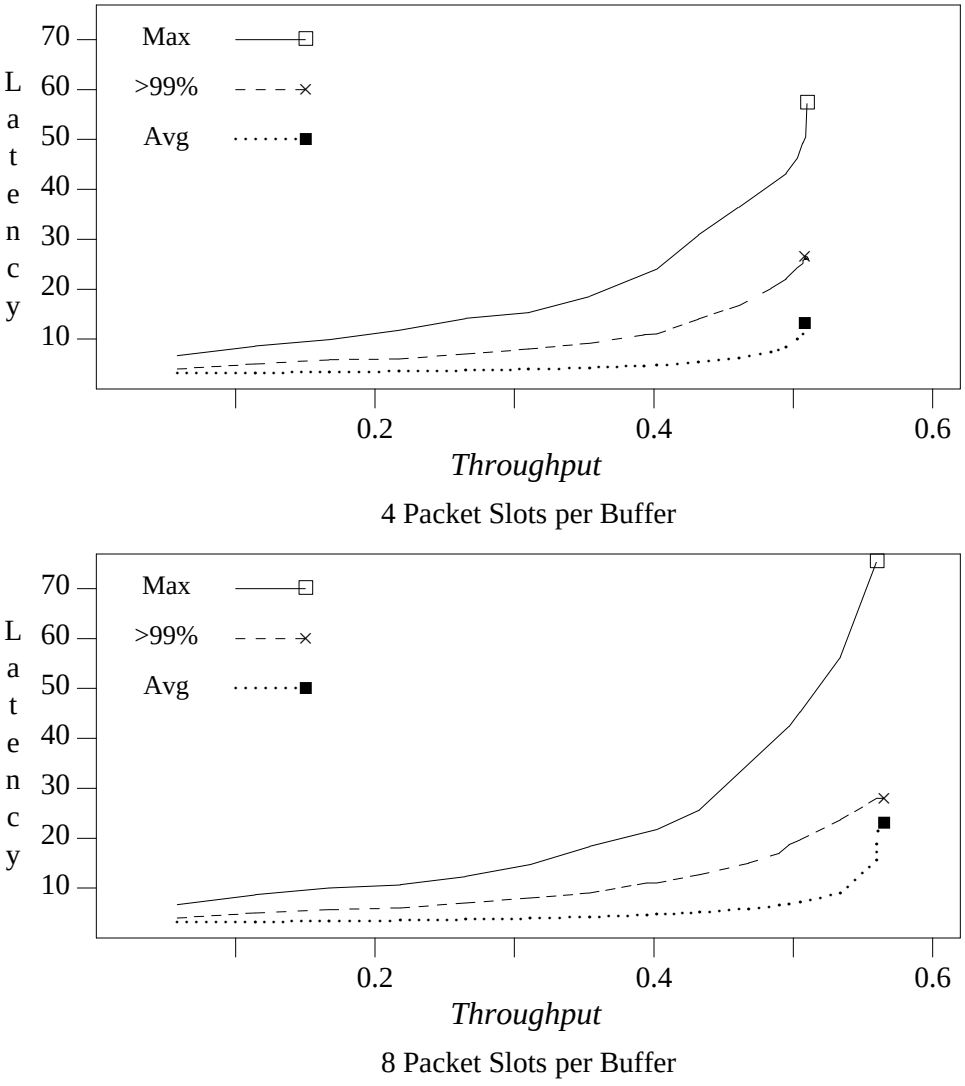


Figure 5.1: *Variance of Latency, FIFO Switches.* Maximum, 99th and Average Latency vs. Throughput. Four packet slots per buffer.

We have previously shown that networks composed of DAMQ, SAMQ, or SAFC switches perform significantly better than networks composed of FIFO switches (Figure 4.1). As shown in Table 5.1, these three practical switches as well as the CBDA switch perform better than FIFO switches in terms of worst-case

	Buffer Type	Throughput									
		0.10		0.20		0.30		0.40		0.50	
> 99% / max	FIFO	4.75	8.10	5.95	11.11	7.78	15.03	10.97	23.77	23.48	45.08
avg		3.14		3.38		3.79		4.65		9.34	
	SAMQ	5.76	9.80	6.75	13.88	9.00	19.79	12.00	26.74	17.88	39.25
		3.24		3.58		4.09		4.90		6.57	
	SAFC	5.38	10.10	6.73	13.78	8.16	20.06	11.00	21.60	14.38	32.83
		3.22		3.50		3.88		4.42		5.28	
	DAMQ	4.76	7.76	5.67	10.24	7.00	13.74	8.88	16.87	11.11	22.27
		3.14		3.36		3.68		4.16		4.91	
	CBDA	4.39	6.43	5.00	8.11	6.00	9.60	7.00	11.75	8.00	14.40
		2.76		3.29		3.50		3.80		4.19	

Table 5.1: *Variance of Latency, All Switches.* Maximum, 99th Percentile and Average Latencies vs. Throughput. Four slots per input port (a total of sixteen slots in the CBDA switch). Uniformly distributed traffic, 100% of packets normal.

packet latency under heavy load (throughput). However, even for these ‘‘better’’ switches the maximum (and 99th percentile) latencies are several times larger than their average latency. Even for the CBDA switch, which attempts to achieve fairness by arbitrating between incoming packets based upon the amount of time spent in the previous switch, the 99th percentile latency is double the average latency at 50% throughput, and the maximum latency is double that. These results indicate that the problem of large variations in the latency of packets, some of which may be ‘‘high priority,’’ cannot be solved by using different buffer organizations without special support for the high-priority packets.

In the remainder of Ch. 5 we investigate switch designs that provide support for high-priority packets. Alternatives designs are presented in the order of

increasing complexity and hardware cost. For each switch organization we describe specific design considerations, simulations results, and a brief summary of the major conclusions from the simulation results. The next subsection describes *priority arbitration*, in which the arbiter gives priority to queues which have high-priority packets at their head. Section 5.2 introduces the use of dedicated queues for high-priority packets (the support mechanism discussed in Ch. 3). Multiple dedicated queues for high-priority packets, one queue per output port, are discussed in Section 5.3. Finally, Sec. 5.4 evaluates the use of dedicated buffers for the high-priority packets (i.e. two physically distinct buffers per input port, one dedicated to high-priority packets).

5.1. Arbiter Support for High-Priority Packets

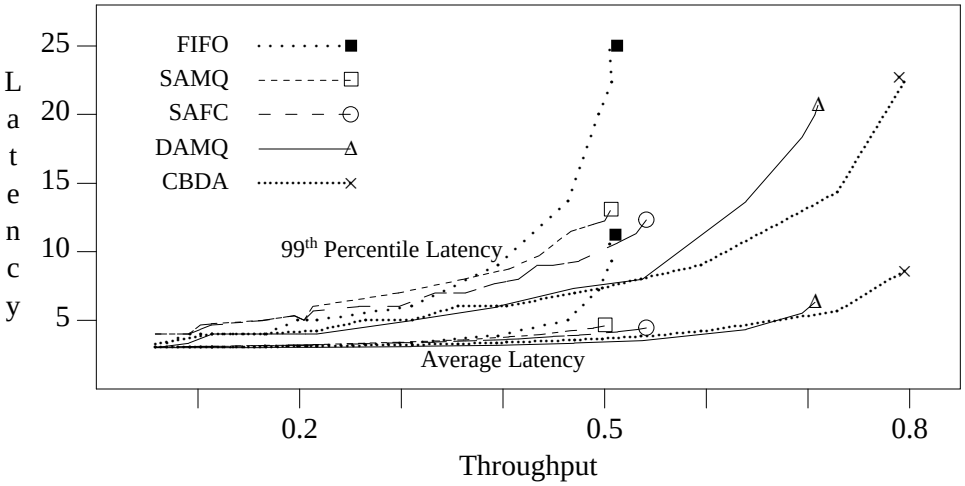
One way to support high-priority packets is to modify the crossbar arbiter — the controller that determines which input buffer is connected to which output port. The purpose of such a modification is to provide preferential arbitration to high-priority packets. Specifically, when determining the crossbar configuration, the modified arbiter will give priority to those queues which have a high-priority packet at their head. At each cycle, the arbiter first attempts to connect queues with high-priority packets at their heads to output ports, and then connects queues with normal packets at their heads to any remaining available output ports. For the CBDA switch, support for high-priority traffic in the arbiter means that if multiple packets destined for the same output port arrive simultaneously, the high-priority packets will be placed in the queue for that output port ahead of any normal packets. Furthermore, if the number of available slots in the buffer of the CBDA

switch is less than the number of senders to the switch, senders with high-priority packets at the head of their queue will be given priority.

In order to evaluate the effectiveness of the above scheme we ran simulations of the 64×64 omega network where a randomly selected 5% of all the packets generated were marked as high priority. Figure 5.2.a compares the 99^{th} percentile latency to the average latency of the high-priority packets only in networks using priority arbitration. Figure 5.2.b compares the 99^{th} percentile latency of high-priority packets for networks where the switches provide preferential arbitration for high-priority traffic to the 99^{th} percentile latency of packets in a network with no support for multiple packet classes. The results in Fig. 5.2.b indicate that, especially for networks with non-FIFO buffers, the modified switches improve high-priority performance over a system with no special support. However, as shown in Fig. 5.2.a, the 99^{th} percentile latencies are still much worse than the average latencies, especially under high network load. Supporting for high-priority packets in the arbiter alone is not sufficient; since the high-priority packets often wait in queues behind low-priority packets and receive preferential treatment for only the brief time they spend at the heads of queues. There must be a mechanism for the high-priority packets to bypass the normal packets.

5.2. Dedicated Queues for High-Priority Traffic

The SAMQ, SAFC, DAMQ, and CBDA switches achieve low average latencies by employing multiple queues of packets, one queue associated with each output port, to prevent packets which are destined for different output ports from blocking each other. A possible extension of this idea is to implement an



a) Priority arbitration, comparing 99^{th} percentile and average latencies.

	Buffer Type	Throughput				
		0.10	0.20	0.30	0.40	0.50
>99% Latency *prior. arbit.* / *normal switch*	FIFO	4.00	5.00	5.89	9.34	21.08
		4.75	5.95	7.78	10.97	23.46
	SAMQ	4.55	5.13	7.05	8.65	12.25
		5.76	6.75	9.00	12.00	17.88
	SAFC	4.27	5.15	6.06	7.78	10.21
		5.38	6.73	8.16	11.00	14.38
	DAMQ	3.59	4.00	4.89	6.09	7.63
		4.76	5.67	7.00	8.88	11.11
	CBDA	3.81	4.13	5.00	6.00	7.37
		4.37	5.00	6.00	7.00	8.00

b) Priority arbitration and no support for priority, comparing 99^{th} percentile latencies between the two.

Figure 5.2: *Priority Arbitration Performance.* Preferential arbitration for high priority packets. Four slots per input port. 5% high-priority packets. Throughputs shown are *total* network throughputs.

additional queue at each input port dedicated to high-priority packets. This should allow high-priority packets to be routed through the switch as soon as they arrive in preference to any normal packets waiting to be transmitted out of the same output port. This idea cannot be applied to a FIFO buffer since it has only a single queue; FIFO switches with the priority arbitration scheme discussed above are used in the following comparisons.

Implementing a dedicated queue for high-priority packets in the SAMQ and SAFC buffers involves adding additional storage (at least one packet slot) as well as a slight increase in the complexity of the arbitration circuitry. Since each queue in a SAMQ or SAFC buffer has at least one packet slot, the minimum size SAMQ or SAFC buffer in a 4×4 switch with support for high-priority traffic is five packet slots. On the other hand, adding the dedicated queue to the DAMQ buffer does not require any additional buffer storage — the available storage is dynamically shared between queues. A few (two or three) more control registers are needed to manage an additional linked list. In order to present a fair comparison of the performance of networks with the different buffer types, many of the results presented in this section were obtained using a total buffer size of five packet slots per input port. For the CBDA switch, special queues for high-priority traffic mean that the switch maintains, in its central buffer, a separate FIFO queue for each priority level at each output port. For each output port, packets are sent from the queue of normal packets only if the high-priority queue is empty. Furthermore, if the number of available slots in the buffer of the CBDA switch is less than the number of senders to the switch, senders with a non-empty high-priority queue are given priority.

We have simulated 64×64 omega networks constructed out of switches which

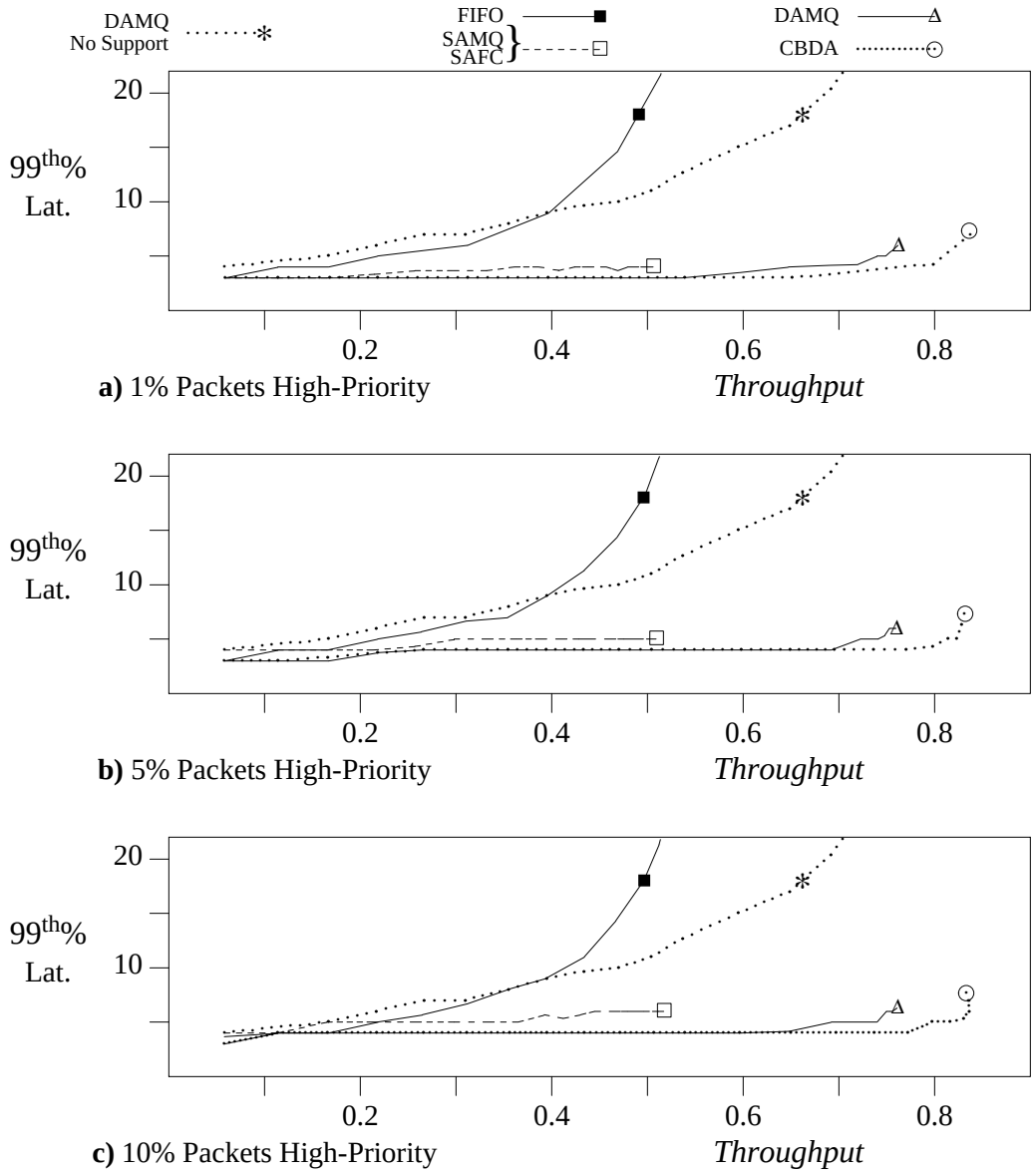


Figure 5.3: *Priority Queue, Various Percentages of High-Priority Packets.* The impact of a dedicated queue for high-priority packets at each input port. Shown are the 99th percentile latencies of high-priority packets vs. total throughput. Five slots per input port (a total of twenty packet slots in the CBDA switch).

include a dedicated queue for the high-priority packets at each input port. The 99th percentile latencies of the high-priority packets for networks with 1%, 5% and 10% high-priority traffic are shown in Fig. 5.3. For comparison, the figure also shows the 99th percentile latencies of a network with five-slot DAMQ buffers with no support for high-priority packets. The DAMQ buffer was chosen since, as shown in Tab. 5.1, this buffer results lower 99th percentile latencies than the other practical switches. These simulation results demonstrate the benefits of dedicated high-priority queues. Networks composed of the three practical switches which can support a separate high-priority queue achieved significantly lower 99th percentile latencies compared to those networks without separate high-priority queues. Furthermore, the 99th percentile latencies for high-priority packets are lower than the *average* latencies with switches that do not have support for high priority traffic (Figure 4.1).

Switches with DAMQ buffers outperform the SAMQ and SAFC switches by achieving lower 99th percentile latencies for high-priority traffic. The DAMQ switch is better at handling high-priority packets for the same reason it provides lower average latency for normal traffic — while only a fraction of the buffer space of the SAMQ and SAFC buffers is available to any single packet arriving at an input port (one fifth, in this case), the entire buffer is available to packets arriving at the DAMQ switch. Thus, the DAMQ switch better utilizes the available space, blocks incoming packets less often, is capable of handling higher throughputs, and delivers packets with lower latencies.

Figure 5.4 shows the 99th percentile latency of high-priority packets in a DAMQ network with high-priority queues, with priority arbitration, and with no

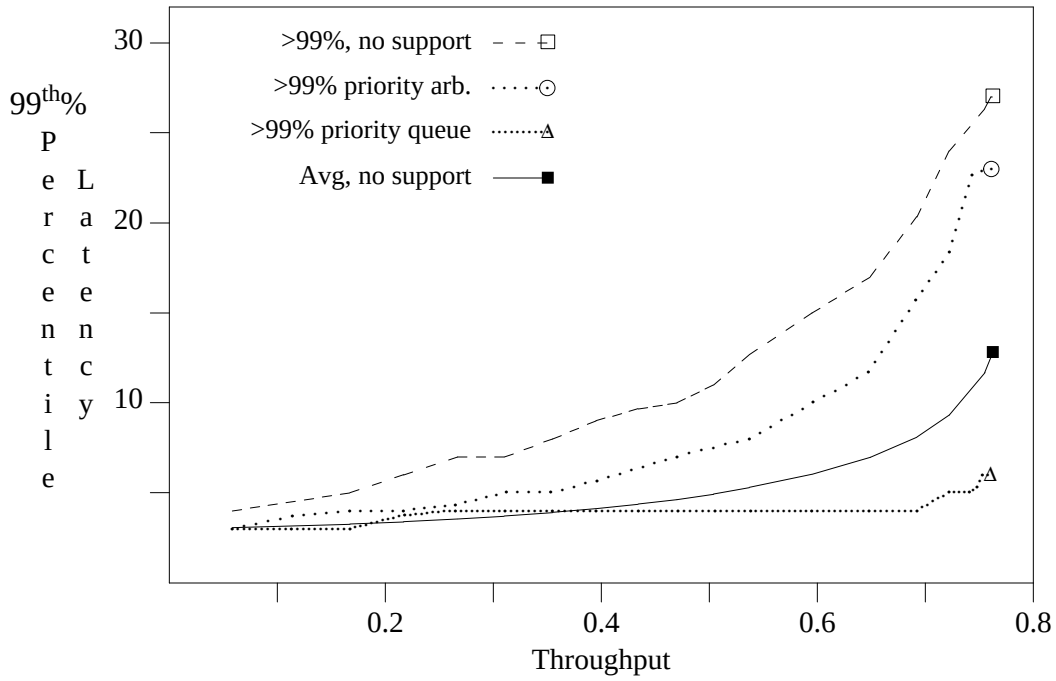


Figure 5.4: *Increasing Levels of Support for High-Priority Packets.* The average latency for normal traffic and the 99^{th} percentile latency for high-priority traffic without support, with priority arbitration and with priority queues vs. total network throughput. DAMQ buffers with five slots per input port. 5% high-priority traffic.

support for high-priority traffic. The average latency in a DAMQ network with no high-priority support is also shown. In all cases the buffers have five packet slots and 5% of the packets are high priority. With dedicated queues for high-priority traffic, the 99^{th} percentile latency for this traffic is stable at near the minimum network latency for a wide range of throughputs. With the dedicated queues, at medium and high throughputs, the 99^{th} percentile latency of the high-priority traffic is actually less than the average latency for normal traffic. An interconnection network with switches using DAMQ buffers with a high-priority queue can thus provide low worst-case latencies for high priority packets even

under network loads approaching saturation.

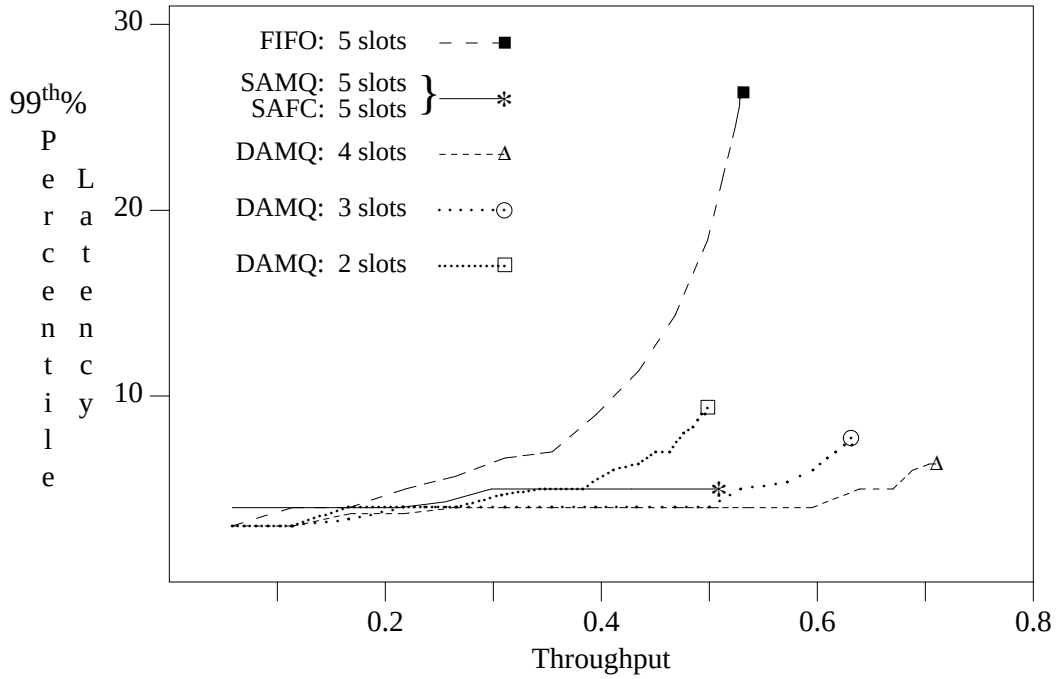


Figure 5.5: *Priority Queue, DAMQ Buffer Efficiency.* The 99th percentile latency of high-priority traffic vs. total network throughput. FIFO, SAMQ, and SAFC buffers with five packet slots per input port, DAMQ buffers with four, three and two packet slots per input port. 5% high-priority traffic. Dedicated queues for high-priority packets.

The advantages of the DAMQ buffer are further demonstrated by the results shown in Figure 5.5. The 99th percentile latencies of a network composed of DAMQ switches with two, three, and four slots are compared with the 99th percentile latencies for networks composed of FIFO, SAMQ, and SAFC switches with five slots. The DAMQ switch with only two slots significantly outperforms the FIFO switch with five slots. The DAMQ switch with three slots outperforms the SAMQ and SAFC switches with five buffer slots.

Figure 5.6 shows the impact of buffer size on the effectiveness of using dedicated queues for high-priority traffic with DAMQ buffers. For all buffer sizes, the dedicated queues result in significantly better performance for the high-priority packets than in a network where there is no special support for such packets. With very small buffers (two packet slots), even with dedicated queues, the 99^{th} percentile latency of the high-priority packets increase significantly as the total network throughput increases above 0.3. Furthermore, this 99^{th} percentile latency of the high-priority traffic is always higher than the average latency in a network without the dedicated queues. The reason for this rather poor performance with two slot buffers is that at most two queues can exist in the buffer at any stage cycle.

The maximum *total* throughput of the network increases significantly as the buffer size is increased from two to four slots and then six slots (Figure 5.6). With the larger buffers, more queues can exist in the buffer simultaneously and, at any point in time, the probability of a full buffer is lower than with two slot buffers. Hence, with dedicated queues for high-priority packets, it is much less likely that such a packet will be blocked due to a full buffer in the next stage. As a result, with four slot and six slot buffers with dedicated queues for high-priority traffic, the 99^{th} percentile latency of the high-priority packets remains very low (4 stage cycles) even as the network throughput increases up to 0.6. Furthermore, this 99^{th} percentile latency of the high-priority traffic is always *lower* than the average latency in a network without the dedicated queues.

This subsection has demonstrated the benefits of using dedicated queues for high-priority packets. DAMQ buffers were shown to be a particularly effective mechanism for exploiting such dedicated queues. We have now shown that

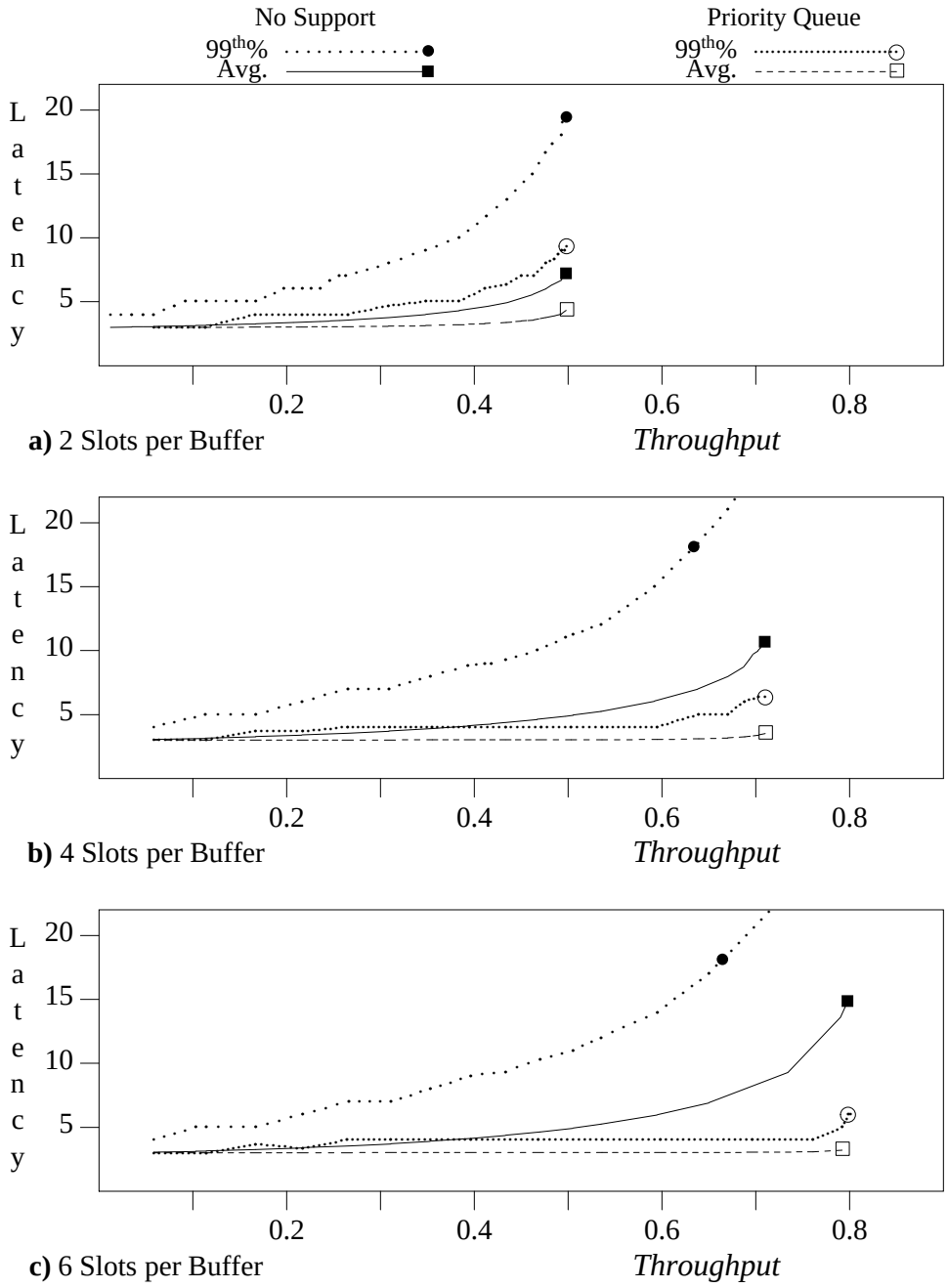


Figure 5.6: *Impact of Varying Buffer Size, DAMQ Buffers.* The average and 99^{th} percentile latencies for the 5% high priority traffic. Network with no support for high-priority packets and network with dedicated queues.

DAMQ buffers achieve higher performance than other practical buffers for high-priority traffic and for normal traffic (Sec. 4.5). Thus, for the rest of this chapter, DAMQ buffers with dedicated queues for high-priority traffic, will serve as a benchmark against which alternative designs will be compared.

5.3. Multiple Dedicated Queues for High-Priority Traffic

The previous subsection proposes support for high-priority traffic using, at each input port, a DAMQ buffer with a separate queue for high-priority packets. For normal traffic, DAMQ buffers achieve superior performance to FIFO buffers by, at each input port, providing a separate queue for each output port [Tami88b]. Hence, it should be possible to achieve better performance for high-priority traffic by providing, at each input port, a dedicated queue for high-priority traffic for each output port, in addition to the separate queues for normal traffic. With this scheme, for a 4×4 switch, each buffer will contain eight queues, instead of the five queues proposed in the previous subsection. The additional hardware required by this scheme may be worthwhile for applications where minimizing worst-case latency is particularly critical. SAMQ and SAFC switches with multiple dedicated queues for high-priority traffic require even more hardware since additional buffer storage is needed for each additional queue. Since, in addition, DAMQ switches with high-priority queues have already been shown to achieve superior performance compared to SAMQ and SAFC switches, only DAMQ switches are considered in this subsection.

Figure 5.7 compares the 99^{th} percentile latencies of the high-priority traffic with eight-queue and five-queue DAMQ switches, varying the total throughput and

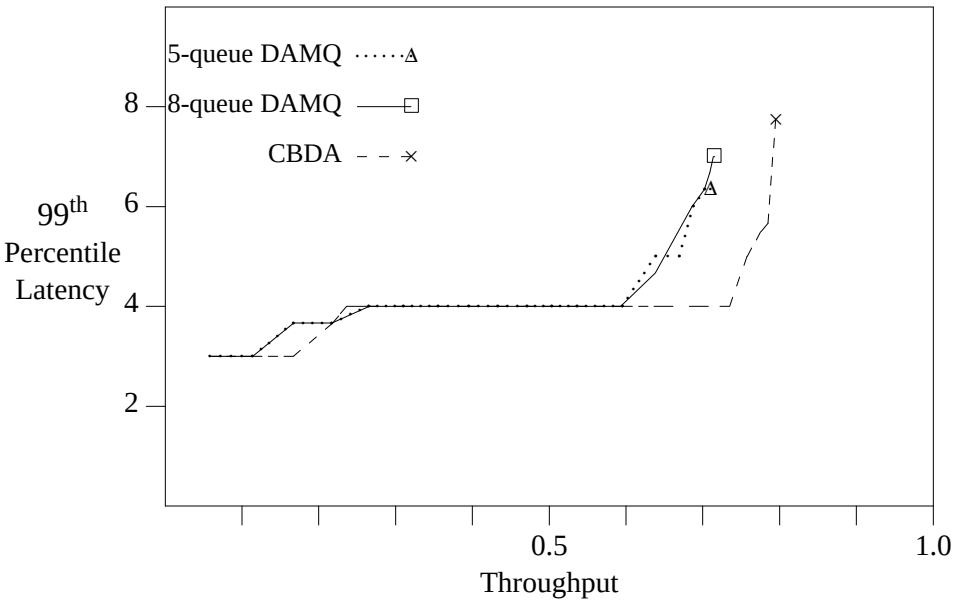


Figure 5.7: *Five-Queue and Eight-Queue DAMQ Buffers.* The 99th percentile latency of high-priority traffic vs. the total network throughput. Four packet slots per input port. 5% high-priority traffic.

keeping the percentage of high-priority packets constant at 5%. The performance of a network with CBDA switches is shown as a reference point for the level of performance that can be achieved. Under these conditions, the two DAMQ switches behave almost identically. The reason for this is that multiple queues are beneficial only when there are more than one packet in the buffer simultaneously. With only 5% high-priority packets it is rare that more than one high-priority packet is in a buffer at any instant.

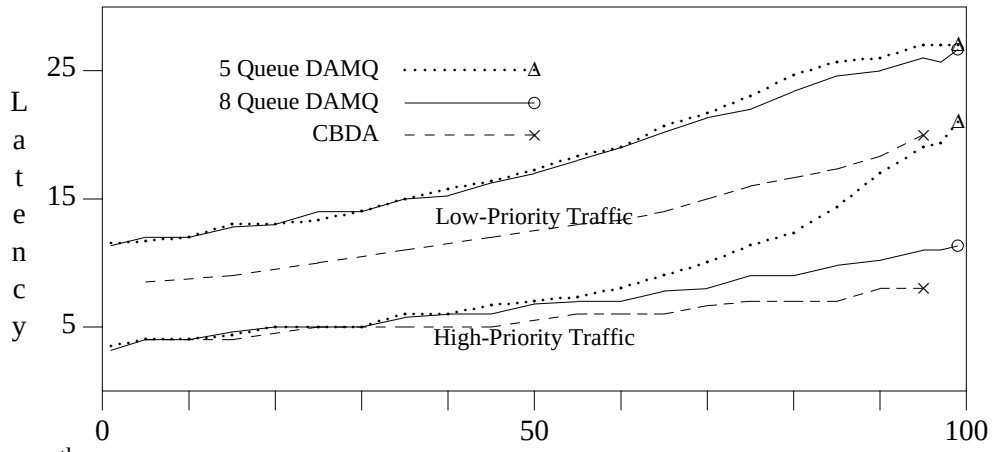
In a network constructed from switches that support high-priority traffic, it is useful to know up to what level (proportion) of high-priority traffic will the separate queues be able to maintain good worst-case performance, i.e., low 99th percentile latency, for the high priority packets. To answer this question we have

simulated the 64×64 omega network of four-slot DAMQ buffers under a constant total network throughput of 50% with varying proportions of high-priority traffic. It should be noted that without support for high-priority traffic, a network of DAMQ switches operating at this throughput achieves an average latency of 4.9 stage cycles and a 99th percentile latency of 11.1 stage cycles (Table 5.1).

Figure 5.8 shows the 99th percentile and average latencies of both the high-priority and low-priority (normal) traffic for a varying percentage of high-priority traffic. For a low percentage of high-priority packets, the five-queue DAMQ buffers perform well, i.e., result in low 99th percentile latency for the high priority traffic. With up to 18% high priority packets, the 99th percentile latency for such packets is below the overall average latency for a standard network using switches with DAMQ buffers (Table 5.1).

The average latencies and 99th percentile latencies of the low priority (normal) packets (Figure 5.8) indicate that, for low percentages of high-priority packets, the support for high-priority traffic has little impact on network performance with respect to normal traffic. Specifically, as the percentage of high-priority traffic varies from 0% to 10%, the change in average latency of the low priority traffic is negligible.

As the proportion of high-priority packets increases beyond 75%, the 99th percentile latency of a network with the five-queue DAMQ buffers is *larger* than 11.1 stage cycles, i.e., larger than the 99th percentile latency for the network composed of switches without support for high-priority traffic (Table 5.1). The reason for this behavior is that the DAMQ switch has four queues at each input port for normal traffic, but only a single queue for high-priority traffic. Hence, with



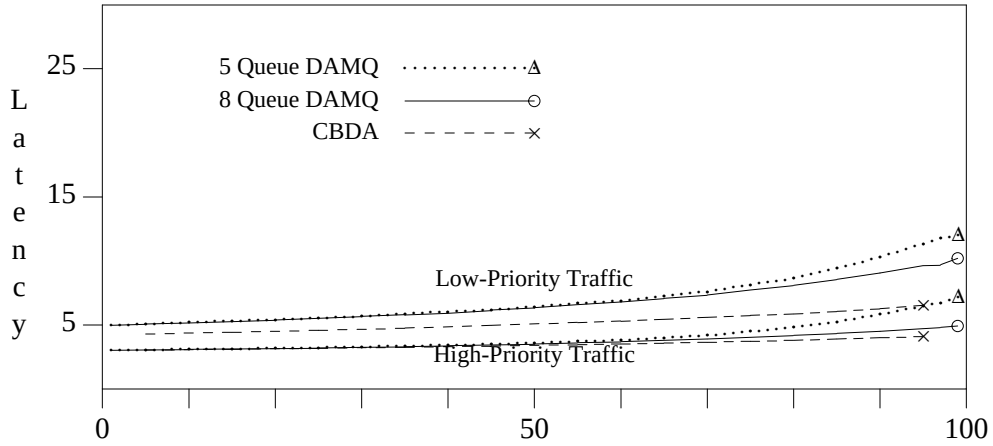


Figure 5.8: *Varying Percentage of High-Priority Packets.* Shown are 99th percentile latencies and average latencies of high-priority and low-priority traffic for a fixed total network throughput of 50%. Four packet slots per input port. Five-queue DAMQ switches, eight-queue DAMQ and CBDA switches.

respect to high-priority traffic, the DAMQ switch with a high-priority queue performs as a FIFO switch. As shown in Fig. 5.1.a, at a throughput of 0.37, the 99th percentile latency of a network with FIFO switches is 10 stage cycles. With the network of DAMQ switches with high priority queues, 0.50 network

throughput with 75% high priority traffic, the absolute throughput of high priority traffic is 0.375. Since part of each input buffer is occupied by normal packets, it is not surprising that the high priority traffic receives worse ‘‘service’’ (higher 99^{th} percentile latency) than the packets in the FIFO network with a throughput of 0.37.

Figure 5.8 shows that when more than 50% of the packets are high-priority, the eight-queue DAMQ switch performs significantly better than the five-queue DAMQ buffer, maintaining a relatively low 99^{th} percentile latency for the high-priority traffic. Furthermore, the improved performance with respect to high-priority packets does *not* come at the expense of the performance for low-priority packets. If the high-priority traffic is less than 50% of the total network load, there are no benefits to using the eight-queue DAMQ buffers. Eight-queue DAMQ buffers should be considered only if the *majority* of the packets are high priority. In this latter case, a better solution might be to use five-queue DAMQ buffers with a single queue dedicated to the low-priority traffic and four queues dedicated to high-priority traffic.

5.4. Dedicated Buffers for High-Priority Traffic

In the previous two subsections we have considered the use of one or more dedicated queues for high-priority packets and demonstrated the advantages of using DAMQ buffers that include such queues. With the DAMQ buffers, the normal packets and high-priority packets share the available buffer storage, even though they are organized in separate queues. Another design option for supporting high-priority traffic is the use of two independent buffers at each input port — one for normal packets and one for high-priority packets. Examples of

such a scheme include the use of an n slot DAMQ buffer for normal packets together with an m slot FIFO buffer for high-priority packets, or an n slot DAMQ buffer for normal packets with an m slot DAMQ buffer for high-priority packets. The motivation for such a scheme is to isolate the high-priority buffering from the normal packet buffering. This would prevent a situation in where a high-priority packet could not be forwarded to the next stage due to a full buffer containing normal packets. As with high-priority queues, for a synchronous network, the high-priority traffic does not compete for link bandwidth with the low-priority traffic since, whenever there is a conflict, the high-priority traffic is always given priority over the low-priority traffic. Thus, with dedicated buffers for high-priority traffic, network performance with respect to such traffic will be *independent* of low priority traffic.

We have evaluated schemes for using dedicated buffers for high-priority packets by comparing their performance, with respect to high-priority traffic, to the performance of a network of DAMQ switches where the buffer storage is *shared* between the high-priority traffic and the normal traffic (Sec. 5.2). For a varying percentage of high-priority traffic, the comparisons were done for a fixed total network throughput of 50% (at which, as shown in Fig. 4.1, a network of switches with four-slot FIFO buffers is already in saturation). We have previously shown that the five-queue DAMQ buffer with the dedicated high-priority queue provides good support at a low implementation cost for high-priority traffic. At 50% total network throughput, the network composed of switches using these buffers is quite heavily loaded, so contention for buffer storage between the high-priority and low-priority traffic might be expected. Hence, this network serves as a good

benchmark, relative to which dedicated buffer schemes can be compared.

We have considered several input port organizations based on a four-slot four-queue DAMQ buffer for normal packets and a dedicated buffer for high-priority packets. DAMQ buffers for normal packets are assumed since they have been shown to have significant performance benefits for normal traffic [Tami88b]. In choosing the buffer for high-priority traffic, we must consider the hardware cost for additional storage and control, as well as performance. Single-slot FIFO buffers are a low-cost choice for the dedicated high-priority buffers. Multi-slot high-priority buffers can improve the performance of dedicated high-priority buffer schemes, but require more hardware. In particular, we have considered the use of dedicated two-slot FIFO buffers and two-slot DAMQ buffers for high-priority traffic.

Figure 5.9 shows the performance of networks with the dedicated high-priority buffer schemes described above as well as the performance of networks with DAMQ buffers which include dedicated queues for high-priority traffic but where storage is shared by all traffic. The simulation results presented are for a fixed total network throughput of 50%. For varying percentage of high-priority traffic, Figure 5.9.a shows the 99^{th} percentile latency of high-priority packets, while Fig. 5.9.b shows the 99^{th} percentile latency of normal packets. Since none of the three shared DAMQ buffers is in its saturation region, they all result in identical performance. With a one slot dedicated buffer, handling of high-priority packets is poor, despite using more total buffer space than the shared four-slot DAMQ buffers. Performance with respect to the normal packets is identical with all the schemes for the proportion of high-priority packets of interest (under 30%).

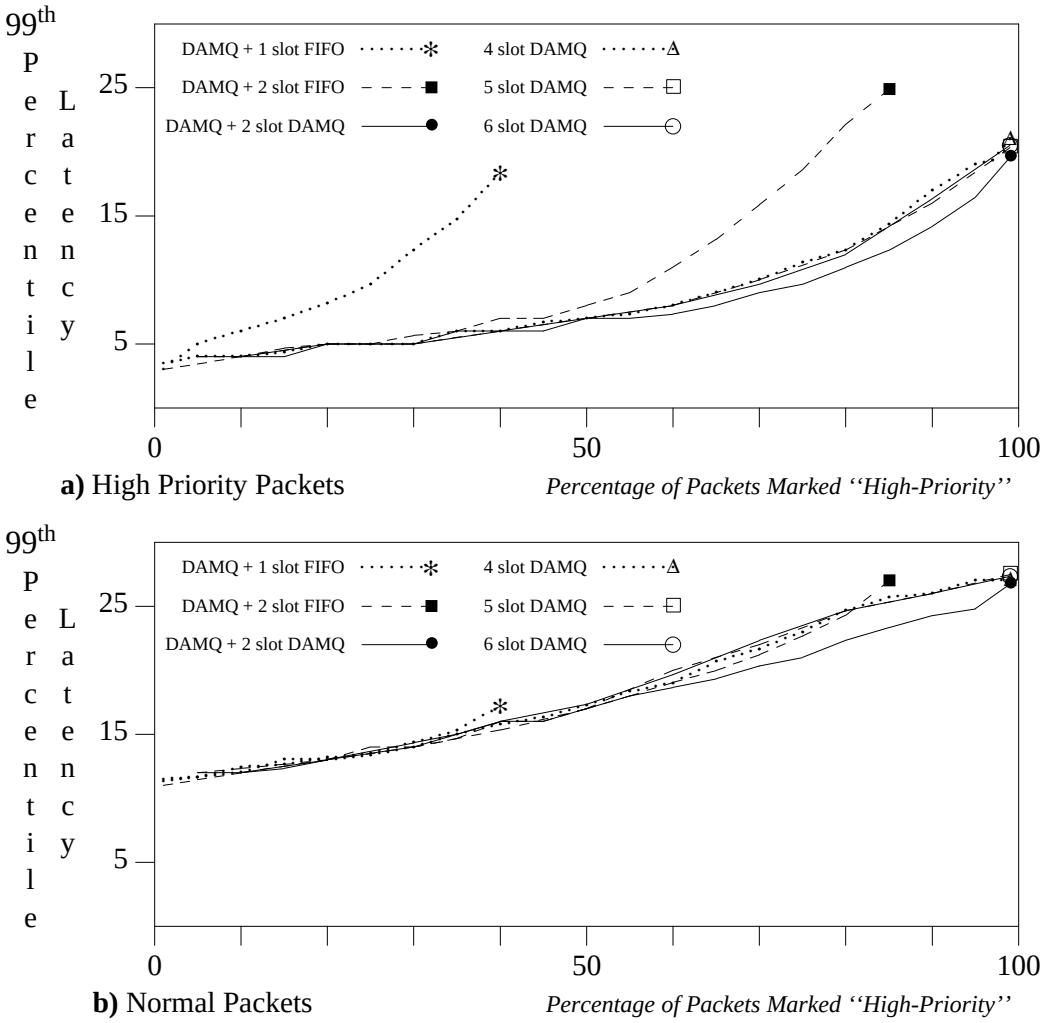


Figure 5.9: *Dedicated Buffers for High-Priority Packets.* The 99th percentile latency of high-priority and of normal packets vs. the percentage of high-priority traffic. The total network throughput is 0.5. Comparing dedicated buffer (4-slot, 4-queue DAMQ plus another) to dedicated queue (5-queue DAMQ) support.

At a low percentage of high-priority traffic (under 30%), the shared DAMQ buffers deliver equal performance to the performance of the two slot dedicated buffer schemes, despite the fact that the shared buffers are also supporting the total network throughput of 0.5. Since DAMQ buffers are already used to achieve high

performance for normal traffic, very little additional hardware is required to support dedicated queues. Dedicated two-slot buffers require significantly more hardware for both storage and control. In addition, the two buffers schemes are less likely to adapt well to a variety of conditions (such as very high throughput with *no* high-priority packets) due to the *static* partitioning of the available storage. Thus, dedicated buffers for high-priority traffic should be considered only if there is some reason why DAMQ buffers cannot be used. Under these conditions, it is useful to know that similar performance, with respect to high-priority traffic, can be achieved using this alternative approach.

5.5. Summary: Support for High-Priority Traffic

In order to use large multiprocessors and multicomputers for real-time applications, it must be possible to guarantee, with a high degree of confidence, that high-priority traffic can be transmitted through the network with low specified latency. With conventional interconnection networks used in multiprocessors and multicomputers, the worst-case latency of traffic through the network can increase dramatically as the load on the network increases. This may prevent the use of these interconnection networks for critical real-time applications or force their use with very low utilization (and thus high cost/performance ratio) in order to guarantee low maximum latency.

For interconnection networks composed of small $n \times n$ switches, we have shown that simply increasing the size of conventional buffers in the switches does not result in improved performance for high priority packets. There is thus a fundamental need for new buffer organizations which support high-priority

packets. We have developed a technique for efficiently supporting high-priority traffic, while maintaining good performance for normal traffic. Our scheme is based on a small modification of the *dynamically allocated multi-queue* (DAMQ) buffer. The modifications to the DAMQ buffer in order to support high-priority traffic involve a few additional control registers and somewhat more complex arbitration of the crossbar switch. Overall these modifications are expected to require only a small percentage increase in total buffer area.

We have evaluated alternative approaches to providing support for high-priority packets in $n \times n$ switches. This evaluation was based on implementation complexity and simulation studies of a multistage interconnection network transmitting packets with two levels of priority. Our simulations have demonstrated five key points. (1) In a conventional network with a single priority level for all packets, worst-case latency for packets can be several times higher than average latency. Hence, there is a need to identify and provide preferential treatment to those packets for which fast service is particularly important. (2) Using the priority of packets as the determining factor in arbitrating contention within each switch does not, on its own, provide sufficient support for high-priority traffic. Such arbitration does not reduce the 99^{th} percentile latency of the high-priority packets to the level of average normal packet latency, even under moderate network load. (3) As long as the proportion of high-priority traffic is low, dedicated queues for high priority traffic reduce the 99^{th} percentile latency of the high-priority packets to the level of the average latency for normal traffic under a wide range of network throughputs. (4) When the proportion of high-priority traffic is large and the network is heavily loaded, multiple dedicated queues for

high-priority traffic can reduce the 99th percentile latency relative to the single dedicated queue approach. However, this performance advantage is marginal and the associated implementation cost is relatively high. (5) Dedicated buffers for the high-priority traffic can provide the same performance advantage as dedicated queues in shared DAMQ buffers. However, since DAMQ buffers are needed to maximize performance for normal traffic, the implementation cost for dedicated buffers is much higher than the cost of adding dedicated queues to DAMQ buffers. Hence, there is no reason to use dedicated buffers.

Our results indicate that the DAMQ buffer with a single dedicated queue for high-priority packets provides support for high-priority traffic which is superior to the support provided by alternate switch designs based on a dedicated high-priority queue. This resulting network performance, with respect to high-priority traffic, is very close to the performance of a network based on ‘‘ideal’’ switches for which a practical implementation is not feasible. Hence, given the low hardware overhead of the scheme based on a DAMQ buffer with a single high-priority queue, it is clearly the preferable option.

Chapter Six

Buffer Performance Evaluation with Asynchronous Networks

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

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

This chapter evaluates the performance of the DAMQ buffer in an asynchronous network. In the next section, the simulator which generated the performance numbers discussed in this chapter is described. In Sec. 6.2, the performance of the DAMQ buffer in an asynchronous multistage interconnection network (omega topology) is evaluated. We compare the latency and throughput

of traffic in networks composed of DAMQ switches to that of networks with other buffer architectures. Also, the performance of the asynchronous omega networks are compared to the synchronous network results reported in Ch. 4. Finally, in Sec. 6.3, the simulator is used to demonstrate the ability of the DAMQ buffer to implement deadlock-free dimensional routing and support high communication throughput for a two-dimension torus network.

6.1. Simulating an Asynchronous System

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

6.1.1. Network Architecture

The simulator was modified to allow topologies other than an omega network. Currently, *topT* can be set to omega, two-dimensional torus, two-dimensional mesh or hypercube (results from the first two topologies are presented in this dissertation). Networks of various sizes in these four topologies can be established by setting the *sys_size* parameter. The size of the switches (the number of input

and output ports per switch) can be set via the variable *switch_size*. These variables are not independent: for tori and meshes, which are two-dimensional and square, $switch\_size = 5$; for an omega network, size of the network (the number of senders and receivers) is a power of the size of the switches ($sys\_size = switch\_size^{n}$) and, for hypercubes, $sys\_size = 2^{switch\_size}$.

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

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

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

Table 6.1: *Parameters to the Asynchronous Simulator.*

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

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

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

6.1.2. Packet Injection Parameters

Packets can be injected into the simulated network either stochastically or deterministically (i.e. trace-driven simulations). Deterministic packet injections are controlled by a file which specifies the size, source, destination and time of creation for each packet which traverses the network. To date, this feature has only been used for debugging the simulator. For stochastic packet creation there are a number of parameters. First, there is the inter-message delay (*msgint*) — the time

from when one packet is transmitted to when the next message is created. Three delay distributions are supported by the simulator: constant (each message is created a fixed number of clock cycles after the previous is injected into the network), uniform (the delay varies uniformly over a specified range) and geometric. All of the results presented in this dissertation use the geometric inter-message delay distribution.

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

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

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

6.1.3. Sender Grouping

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

6.2. DAMQ Buffer Performance: Asynchronous MIN

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

6.2.1. Constant-Length Packets, Asynchronous Network

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

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

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

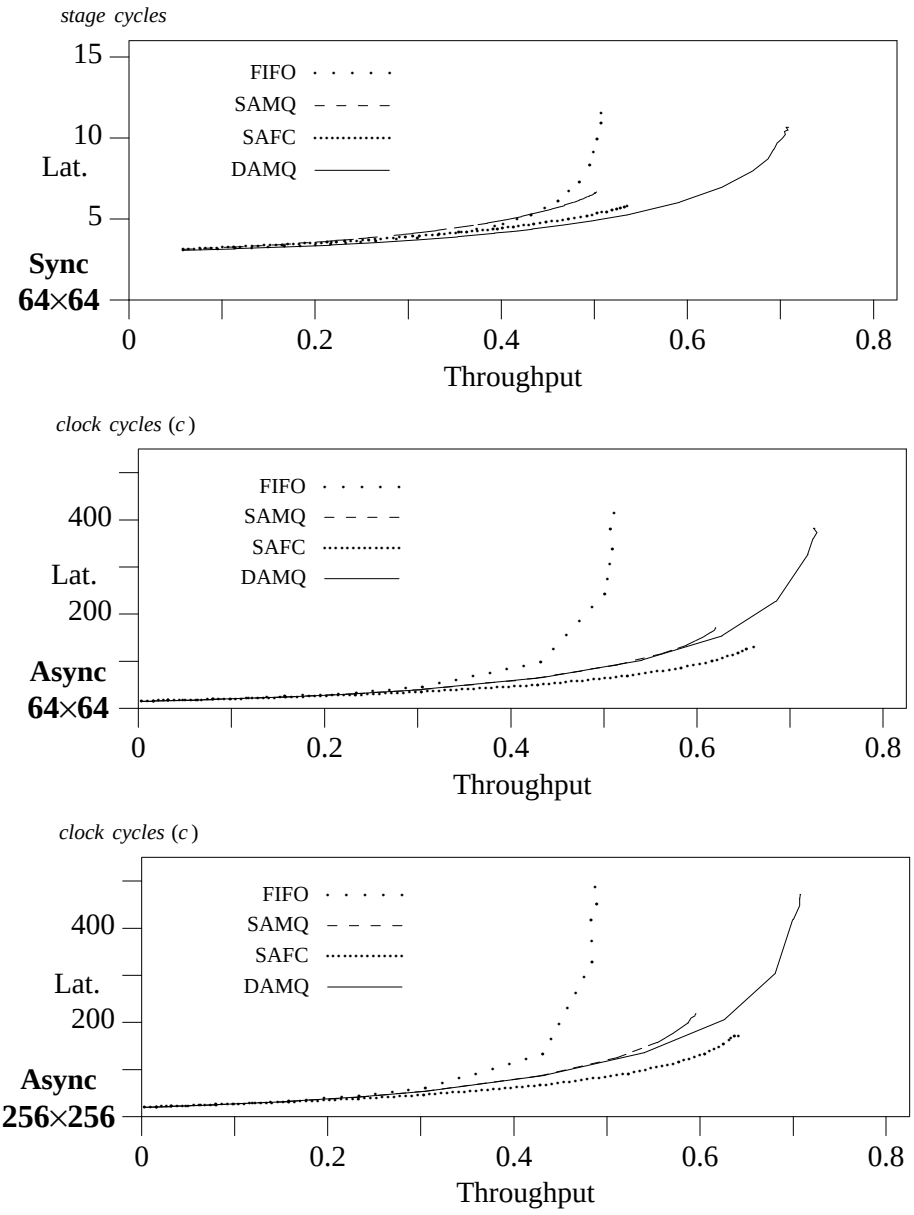


Figure 6.1: *Synchronous vs. Asynchronous Protocols, Omega Network.* Average latency vs. throughput results for 64×64 omega network with synchronous protocol (also in Fig. 4.1) and 64×64 and 256×256 omega networks with asynchonous protocol. 4×4 switches, thirty-two-byte packets, storage for four packets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

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

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

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

The reason for the increase in the saturation throughput of the SAMQ and SAFC networks has to do with a reduction in the back pressure experienced by the switches in the first stage of the network. As was seen in Tab. 4.1, small statically allocated buffers discard or block packets (depending upon the flow control mechanism) even at low applied loads; their buffer utilization is inefficient relative

to dynamic buffer allocation. Virtual cut-through increases the effective buffer size by reducing the amount of time packets spend in buffers. At saturation throughput, a relatively small percentage of the packets traversing the network experience virtual cut-through. The virtual cut-through that does occur, occurs in the latter stages of the network, which experience a lower applied load, shorter packet queues and less back pressure than does the first switching stage. FIFO- and DAMQ-buffered networks, on the other hand, experience little or no reduction of first-stage back-pressure in going to an asynchronous network. Referring again to Tab. 4.1, these buffers discard (or block) packets with a much lower probability at low throughputs, and so receive little benefit from virtual cut-through when operating at saturation throughput.

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

6.2.2. Self-Synchronizing Networks

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

This phenomena is illustrated by the solid line of the graph in Fig. 6.2. This

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

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

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

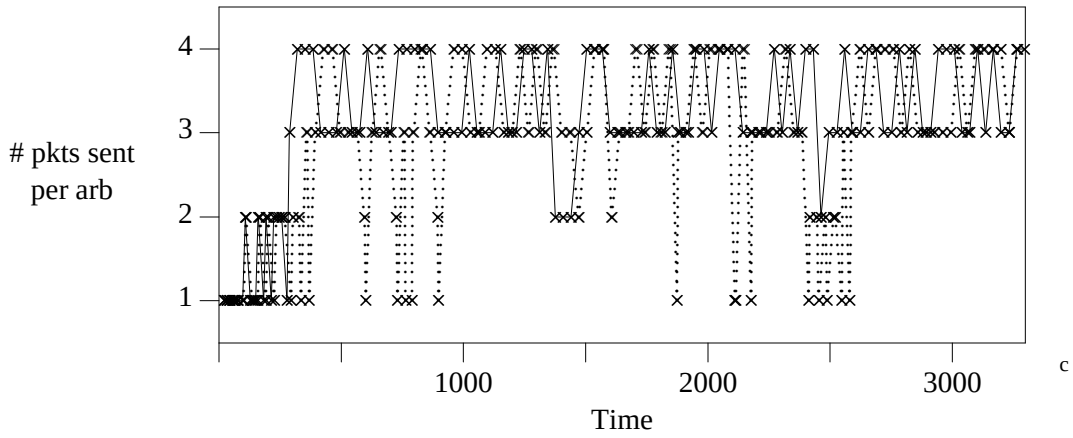


Figure 6.2: *Self-Synchronizing DAMQ Switches.* Results of a 4×4 network (a single DAMQ switch), transmitting thirty-two byte packets at saturation throughput. Shown is the number of packets transmitted per arbitration vs. time (clock cycles) for networks with zero cycles of link idle time between packets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

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

In order to visualize this loose synchrony, we defined a logical arbitration $arbitration^L$ to be a set of crossbar arbitrations such that each arbitration occurs on the same cycle as another member of arbitration^L or *link_rest_time* clock cycles from another member of arbitration^L. Thus, if $link\_rest\_time = 2$, and if buffer B_a begins a transmission on cycle t_x, and buffer B_b begins a transmission on cycle t_{x+2}, then B_a and B_b have achieved loose synchrony, the arbitrations which occurred on t_x and t_{x+2} are members of the same arbitration^L, and the number of

packets transmitted on this arbitration^{L} is two. Further, if B_c begins a transmission on cycle t_{x+4}, and B_d begins one on cycle t_{x+6}, then *four* packets where transmitted on this arbitration^{L}. On the other hand, if the transmission from B_b were to occur on t_{x+1}, then B_a and B_b are not operating in synchrony, even though the times in which their transmissions begin are even closer together, and the arbitrations for B_a and B_b are members of different arbitration^{L}s.

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

One could prevent a circuit from self-synchronizing in the presence of constant-length packets by not using the same link rest time after every packet. This would be detrimental to the network performance, however. When the network synchronizes, it gives the arbitration logic more buffer / output port

pairings to choose from, which improves port utilization. Evidence of this is provided simply by the fact that, when buffers become synchronized, they go from being idle to transmitting, and buffers lose synchrony when they go from transmitting to being idle.

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

6.2.3. Variable-Length Packets, Asynchronous Network

In this section, the performance of multistage interconnection networks transmitting variable-length packets is examined. Given the conclusion of the previous section — that switches with DAMQ buffers maintain their high performance in asynchronous networks by achieving a loose synchrony — one might expect the SAFC switch performance to improve relative to the other buffers in the presence of variable-length packets. The SAFC switch is the only

architecture examined which is not synchronized by forwarding constant-length packets at saturation throughput; all four input ports participate in every output port arbitration, and this is unaffected by the asynchrony caused by variable-length packets.

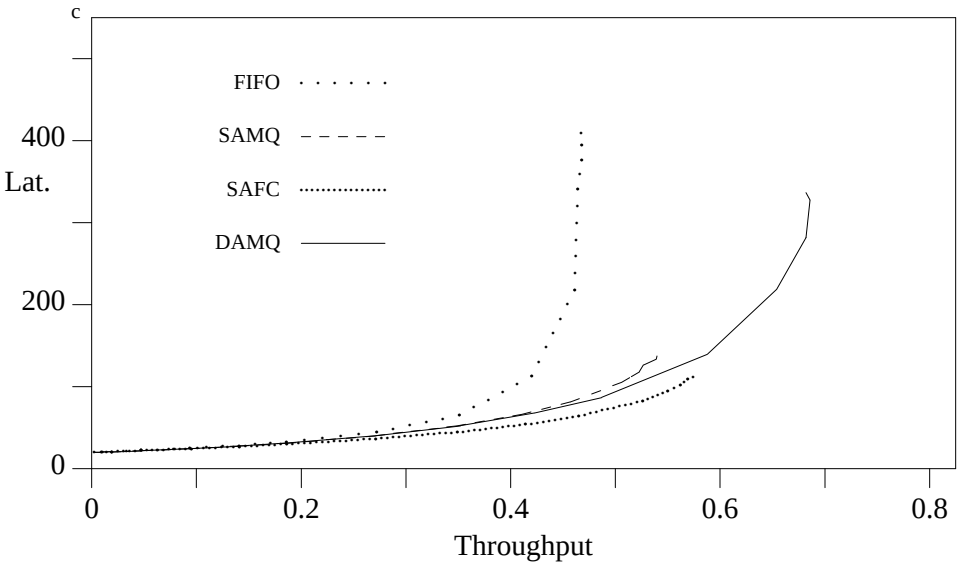


Figure 6.3: *Lat. vs. Thpt., Variable-Length Packets.* 256×256 omega network composed of 4×4 switches. Packets vary in length uniformly from six to thirty-two bytes. Packet destinations randomly chosen, uniform distribution.

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

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

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

6.2.4. Network Throughput with Increased Buffer Memory

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

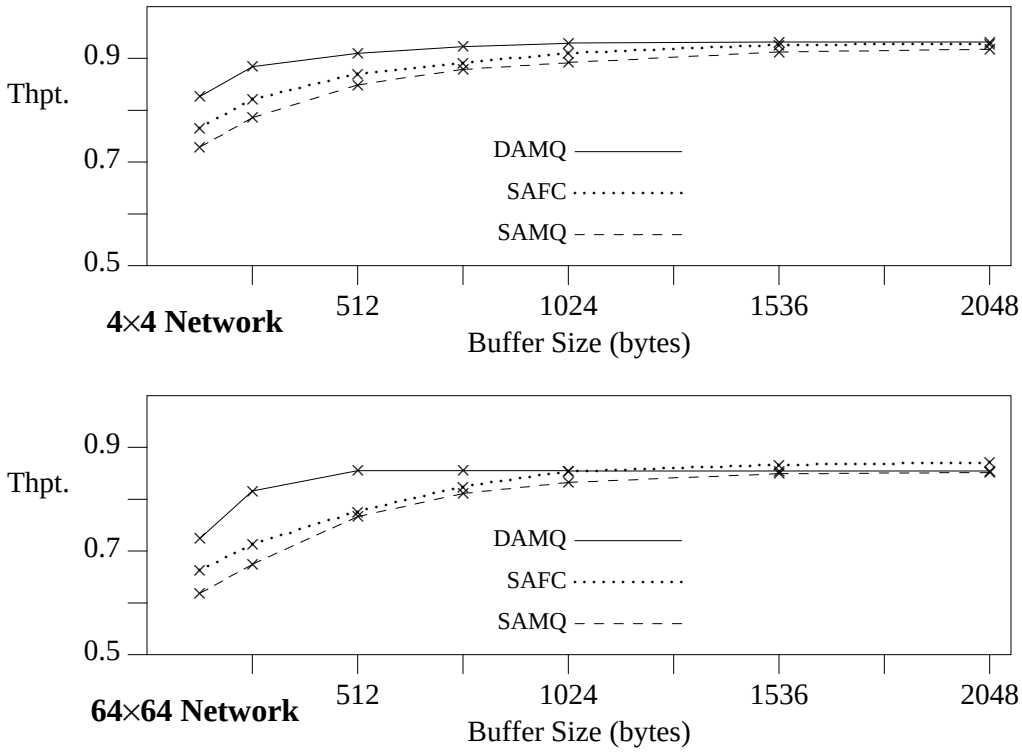


Figure 6.4: *Saturation Throughput vs. Buffer Size, Constant-Length Packets.* Shown is the maximum netork throughput as a function of the amount of buffer memory at each input port for a 4×4 network (a single chip) and a 64×64 network.

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

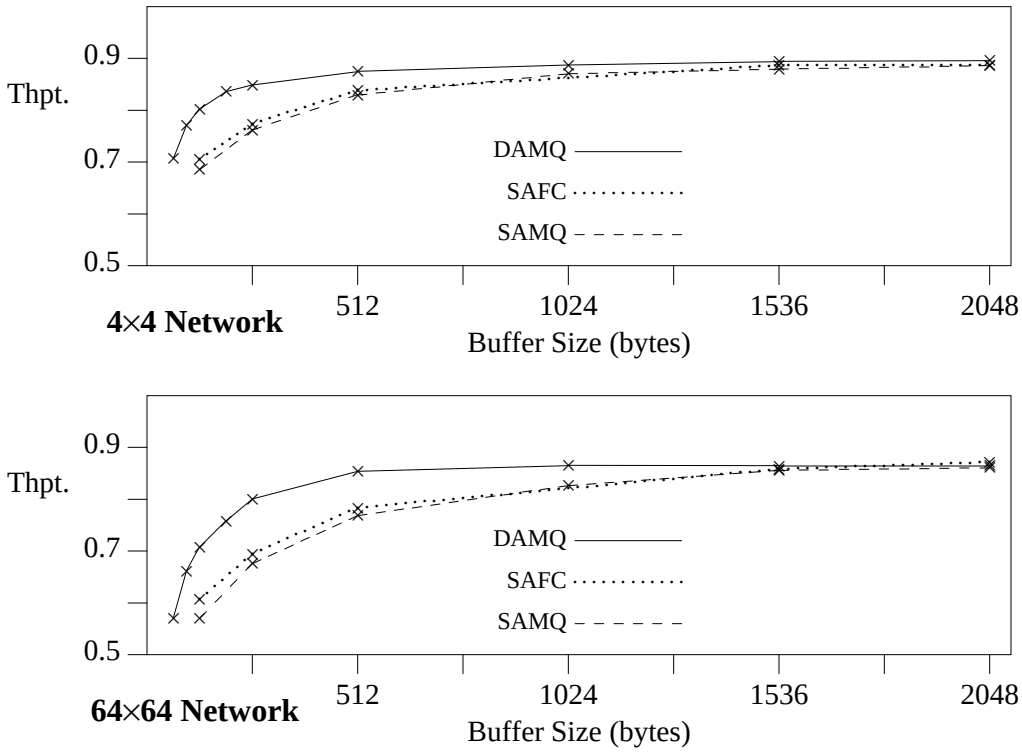


Figure 6.5: *Saturation Throughput vs. Buffer Size, Variable-Length Packets.* Shown is the maximum netork throughput as a function of the amount of buffer memory at each input port for a 4×4 network (a single chip) and a 64×64 network.

6.2.5. Summary: Asynchronous MIN Performance

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

The simulation results presented in this section demonstrate the performance of the DAMQ buffer in an asynchronous network. Multi-queue buffering continues

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

6.3. Network Performance, Torus Topology

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

6.3.1. Dimensional Routing with Multi-Queue Buffers

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

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

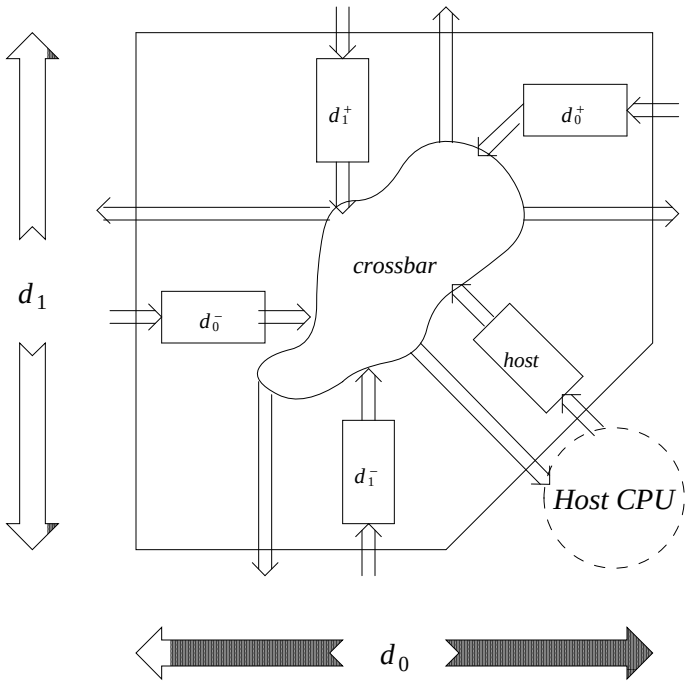


Figure 6.6: *Switch for Two-Dimensional Torus.*

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

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

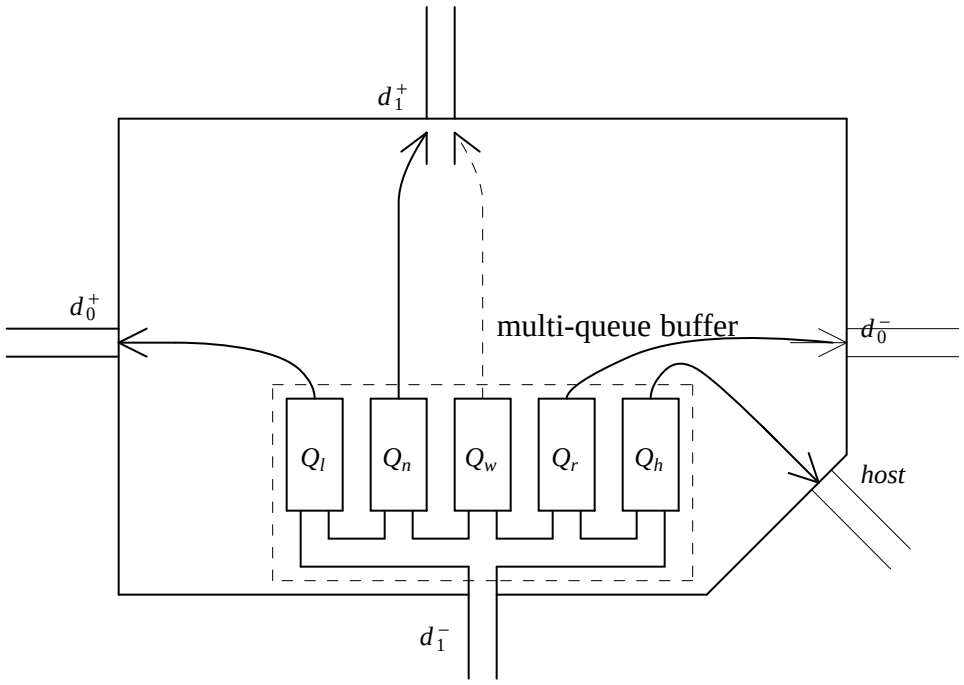


Figure 6.7: *Multi-Queue Buffer for Two-Dimensional Torus.*

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

Given the buffer organization above, deadlock-freedom can be guaranteed if Q_n is not allowed to occupy the entire buffer memory. The packets in Q_h are assured to eventually make progress, as the local node is assumed to be a packet sink. The packets in $Q_{l,r}$ will eventually move forward, since, if the current dimension is guaranteed make progress, then the succeeding dimensions are. Thus,

the only concern is to prevent the packets in $Q_{n,w}$ from creating a cyclical dependency, and this is accomplished by preventing Q_n from occupying the entire buffer memory at an input port.

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

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

6.3.2. Performance Evaluation: Dimensional Routing

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

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

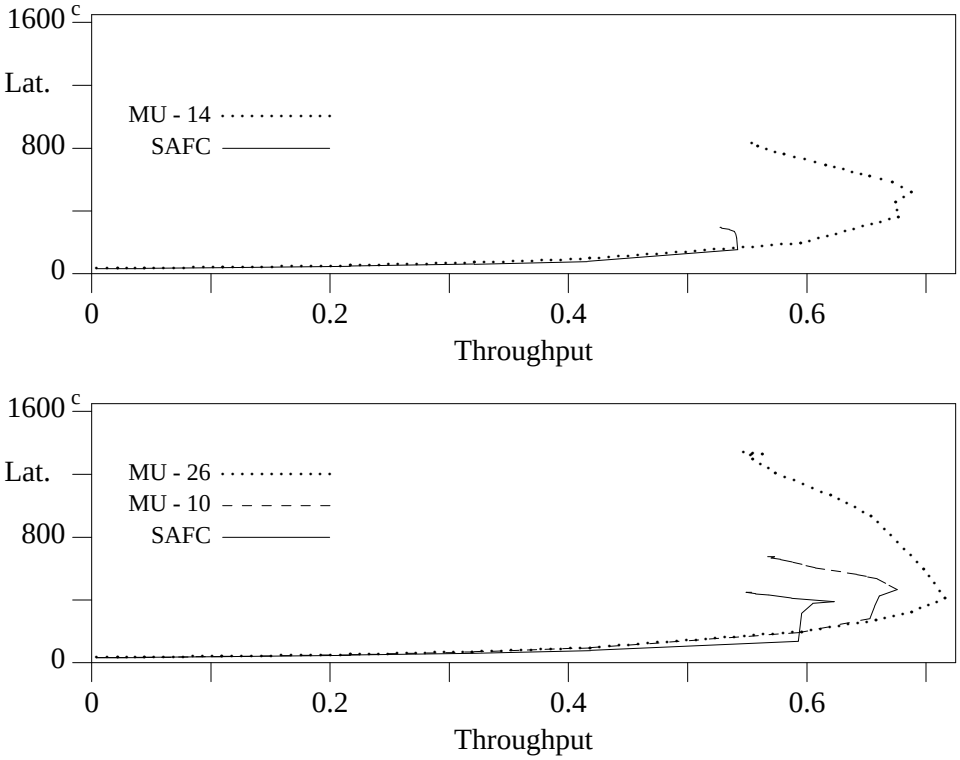


Figure 6.8: *Latency vs. Throughput, 121-Node 2-D Torus.* 11×11 torus network composed of 5×5 switches. Thirty-two byte packets. DAMQ buffers (maximum-usage flow control) and SAFC buffers. Packet destinations randomly chosen, uniform distribution. The top graph is the performance of 160-byte buffers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

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

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

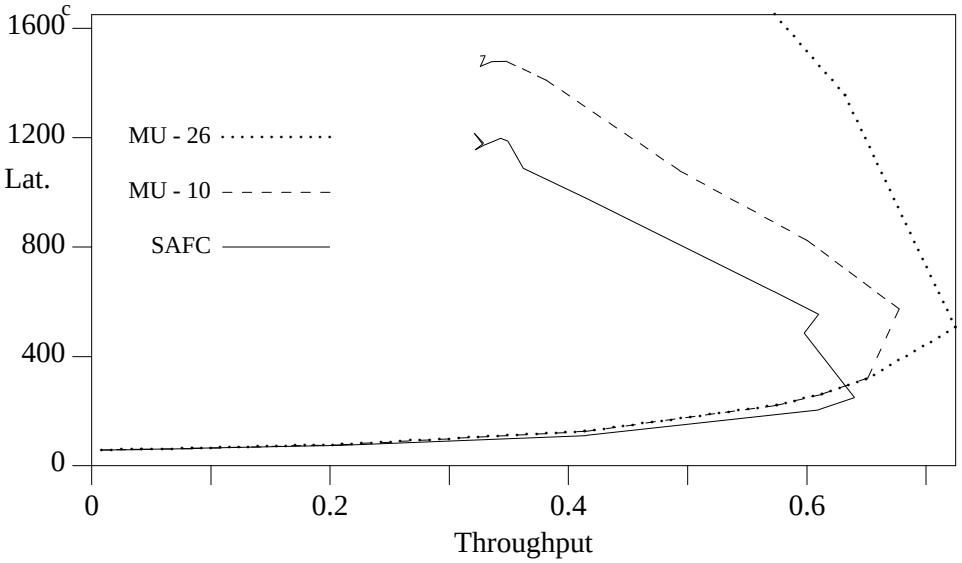


Figure 6.9: *Latency vs. Throughput, 441-Node 2-D Torus.* 21×21 torus network, composed of 5×5 switches. Thirty-two byte packets. Packet destinations randomly chosen, uniform distribution. 256-byte DAMQ buffers with maximum usage flow control (‘‘MU-10’’ and ‘‘MU-26’’ indicate maximum usage flow control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

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

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

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

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

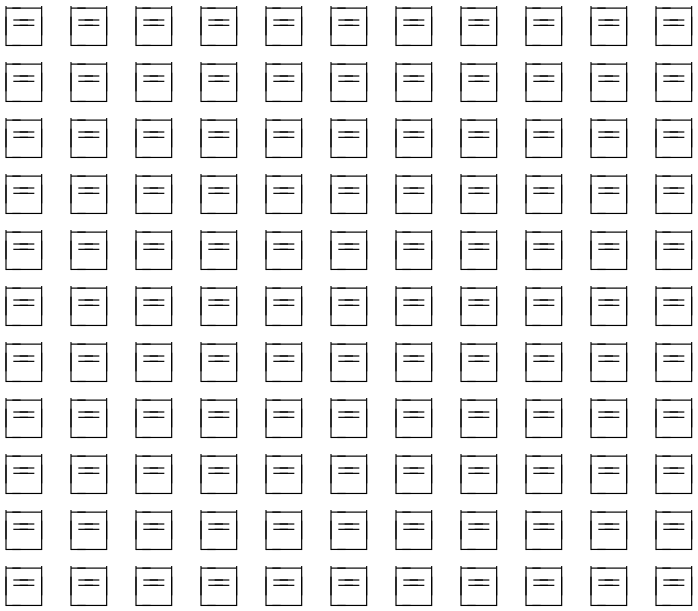


Figure 6.10: *Visualizing Buffer Utilization I.* Torus network operating just below saturation throughput. Differences in the average queue lengths in buffer at the host-interface input port. '=' indicates queues of ≈ equal length, 'p' indicates that Q_1^+ averages at least one block more than Q_1^-,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

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

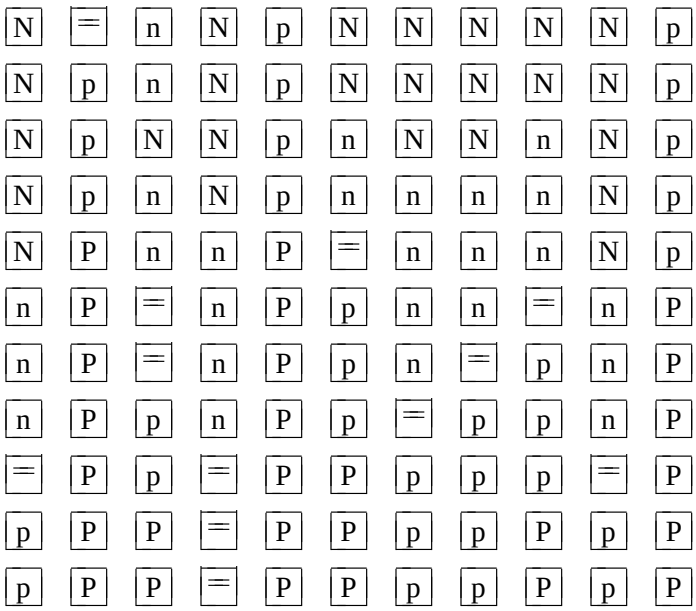


Figure 6.11: *Visualizing Buffer Utilization II.* Torus network operating at saturation throughput.

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

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

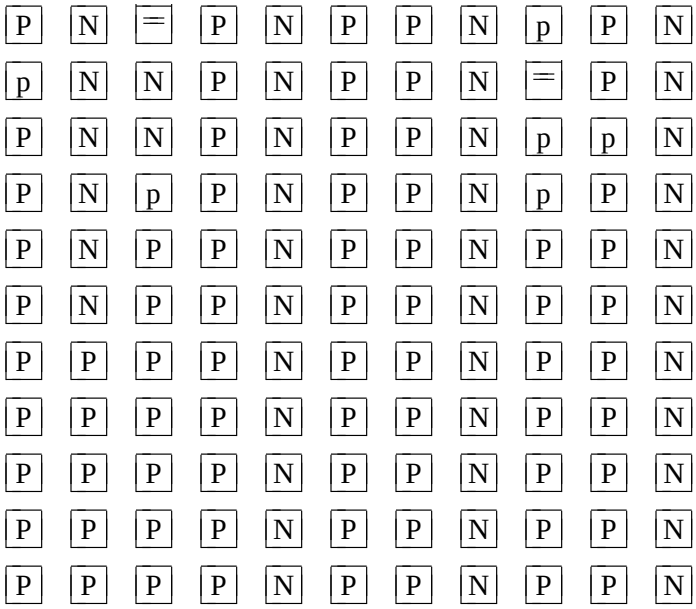


Figure 6.12: *Visualizing Buffer Utilization III.* Torus network operating in the reactive region.

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

In [Dall90b], Dally derives equations which model the performance of k-ary n-cubes which use wormhole routing. The switches in his model are buffer-less; the storage of packets within the network is not modeled. He determined that the maximum throughputs of 1K-node and 4K-node torus networks were 0.36 and 0.35, respectively (throughput measured as the fraction of capacity utilized). This is significantly lower than the throughput results presented in this dissertation for

either the DAMQ or SAFC buffers — Dally acknowledges that buffering can and will improve network throughput. A more significant difference between the results presented in [Dall90b] and here is that Dally reported negligible reactivity, stating that the throughput of the network is proportional to the applied load until saturation, at which point it remains constant or drops slightly. This is in direct contrast to our results, where networks can lose 20%-50% of their available throughput due to reactivity. Dally states that the networks he modeled were non-reactive ‘‘because 1) the network is source queued, and 2) messages that encounter contention are blocked rather than aborted.’’ [Dall90b] We believe that the networks he modeled were only marginally reactive either because of the lack of buffering in the network or because his model did not incorporate back-pressure (since Dally’s model is intrinsically buffer-less, it is impossible to distinguish between the two possibilities).

6.3.3. Summary — Dimensional Routing

We have presented simulation results comparing the performance of torus networks (k-ary 2-cubes) constructed of switches implemented with DAMQ and SAFC buffers. These are store-and-forward buffers capable of virtual cut-through. Deadlock-free dimensional routing is implemented via multi-queue buffers. The DAMQ buffer dynamically allocates its buffer space among its multiple queues, and thus can only read from one queue at a time. The SAFC buffer statically allocates its buffer space. Static buffer allocation has two major impacts on the buffer architecture. First, this facilitates reading multiple packets from a single input port buffer simultaneously (see Sec. 3.3), and, second, it reduces the

efficiency with which the memory is utilized. Our simulation results indicate that the superior buffer utilization provided by the DAMQ buffer has a greater impact upon network throughput than does the multiple simultaneous reads supported by the SAFC buffer, for the sizes of buffer memory examined.

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

6.4. Summary

This chapter presented an asynchronous communication network simulator. This simulator provides more functionality than does the simulator presented in Ch. 4. First, the switches being simulated can be connected in topologies other than an omega network; torus, mesh and hypercube topologies are currently

implemented. Second, since asynchronous communication allows variable-length packets, parameters were added to the simulation to allow message length distributions and a maximum packet length to be specified. The third significant feature of this simulator not present in the synchronous network simulator is the ability to segregate the *senders* (packet sources) into groups, giving each group its own packet creation and destination assignment distributions and keeping separate performance statistics for each group.

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

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

The performance of statically allocated buffers decreases in the presence of

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

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

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

saturation throughput than do SAFC buffers — in fact, 256-byte DAMQ buffers outperform 320-byte SAFC buffers in the networks we simulated.

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

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

Chapter Seven
The DAMQ Buffer Chip

The previous three chapters of this dissertation evaluated the DAMQ buffer architecture in the context of a total communication network. This evaluation involved simulation studies that compared the performance of networks whose configurations are identical except for the type of buffers in the switches. The validity of these simulations relies upon the accuracy of two assumptions. The first is that the DAMQ buffer is capable of transmitting / receiving data at as high a rate as a FIFO buffer. The second assumption is that it is capable of forwarding packets with as low a latency (from when they were received) as a FIFO buffer. This chapter examines an implementation of the DAMQ buffer with two goals in mind. The first is to demonstrate the feasibility of the DAMQ buffer architecture. The second goal is to show that the DAMQ buffer's complexity does not reduce its raw performance relative to the much simpler FIFO buffer architecture.

While the DAMQ buffer was designed to be a functional unit within a single-chip communication switch, this chapter describes a ''stand-alone'' buffer, the *DAMQ Buffer Chip*. Constructing a complete communication switch will require combining multiple DAMQ Buffer Chips together with several other key building blocks, such as a router and arbiter.

When considering a single packet buffer (as opposed to a switching network), there are two key measures of performance. The first is the raw bandwidth of the buffer — the amount of data which can be transmitted and / or received per unit time. The second is the minimum cut-through latency of the buffer — the interval

between the time a packet begins to arrive at the input port of an empty, idle buffer and the time said packet begins to depart. This chapter demonstrates that the DAMQ Buffer Chip is the equal of a FIFO buffer in both of these measures.

The key to showing that the DAMQ buffer can sustain as high a raw bandwidth as a FIFO buffer is the fact that the layout of the physical memory used to store the packets is nearly identical for the two buffer architectures. If we assume that the FIFO buffer's control logic is infinitely fast, then it will be its buffer memory which determines the rate at which the FIFO buffer communicates. Thus, asserting that the DAMQ Buffer Chip has as high a raw bandwidth as a comparable FIFO buffer is a two-step operation. First, we design the DAMQ Buffer Chip with a buffer memory whose latency is as low as we are capable of making it. Then, we implement the control logic such that it is the buffer memory which determines the DAMQ Buffer Chip's clock rate.

We take a different tack to demonstrate that the DAMQ Buffer Chip achieves as low a cut-through latency as a FIFO buffer. The approach here is to find a chain of serial dependencies that exist with both FIFO buffers and DAMQ buffers and which combine to determine a theoretical minimum cross-switch latency. If the DAMQ Buffer Chip can be implemented so that it achieves this minimum latency, the implication would be that the cut-through latency with a DAMQ buffer can be as short or shorter than the cut-through latency with a FIFO buffer.

The next section describes the microarchitecture of the DAMQ Buffer Chip. The hardware structures which organize the packets into multiple queues are explained. A minimum virtual cut-through latency of four clock cycles is established in this section. Having set this as a performance goal, the hardware

mechanisms capable of meeting this goal are presented. Sec. 7.2 discusses the layout and circuit performance of the individual functional units. Results of SPICE circuit simulations are presented. These results demonstrate that the DAMQ Buffer Chip achieves a raw bandwidth equal to that of a FIFO buffer. Finally, Sec. 7.3 summarizes the contributions of this chapter.

7.1. The Architecture and Microarchitecture of the DAMQ Buffer Chip

The external interfaces and the context in which the DAMQ Buffer Chip is intended to operate will be discussed, and then the internal microarchitecture described. The DAMQ Buffer Chip is a packet buffer with a single byte-wide input port and a single byte-wide output port. It is designed to receive packets from and transmit packets to other DAMQ Buffer Chips. A communication switch can be constructed by connecting the output ports of a number of DAMQ Buffer Chips to the inputs of a crossbar — the input ports of the buffers become the input ports of the switch, and the outputs of the crossbar are the switch's output ports.

The packet format used by the DAMQ Buffer Chip is a single *header byte* followed by one to thirty-two bytes of data. The header byte contains the routing information for the packet plus a bit which indicates whether the packet is thirty-two bytes in length or less. If it is less than thirty-two bytes, then the byte following the header indicates the length of the packet. Packets arriving at an input port have their data stored in the memory array of the buffer at that input port while the header is sent to the router. The router determines the output port to which the packet should be forwarded. With some routing schemes, such as virtual circuits [Bert87], the router also returns a new header byte to be prepended to the

packet for its next ‘‘hop’’ through the network. In the DAMQ Buffer Chip, the router returns a four-bit value specifying which of the four queues to append the packet to, an eight-bit header to be associated with the packet, and a *valid queue* signal which causes the routing information to be latched and processed.

In order to forward a packet, a connection must be established between the DAMQ Buffer Chip and the appropriate switch output port. The *crossbar arbiter* establishes connections based upon the state of all the input buffers and the switch output ports. Each buffer has four request lines which indicate to the arbiter which of the buffer's packet queues are not empty. The arbiter responds to each buffer with a four-bit value specifying which of the four queues is to transmit (i.e. to which output port the buffer has been connected) and a *valid output port* signal which causes the arbitration information to be latched and processed by the buffer. Upon receiving a valid output port signal from the arbiter, the DAMQ Buffer Chip commences transmission of the packet at the head of the queue specified by the arbiter. The packet is transmitted at a rate of one byte per clock cycle, with the first byte being the new header which was supplied by the router.

Since the intended application for the DAMQ Buffer Chip is in the communication switch of a tightly-coupled multicomputer, it is assumed that all switches operate at the same frequency. We avoid the difficulties of distributing a global clock signal by allowing each switch to generate its own clock via a local oscillator. *Synchronizers* are placed between the input port of the switch and the DAMQ Buffer Chip's input port. These determine for each packet whether to latch incoming data on the rising or falling edge of the local clock [Tami88a]. The probability of synchronization failure due to clock drift over the course of a

reception is reduced to insignificance by restricting the length of packets allowed in the system. For this implementation, the maximum packet size is thirty-two bytes [Tami88a].

Having described the external architecture and context of the DAMQ Buffer Chip, the rest of this section will describe the design and implementation of its internal features. Sec. 7.1.1 describes the basic organization of the DAMQ Buffer Chip. Sec. 7.1.2 addresses the specific hardware units which implement this organization. Minimizing the cut-through latency of the buffer is the driving force behind the microarchitecture of the DAMQ Buffer Chip — Sec. 7.1.3 establishes a ‘‘minimum’’ virtual cut-through latency of four clock cycles and describes in detail the hardware features which support this. Finally, Sec. 7.1.4 summarizes the DAMQ Buffer Chip microarchitecture.

7.1.1. The Basic Organization of the DAMQ Buffer Chip

The DAMQ buffer is a single storage array in which memory is dynamically allocated among multiple queues of packets, one queue for each output port. Such dynamic partitioning of the buffer between the queues is significantly more efficient than static partitioning, in terms of both implementation complexity (Ch. 3) and buffer utilization (Chs. 4 and 6). While the order in which queues are serviced is ‘‘random’’ (determined by the arbiter), it is desirable to transmit the packets within each queue in FIFO order (some transmission protocols such as virtual circuits require this).

With dynamic buffer allocation, a packet’s location within the buffer cannot be determined by the location of the packet immediately ahead of it in its queue or

the location of the packet whose arrival preceded it. The DAMQ Chip thus requires a mechanism to rapidly locate, access, and transmit the first packet in any one of the queues. Further, in order to handle variable-length packets efficiently, the DAMQ buffer must be able to quickly allocate memory for incoming packets without wasting buffer space due to internal or external fragmentation.

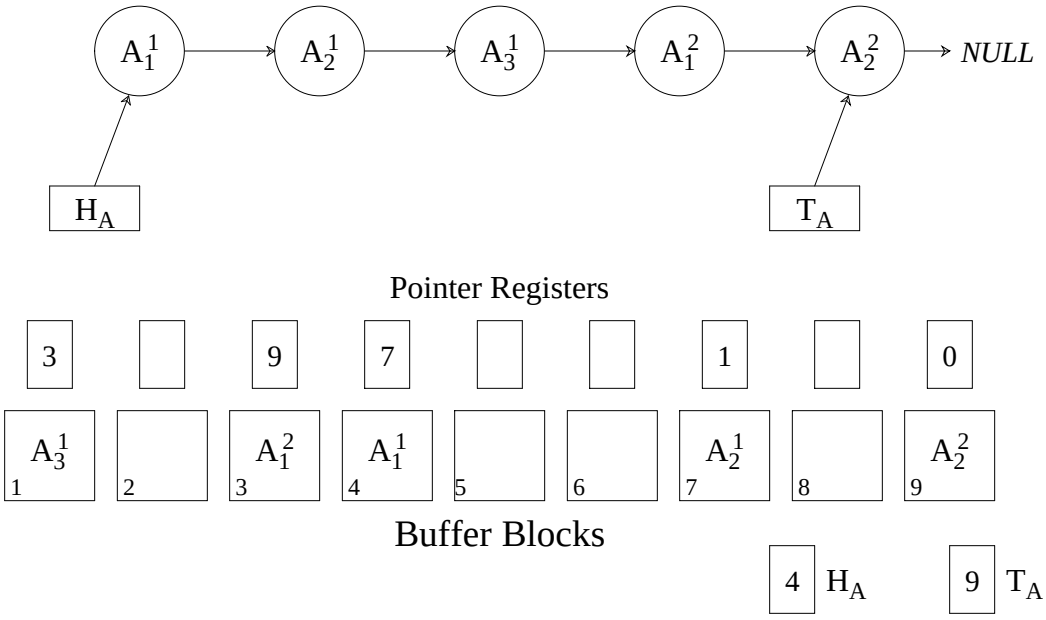


Figure 7.1: *Linked Lists via Buffer Blocks.* Pointer registers link the blocks which comprise a packet (A_1, A_2, etc.) and the packets which comprise a queue (A^1 and A^2 of queue A).

The problem of maintaining the queue organization is resolved via linked lists. The memory array of the DAMQ buffer is partitioned into fixed-size *buffer blocks*. Each queue is a linked list of these blocks (Fig. 7.1). *Head registers* and *tail registers*, which point to the first and last block of each queue, are located outside the array of buffer blocks, allowing access to any one of the queues. To reduce internal fragmentation due to variable size packets, the buffer blocks (eight

bytes) are significantly smaller than the maximum packet size (thirty-two bytes in this implementation). Each packet is stored in one or more buffer blocks. For each queue, the linked list of buffer blocks defines the FIFO order between packets as well as the order of blocks that make up each packet. A linked list of ‘‘free buffer blocks’’ (the *free list*) is used to keep track of storage available for incoming packets. For each buffer block there is a *pointer register*, stored outside the packet memory array, which contains the address of the next buffer block of its list or a *NULL* value signifying that it is the last block in the queue.

7.1.2. The DAMQ Buffer Chip Control Logic

The control logic of the DAMQ buffer is significantly more complex than that of a FIFO buffer. The process of receiving a packet involves: (1) locating the block at the head of the free list and directing incoming data to that block, and (2) altering the linked list pointers, logically moving the block to the tail of the appropriate queue. When a packet is transmitted, the block which is at the head of the current queue is moved to the tail of the free list while its data is being read. Whether receiving or transmitting, the primary function of the control logic of the DAMQ buffer is to move a buffer block from the head of one queue to the tail of another.

The control logic of the DAMQ buffer must be able to manipulate the linked list pointers, moving buffer blocks from one linked list to another, in as few clock cycles as possible. The control must also satisfy three additional requirements: (1) minimize the silicon area required for the buffer control hardware, (2) support the simultaneous reception and transmission of packets (both to maximize link

utilization and to support virtual cut-through), and (3) ensure that the control logic is not on the DAMQ buffer's critical path. The last requirement is key to allowing the DAMQ buffer to realize its performance advantage [Tami88b, Tami88c] With its simple control logic, other factors (e.g. reading from the memory array or the raw bandwidth of the links) will determine the clock rate of a FIFO buffer.

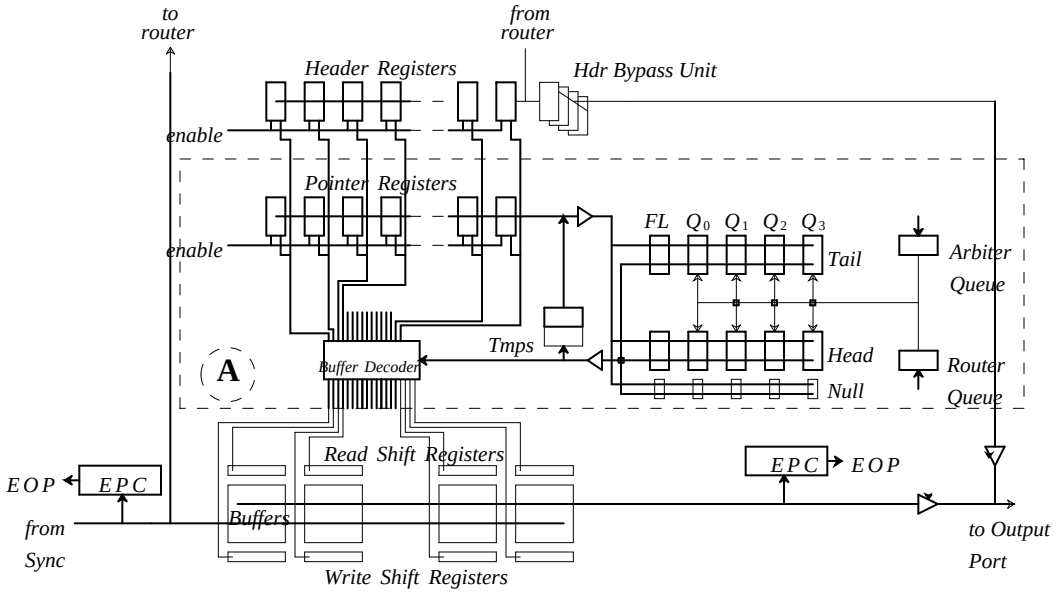


Figure 7.2: *DAMQ Chip Control Logic.* The circuitry in the rectangle marked 'A' implements the five linked lists.

The data path and key control circuitry of the DAMQ buffer are shown in Fig. 7.2. Logically moving a block from one queue to another requires three clock cycles:

cycle 1: The value in the head register of the queue the block is being removed from is copied to one of the two temporary registers. This value is also sent to the decoder, where it is used to select the pointer register containing the address (block number) of the next block in the queue. This pointer register is read and the address is written into the head register, thus removing the block from the queue.

cycle 2: The tail register of the queue to which the block is being appended is read; the value is sent to the decoder to select the pointer register of the block currently at the tail of the queue (if the the queue is empty, the the null register will prevent any pointer register from being addressed). The value in the temporary register, which is the address of the block being ‘‘moved,’’ is copied to the addressed pointer register. The value in the temporary register is also copied to the tail register of the destination queue. The block is now pointed to both by the pointer register of the block ahead of it in the new queue and the tail register of the new queue.

cycle 3: The tail register is read (to address the pointer register of the buffer block), and the *NULL* value is stored in the pointer register, marking this the last block of the queue.

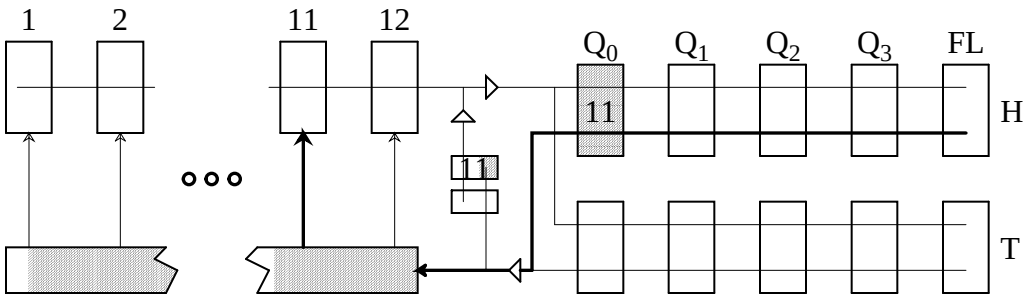


Figure 7.3: *Addressing Phase.*

The operations in each of the three clock cycles are divided into two phases: *addressing* and *data movement*. In the addressing phase, a head or a tail register is read, and its value is driven through the decoder to address a pointer register. This value may also be stored in a temporary register. In the data movement phase, data is either copied from a pointer register to a head / tail register, or data is copied from a temporary register to both a pointer and a head / tail register. Figs. 7.3 and

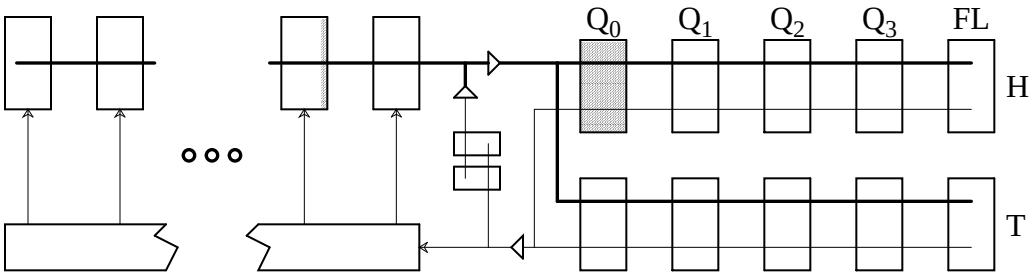


Figure 7.4: *Data Movement Phase.*

7.4 show the actions in each phase of the functional units which implement the linked lists.

Since the buffer must be able to simultaneously receive and transmit packets, the buses and registers are multiplexed. The transmission reception state machines control the data path on alternate clock cycles. Hence, a minimum of five clock cycles are needed to complete the three linked list manipulations involved in receiving or transmitting a packet. However, when a packet destined for an empty queue arrives, there is no need to wait for all of these operations to complete before beginning to forward it (Sec. 7.1.3).

The data path and buffer memory are controlled by three finite state machines (FSMs). FSM_{sto} stores incoming packets into buffer blocks. FSM_{rtr} removes blocks from the free list and appends them to the queue specified by the router. It uses the data path on the same cycles that FSM_{sto} does. FSM_{fwd} forwards the packets. This division of control allows concurrent packet reception, routing and transmission. The queue to which FSM_{rtr} appends packets is specified by the *router queue* register (Fig. 7.2). Similarly, the queue from which FSM_{fwd} transmits packets is specified by the *arbitration queue* register — the finite state machines only distinguish between ‘‘the queue’’ and the free list when specifying head / tail

registers for reading or writing.

7.1.3. Minimizing Virtual Cut-Through Latency

Supporting virtual cut-through is key to maintaining low latency communication. With virtual cut-through [Kerm79] the switch can begin to forward a packet before it has completely arrived. A packet can only be cut through a switch if the buffer at which it is arriving is not already transmitting a packet, the output port for which the packet is destined is not connected to another buffer, and there are no packets already in the buffer waiting to be transmitted through that same output port. Optimally, when these conditions exist, the packet will be forwarded as soon as it is routed and the buffer is connected to the appropriate output port. One of the concerns in designed a complex packet buffer such as the DAMQ Buffer Chip is that the buffer will increase the virtual cut-through latency. This section establishes a ‘‘minimum’’ virtual cut-through latency, and then enumerates the hardware requirements for the DAMQ Buffer Chip to achieve this minimum.

Thus, there are two keys to low latency virtual cut-through. First, the routing of the packet and arbitration of the switch must be performed as quickly as possible and in parallel with packet reception. Second, the packet must be available for transmission as soon as the arbitration is complete, no matter how much of the packet has been received at that point. The first condition is satisfied by keeping the routing information at the head of each packet, and by keeping the router and arbiter units independent of the buffer itself. These two measures allow the routing to commence the cycle the packet begins to arrive, and the arbitration to commence

as soon as the routing is complete. The second condition is partially met by having separate FSMs for reception and transmission, but some special-purpose hardware is necessary to ensure that the arriving packet is available for transmission when the arbitration is complete.

Cycle	Action	Implication
1	Header is synchronized	
2	Header $\rightarrow$ Router, Router returns valid queue + new header	completion of routing triggers FSM_{rtr}
3	Arbiter receives new connection request, switches the X-bar, returns valid output port. FSM_{rtr} appends the packet to the [empty] queue.	completing of arbitration triggers FSM_{fwd}
4	Packet header traverses the crossbar	FSM_{fwd} must cause header to be read at beginning of this cycle and access buffer memory to send first data byte on the following cycle.
5	First byte of packet traverses the crossbar	To transmit byte at beginning of this cycle, the buffer memory was accessed the previous cycle.

Table 7.1: *Requirements Imposed by Minimizing Cut-Through Latency.* Assuming that routing cannot occur until the packet header is received, arbitration cannot occur until the packet is routed, and transmission cannot begin until the crossbar is arbitrated.

Table 7.1 displays the sequence of events imposed upon the DAMQ buffer by the goal of minimizing virtual cut-through latency and our assumptions with respect to the implementation. We assume that the packet cannot be routed until the packet's header passes through the synchronizer, that the crossbar cannot be arbitrated until the router determines to which output port the packet is destined, that transmission cannot commence until the crossbar arbitration is complete, and,

finally, that routing and arbitration will each take at least one clock cycle to perform. Thus, our goal is to support a four-cycle virtual cut-through.

To support a four-cycle virtual cut-through latency, FSM_{rtr} *must* have control of the data path on the cycle following the completion of the routing. Similarly, FSM_{fwd} must control the data path on the cycle following a crossbar arbitration. However, a packet may arrive on any given clock cycle. Thus, the data path must be demand multiplexed between the reception and transmission control logic. Further, the validqueue signal from the router may arrive as late as the end of a clock cycle, but FSM_{rtr} must react to the signal and take control of the data path at the beginning of the next cycle. Sec. 7.2.3 details how the finite state machines support virtual cut-through and demand multiplexing.

As can also be seen from Tab. 7.1, FSM_{fwd} must be able to access both the new packet header and the first byte of data on the clock cycle following the arbitration in order to provide the minimum virtual cut-through latency (the buffer memory read requires a full clock cycle). This presents two problems for the hardware: how to support single-cycle access to a packet header, and how to set the head-of-queue hardware (which previously held nonsense, for an empty queue) to point to the new packet. The remainder of this section details our solution to these problems.

Packet headers are stored in an array of *header registers*. Each header register is associated with a buffer block. Under normal conditions, the DAMQ buffer control addresses the header registers via the same mechanism it addresses the buffer blocks. This addressing mechanism involves reading the block identifier from the head register of the selected queue during the first phase of the clock

cycle. The block identifier is sent to the decoder, which asserts the select lines for the header register array, pointer register array and the buffer blocks themselves. However, if this indirect addressing mechanism is used to access the header byte of the packet to be transmitted, the header will not traverse the crossbar until at least two clock cycles after arbitration. What is needed is a mechanism to provide direct access to the header of the packet to be transmitted.

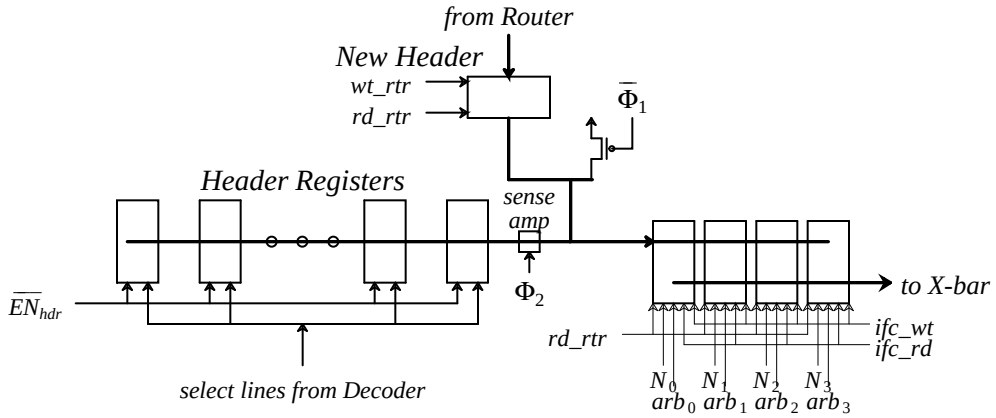


Figure 7.5: *The Header Register Array.* Header bypass registers can be explicitly written ($ifc\_wt \cdot arb_x$) or implicitly written ($rd\_rtr \cdot N_x$) (arb is the four bits from the arbiter which select the queue to be transmitted, N is the values of the four null registers associated with the packet queues).

The *header bypass registers* in the header register array (Fig. 7.5) provide direct access to the header byte of the packet at the front of each of the packet queues. There are four bypass registers — each holds the header of the packet at the head of a queue. When the arbiter returns a *valid output port* signal, FSM_{fwd} transmits the header which is in the bypass register associated with that queue / output port. FSM_{fwd} need only assert *ifc_rd* which, in combination with arb_x (the signals which indicate which queue was selected by the arbiter), cause the header byte to be driven over the crossbar. At the end of each packet

transmission, FSM_{fwd} accesses the header register array and loads the bypass register with the header of the new front-of-queue packet.

Virtual cut-through occurs when a queue is empty, however. In the case described in Tab. 7.1, FSM_{fwd} will not have had time to copy the packet's header from the header register to the bypass register before that header is to be transmitted over the crossbar. Since it is FSM_{rtr} which controls moving the packet header from the *New Header* register (Fig. 7.5) to the packet's header register, FSM_{rtr} writes the packet's header into the bypass register when the packet is appended to an empty queue. This is accomplished by writing the header values to the bypass register of every empty queue whenever the *New Header register* is read. Thus, if the router returns a new header at the end of cycle 2 (referring to Tab. 7.1), FSM_{rtr} controls the data path on cycle 3 and reads the New Header register on that cycle, causing the header to be written into the bypass registers of all empty queues.

A similar situation exists with respect to the *head register* of the queue to / from which the cut-through packet is being written / read. As was previously mentioned, the internal organization of the DAMQ buffer is based upon linked lists. Each queue is referenced by two registers; the *head register* points to the first block in the queue, and the *tail register* points to the last. FSM_{rtr} appends a block to a queue by manipulating the tail register of the queue (T_Q). FSM_{fwd} locates the first block of a queue by accessing the head register (H_Q). When the queue is empty, the head and tail registers hold nonsense values.

In clock cycle 4 of Tab. 7.1, FSM_{fwd} reads the head register of the queue, and uses this value to access the buffer block. This implies that, on cycle 3, FSM_{rtr}

must write the identifier of the block which is at the head of the free list to the tail register of the appropriate queue *and* that this value also be written to the the head register. The first issue is making the write to T_Q be the first function performed by FSM_{rtr}. Reading H_{fl} (the free list's head register) can be performed ahead of time because (a) it is known that any incoming packet will be stored in that buffer block and (b) the lines connecting the decoder to the addressing shift registers of the buffer blocks have latches that allow a buffer block to be selected an indeterminate number of clock cycles ahead of when reception will begin. So, the value in the head register of the free list can be stored in the temporary register ahead of time, and FSM_{rtr}'s action during cycle 3 (Tab. 7.1) can be $[read(T_Q); ptr(T_Q) \leftarrow tmp; T_Q \leftarrow tmp]$ — read and decode the tail register of the queue specified by the router; store the value from the temporary register (which is the identifier of the block at the head of the free list) into the pointer register associated with the last block of the queue; store the value from the temporary register into the tail register of the queue.

In the case of an empty queue (potential for virtual cut-through), the *null register* of the queue (N_Q) will prevent any pointer register from being selected when T_Q is read. It will also cause any value written to T_Q to also be written to H_Q. Thus, the state machine does not differentiate between appending a packet to empty vs. non-empty queues; the null register automatically sets the head register of an empty queue to point to the first block appended to the queue.

7.1.4. Summary — DAMQ Buffer Architecture

The DAMQ buffer design is driven by the goal of minimizing the latency of virtual cut-through. A register latches the packet's header as soon as it passes through the synchronizer, and presents it to the router along with an enable signal, allowing the routing to take place immediately. The same separation control and addressing hardware which allows the DAMQ buffer to simultaneously receive and transmit separate packets also allows it to cut a single packet through. Demand multiplexing allows the finite state machines controlling packet reception and transmission to access the data path on the clock cycles immediately following routing or arbitration. A copy of the header for the packet at the head of each queue is stored in a *bypass register*, allowing FSM_{fwd} to begin a transmission without the latency of decoding a block identifier. Finally, *null registers* associated with each packet queue cause dual tail / head register writes to empty queues. With an on-chip router utilizing simple algorithmic routing or small a routing table, and an on-chip arbiter which operates in a single clock cycle, the minimum latency across a single DAMQ switch is four clock cycles. Each additional cycle required to route a packet or arbitrate the switch adds a cycle to this latency.

7.2. Floorplan and Circuit Performance

In Ch. 4, it was shown that, given equal clock rates, a network made up of DAMQ buffers outperforms a corresponding FIFO network. The microarchitectures of the DAMQ and FIFO buffers must be compared, however, to determine whether and to what degree the additional complexity of the DAMQ

buffer affects its performance. This complexity could cause the DAMQ buffer to operate at a slower clock rate than the FIFO buffer. In addition, the control logic could occupy silicon area which the FIFO buffer would use for additional storage. Additional storage for incoming data increases the FIFO buffer's performance.

In order to compare the performance of the DAMQ buffer to that of the FIFO, the DAMQ Buffer Chip has been designed and layed out using MOSIS scalable (2 μ) CMOS design rules. While the structure of the DAMQ buffer is significantly different from that of a FIFO buffer, the two architectures share a common feature — the buffer memory array. While the DAMQ buffer imposes a logical structure upon the memory array (the buffer blocks), physically, it is identical to the memory array that a FIFO buffer would use. Since the FIFO buffer's control logic is so simple, there is a strong probability that the latency of buffer reads would determine the cycle time of a FIFO buffer (i.e. the memory array accesses would be the critical path).

Thus, in order to assert that the DAMQ buffer can operate at as high a bandwidth as a FIFO buffer, we implemented a high-performance static memory array as the DAMQ Buffer Chip's buffer memory and established that the DAMQ buffer control logic is fast enough to operate within the clock cycle determined by the buffer memory's latency. The next section describes the implementation and SPICE simulation of the buffer memory. Sec. 7.2.2 describes the operations that the control logic must complete in a single clock cycle and demonstrates via SPICE that the buffer memory remains the performance bottleneck. Sec. 7.2.3 presents the circuitry of the finite state machines and a discussion of their operation within the two-phase clock defined by the data path implementation. We conclude the

performance analysis by examining the silicon requirements of the DAMQ and FIFO buffer control logic and summarizing the performance and implementation issues addressed by these sections.

7.2.1. The Memory Array

The integrity of this performance analysis depends upon the implementation of the memory array. If the memory array is slow, then the comparison of the speed of the DAMQ buffer vs. the FIFO will not be valid. If the memory array consumes more silicon area than is required, then our evaluation of the silicon requirement of the DAMQ buffer control logic will be optimistic. These goals were pursued without sacrificing the fundamental goal of a multicomputer switch implementation — minimizing communication latency.

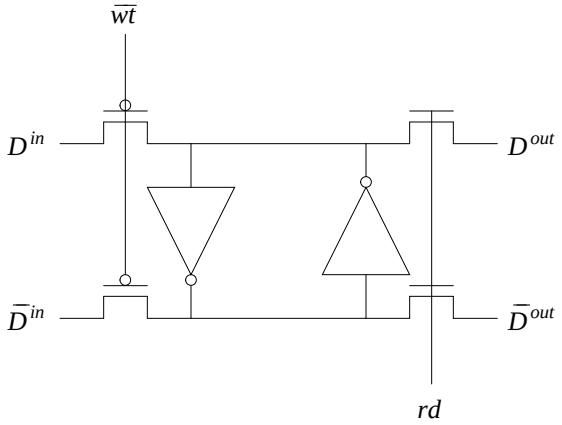


Figure 7.6: *Memory Cell.* Two-ported static memory cell used to construct the buffer memory. Read and write control are independent. Both the read and the write buses are dual-rail.

The memory array is eight bits wide — the same width as the communication links. The array has the capability to read and write a byte of data each clock cycle. It is composed of standard eight-transistor dual-ported static memory cells

(Fig. 7.6). The memory is designed to operate with a two-phase clock. During the first phase, the read bus is precharged and the value to be written is asserted onto the write bus. On the rising edge of the second-phase clock (ϕ_2), read and write select lines (rd and $\overline{wt}$) are asserted on the words of memory to be read and written. A sense amplifier on the read bus accelerates the buffer reads.

In order to increase the speed with which the memory array is accessed, we divided it into two separate arrays (Fig. 7.18), halving the capacitance on the buses. We also took advantage of the sequential nature of the accesses to the memory array by replacing random access address decoding with shift-register addressing. The memory array is divided into eight-byte buffer blocks. Each block has, running along its sides, two eight-bit shift registers: one for reading the block, one for writing. To write to a block of memory, an ''on-bit'' is injected into the write shift register by addressing that block ($\overline{SEL}_{wt}$) while signaling that a write is to begin (EN_{wt}). The bit is shifted during ϕ_1, and the write enable line ($\overline{wt}$) associated with the on-bit is driven low during ϕ_2. When the on-bit reaches the end of the shift register, a signal is generated (EOB_{wt} — end-of-block) indicating that the write to this block is finished.

Figure 7.7 presents the results of a SPICE simulation† of a read of the memory array. At 0 ns, ϕ_2 begins to fall and the precharge of D^{out} and $\overline{D}^{out}$ begins. At $\approx$ 10 ns, D^{out} and $\overline{D}^{out}$ are equal and $\approx$ 5.0 v. ϕ_2 rises at 11.0 ns, which causes rd (the read enable signal) to rise. Since rd is in polysilicon (as is $\overline{wt}$), we measured rd^7, which is the voltage of rd as seen by the memory cell on the

† All of the SPICE simulation results presented in this chapter were generated using slow / slow technology files for 2 mu implementations supplied by MOSIS.

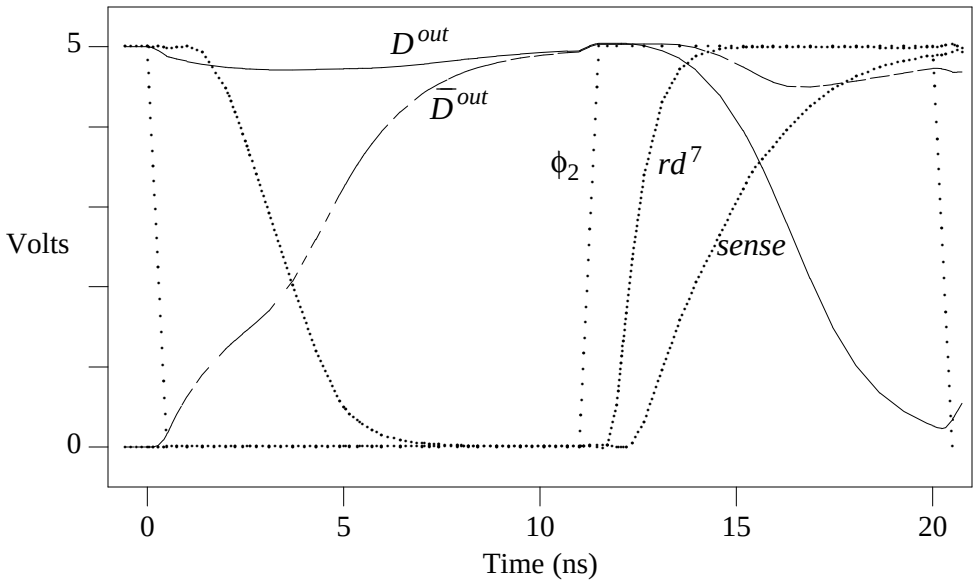


Figure 7.7: *Memory Array Read Latency.* The bus lines (D^{out} and $\overline{D}^{out}$) must be precharged before the rising edge of ϕ_2. The value on the bus is latched on the falling edge of ϕ_2.

opposite side of the memory array from the shift register. The enable signal for the sense amplifier (*sense*) rises soon after rd^7. D^{out} drops below 1.0v at 18 ns — at 20 ns, the value on D^{out} is latched and the precharge for the next read begins. This implementation of the memory array thus defines a 50 MHz two-phase clock in which ϕ_2 has a 45% duty cycle. For comparison, IBM's Vulcan Switch [Stun94], the communication switch of the SP1 multicomputer, is designed to operate at 50 MHz, and also communicates via byte-wide links [Stun94]. Thus, despite using a 2 µ technology, we have successfully implemented a buffer memory which will support the communication throughput demanded by current multicomputers.

7.2.2. The Buffer Control Data Path

The previous subsection established a target clock cycle for the DAMQ Buffer Chip (50 MHz). The goal for the data path implementation is to operate within this clock — i.e. to not be on the critical path. In this subsection, the performance of the functional units which comprise the data path is measured.

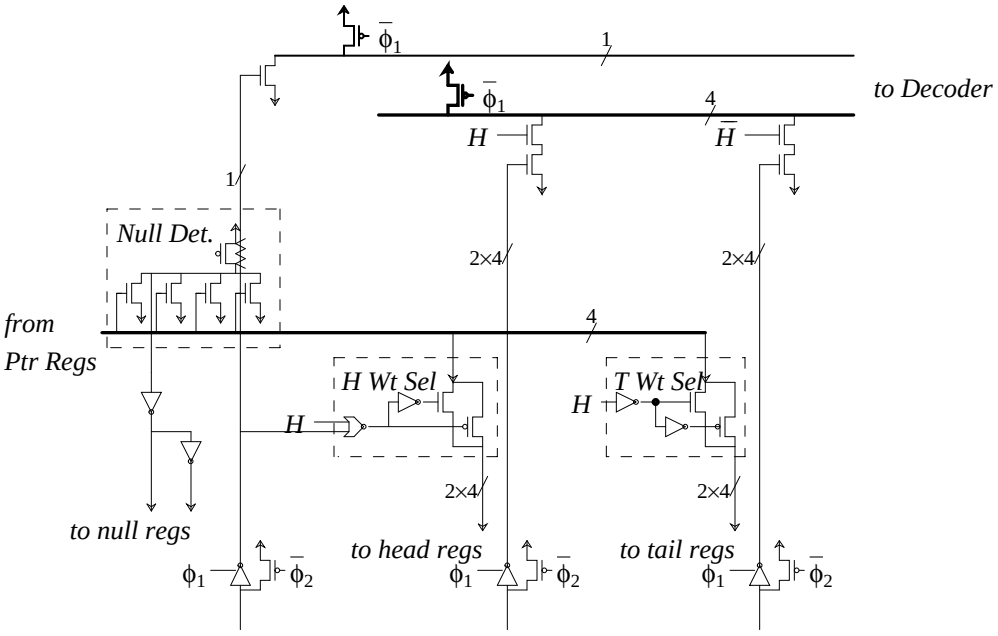


Figure 7.8: *Buses and Associated Logic for the Head/Tail Register Array.*

The actions which occur during a clock cycle begin with the falling edge of ϕ_2. At that point, the finite state machines which control the circuit assert new values on the control lines (Sec. 7.2.3). On the rising edge of ϕ_1, the *head / tail bus* (a dual-rail four-bit bus) is precharged (Figs. 7.8 and 7.10). The buses which are directly driven by the head, tail and null registers are also precharged (Fig. 7.9). It is important that the H and $\overline{H}$ signals be settled before the falling edge of ϕ_1 to prevent charge sharing between the head / tail bus and the transistors gated by H

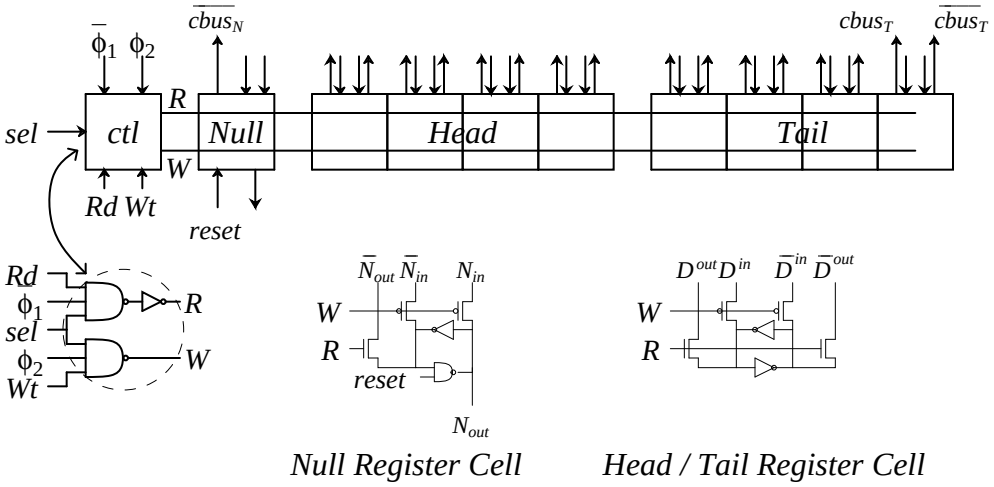


Figure 7.9: *Head / Tail Registers.* The registers which comprise the head and tail register array. A reset line initializes the null registers to '1' (empty queues).

and $\bar{H}$. On the falling edge of ϕ_1, the read signal may be asserted for a head / tail register pair, which begins the *addressing* operations (Sec. 7.1.2). Both the head and the tail registers of the selected queue are read — the $H / \bar{H}$ signal controls pass gates which choose which of their values discharges the bus. The decoder (Fig. 7.10) decodes the block identifier during the underlap between ϕ_1 and ϕ_2, and must complete the decoding of the value (the select lines must have settled to their correct value) by the rising edge of ϕ_2.

The critical path for the addressing occurs when the tail of an empty queue is read. In this situation, there is a '1' stored the null register of the queue, and on the read, the $\bar{N}$ line of the bus will be discharged. In the decoder unit (Fig. 7.10), the decode logic for each of the select signals takes as input $\bar{N}$ and four of the eight lines which comprise the *val* bus. Thus, while no single bit of *val* is connected to more than eight of the twelve select signal decoders (one decoder for each of twelve buffer blocks), $\bar{N}$ gates all twelve. When the tail register of an empty queue

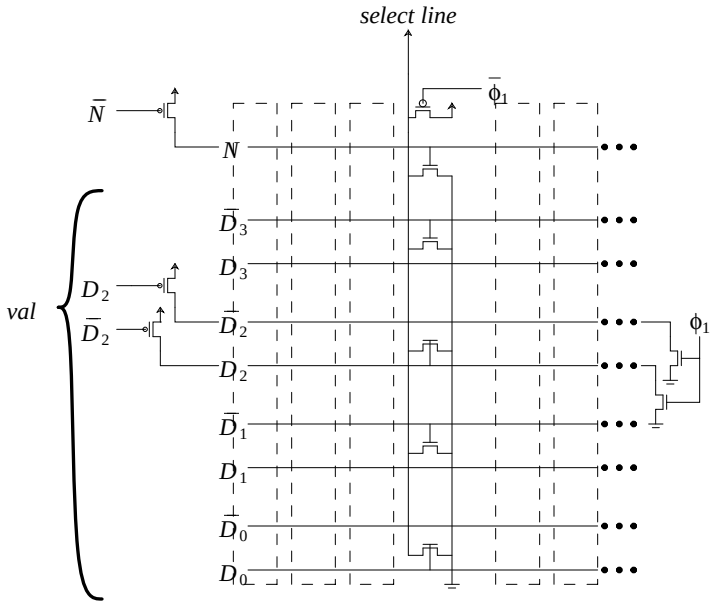


Figure 7.10: *Block decoder.* Decodes four-bit block addresses and asserts a single select line for the addressed buffer block. Address lines are precharged low during ϕ_1, selectively pulled high by *val* on the falling edge of ϕ_1. The select lines are precharged high; the select lines of the unselected blocks must be fully discharged by the rising edge of ϕ_2. The decode cell detailed in the figure is for buffer block 10.

is read, all twelve select lines must be discharged in time to ensure that no pointer register is selected when ϕ_2 rises.

Figure 7.11 shows the results of simulating the reading of a tail register of an empty queue. On the falling edge of ϕ_1, the read enable line is asserted. The selected null register pulls the columnbus ($\overline{cbus}$) down. The inverted value (*cbus*) in turn pulls the $\overline{N}$ line of the bus down. This line, along with the rest of the head / tail register bus, is input for the decoder unit. When the null address line of the decoder is pulled up ($addr_N$), all of the block / pointer register / header register select lines are discharged — no block is selected.

The latency for this operation is just less than 4 ns; for engineering integrity,

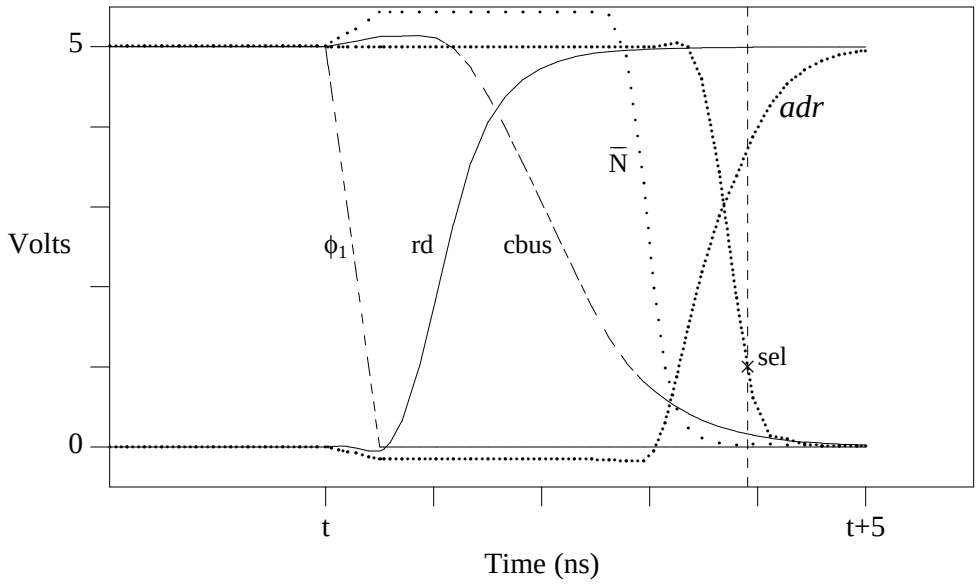


Figure 7.11: *Null Register Read Latency.* The latency from the falling edge of ϕ_1 to when the select lines generated by the decoder unit (Fig. 7.10) drop below 1.0 v.

there must be 5 ns (or more) between the falling edge of ϕ_1 and the rising edge of ϕ_2. If we allow 2 ns for the underlap between ϕ_2 and ϕ_1, and given the 20 ns clock cycle described in Sec. 7.2.1, then the time ϕ_1 is high plus the ϕ_1-ϕ_2 underlap must be 9 ns (or less). If 5 ns are allocated to the underlap between ϕ_1 and ϕ_2, then ϕ_1 is high for 4 ns — which is enough time to settle control signals and precharge the head / tail register buses.

Data movement occurs on the rising edge of ϕ_2. The data movement operations concern the moving of values between the *temporary registers*, the *pointer registers* and the *head / tail registers* (Fig. 7.12). One of three possible operations occurs in the data movement phase. (1) A selected pointer register is read, and its value is written into a head register. The twelve pointer registers (one

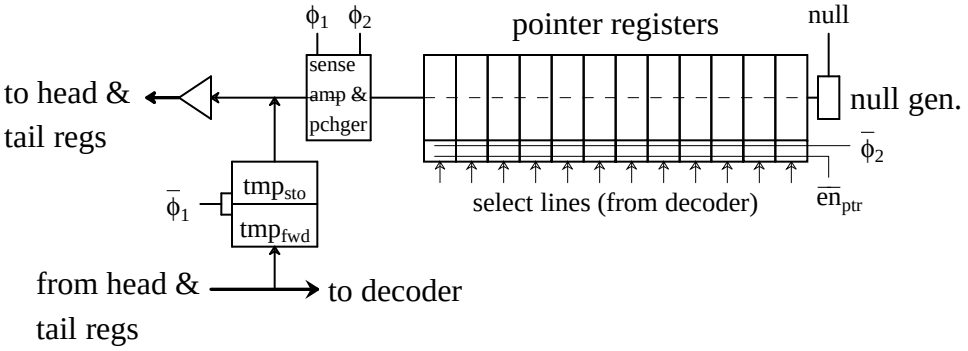


Figure 7.12: *Pointer Register Array.* The pointer registers are accessed via a dual rail bus. The bus is precharged when ϕ_1 is high and discharged during ϕ_2. A sense amplifier accelerates register reads. To move a value from a temporary register to a pointer register, the temporary registers are read on the falling edge of ϕ_1, fully discharging the bus before the rising edge of ϕ_2.

per buffer block) are on a single dual-rail bus — the value is read from a selected pointer register on the rising edge of ϕ_2, and with the help of a sense amplifier and a strong set of buffers is driven into a head or a tail register before the falling edge of ϕ_2. (2) One of the temporary registers is read, and its value is driven into both a pointer register and into a tail register (or both a tail and a head register, if the queue was empty). When a temporary register is read, the read occurs on the falling edge of ϕ_1. When ϕ_2 rises, and a pointer register is selected, the pointer register cannot significantly alter the value on the bus before the sense amplifier is enabled, ensuring that the correct value is driven into the pointer register. (3) The constant NULL (which *is* 0, in this implementation) is written into a selected pointer register.

Figure 7.13 shows the latency of operation (a) — moving a value from a pointer register to a head or a tail register. The rising edge of ϕ_2 causes a pointer register's enable signal to go high. Approximately 2 ns after ϕ_2 rises, the sense

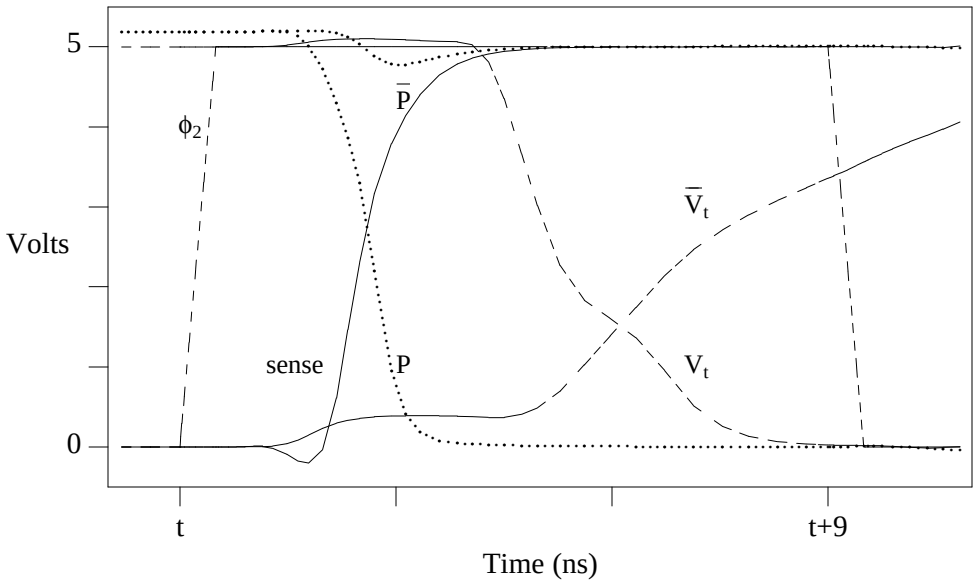


Figure 7.13: *Pointer Register to Tail Register Latency.* The pointer register read begins on the rising edge of ϕ_2. The value read from the selected pointer register is driven into the tail register well before the falling edge of ϕ_2.

amplifier enable (*sense*) rises, forcing the rails of the bus (P and $\overline{P}$) to opposite values. The value on the bus is driven by a buffer through the pass-gates which determine which register set is being written to (head vs. tail), finally flipping the value in the tail register (V_t and $\overline{V}_t$). The values V_t and $\overline{V}_t$ cross 6 ns after the rising edge of ϕ_2.

Figure 7.14 shows the performance of operation (2) (reading a temporary register, driving the value into a pointer, a head and a tail register). The temporary register is read on the falling edge of ϕ_1, giving it an extended period of time to assert its value on the bus. On the rising edge of ϕ_2, a pointer register is enabled, but it has little time to impact the voltage on the bus rails before the sense amplifier is enabled. The sense amplifier cranks the temporary register value into the pointer

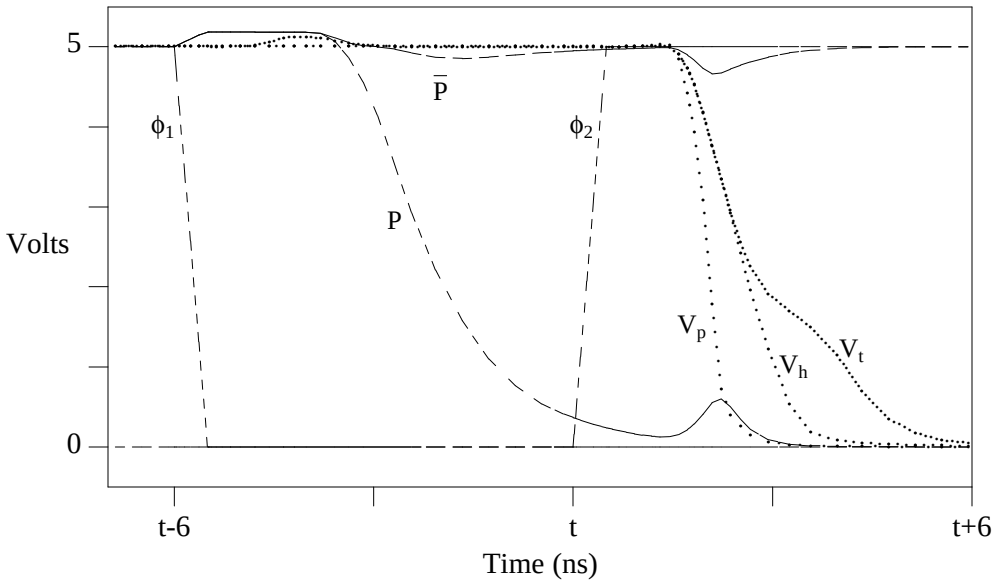


Figure 7.14: *Latency of Temporary Register to Pointer, Head and Tail Register.* The temporary register drives two rails of the bus (P and $\overline{P}$) on the falling edge of ϕ_1. On the rising edge of ϕ_2, a pointer register is selected. Approximately 2 ns later, the sense amplifier is enabled, and the value from the temporary register is cranked into the pointer register (V_p). The value is also written into both the head and the tail register (V_h and V_t). All writes complete well before the falling edge of ϕ_2 ($t+9$).

register. In the mean time, the buffer has been driving the value into the head and tail registers simultaneously; the write to all three registers completes less than 6 ns after the rising edge of ϕ_2 (more than 3 ns before the falling edge of ϕ_2).

Combining the simulation results of the memory array presented in Sec. 7.2.1 with the results presented in this section, we have defined the two-phase non-overlapping clock of the DAMQ Buffer Chip. It is a 50 MHz clock — ϕ_1 has a 15% duty cycle, ϕ_2 has a 45% duty cycle. There is a 5 ns underlap between ϕ_1 and ϕ_2, and a 3 ns underlap between ϕ_2 and ϕ_1. Most significantly, the datapath

operations are not on the critical path; the clock frequency is defined solely by the bandwidth of the memory array. Thus, the DAMQ buffer transmits data with the same raw bandwidth that a (much more simple) FIFO buffer would.

7.2.3. The Finite State Machines

The DAMQ Buffer Chip is controlled by three finite state machines: FSM_{sto} (packet reception and storage), FSM_{rtr} (interaction with the router, buffer block enqueueing) and FSM_{fwd} (packet transmission, buffer block dequeueing). The implementation of these FSMs was performed under three constraints. First, the FSM circuits must utilize the two-phase clock defined in the two previous subsections. Second, the moving of a buffer block from the head of one queue to the tail of another must be accomplished in eight clock cycles or less (each block is eight bytes long and is transmitted / received a byte per clock cycle). The third and most significant constraint is that the hardware and firmware must support demand multiplexing.

7.2.3.1. The FSM Circuit

Figure 7.15 is a schematic of the finite state machine circuit. The FSMs are constructed from NOR-NOR PLAs. On the rising edge of ϕ_2, the output latches are opened and the input latches close, causing the state of the FSM to change. On the falling edge of ϕ_2, the input latches open, latching the current values, and the output latches close, driving the control lines with the signals for the next clock cycle. The exception to this timing is the output lines which are the state bits for the FSM. While driving the control lines on the falling edge of ϕ_2 allows control of

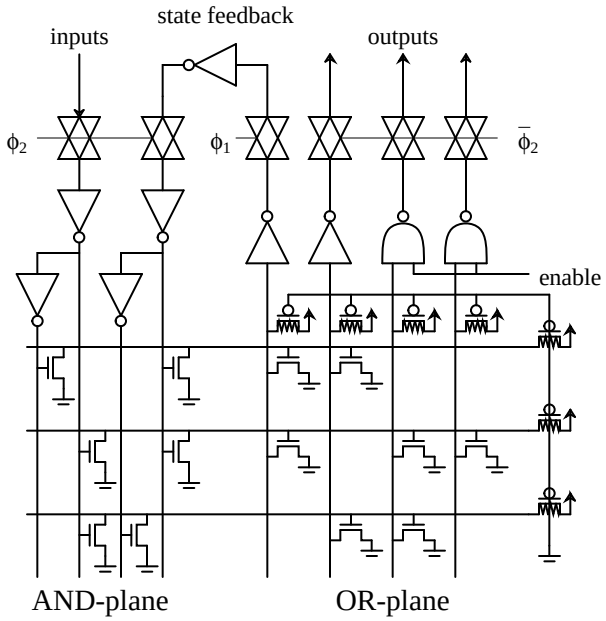


Figure 7.15: *Finite State Machine Circuit.*

actions which are to occur on the rising edge of ϕ_1 (such as driving a header bypass register over the crossbar), passing the state on the falling edge of ϕ_2 would create a race condition. So, the state outputs are passed on the rising edge of ϕ_1.

The FSMs utilize a pseudo-NMOS layout style — both the product and the sum lines are tied to Vcc via weak (resistive) P-transistors. If one or more of the N-transistors in the AND- or OR-plane are closed, the resistance of the P-transistors allows the lines to be quickly pulled down to less than 0.8v. If all of the N-transistors on a line are open, the voltage rises to 5 v. The output signal drivers are located between the OR-plane and the pass gates, rather than beyond the pass gates, to avoid using the non-restored product line voltages (logical zero signals whose level is above 0.5 v) as the capacitive source for the control line drivers when the output pass gates are open. Placing the buffers before the output pass gates necessitates the use of inverters between the state output and input signals.

As Fig. 7.15 indicates, the output drivers can be either inverters or NAND gates. Using NAND gates to drive control signals allows those signals to be masked by an enable signal. The use of an enable signal to mask outputs is discussed in Sec. 7.2.3.2.

The speed at which an FSM operates depends directly upon the size of the FSM, which is a function of the number of inputs and states, the number of product lines, and the number of outputs. FSM_{fwd} is the largest of the FSMs on the DAMQ Buffer Chip — it provides the functionality for transmission which is divided between two FSMs for packet reception. Combining the most heavily populated input, product and sum lines of FSM_{fwd}, the output signal settles ≈ 5 ns after the inputs settle. With every input to the FSMs stabilizing on or before the rising edge of ϕ_2 and the FSMs completing their state transition in 5 ns, the FSM outputs are stable well before the falling edge of ϕ_2.

7.2.3.2. Demand Multiplexing

Under ‘‘normal’’ operation, FSM_{fwd} and FSM_{rtr} use the data path on alternate clock cycles (FSM_{sto} does not share resources with the other FSMs, and thus does not participate in the multiplexing). The signal *SYNC* notifies the FSMs as to which clock cycles they own the data path by alternating high and low each clock cycle. When the SYNC input is low, FSM_{fwd} will control the data path the following clock cycle; when SYNC is high, FSM_{rtr} asserts control the following cycle. Unfortunately, strict time-division multiplexing will not satisfy our performance goals. The need for demand multiplexing of the data path was described in Sec. 7.1.3. Significant resources have been invested to implement

demand multiplexing for the DAMQ Buffer Chip. In this subsection, we present the hardware interface between the DAMQ Buffer Chip and the external router and arbiter (it is interactions with these external units which triggers the demand multiplexing), then discuss the implementation of the *SYNC* signal (this is the signal which indicates to the FSMs which of them controls the data path), and finally we discuss how enable signals and control-signal masking were used to implement ‘‘immediate’’ demand multiplexing.

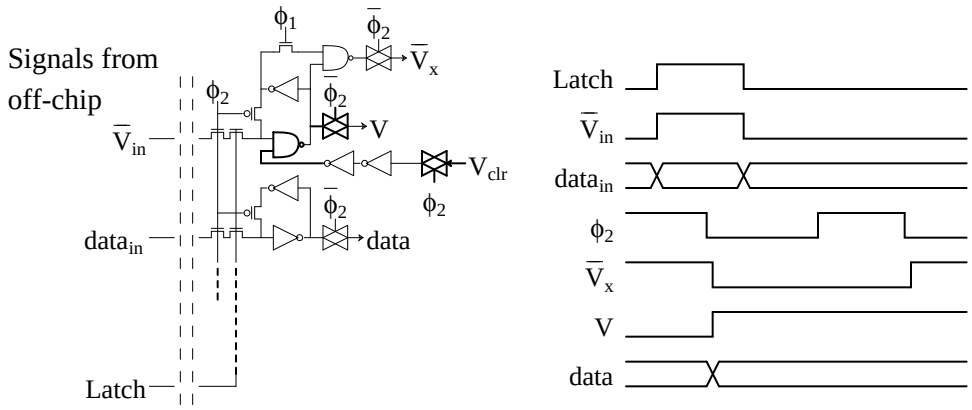


Figure 7.16: *Interface to Router and Arbiter.* Signals from the router or arbiter must be held through the falling edge of ϕ_2. The strobe signal ($\bar{V}_x$) is asserted for a full clock cycle after the data from the router or arbiter is latched (falling edge of ϕ_2 to falling edge of ϕ_2). Once a value is latched, it can only be cleared by the DAMQ buffer control via V_{clr}.

Figure 7.16 is a schematic of the interface register — the router and the arbiter each have an interface register through which they interact with the DAMQ Buffer Chip. The router / arbiter sets the valid signal ($\bar{V}_{in}$) and the input data, and then asserts *Latch*. These signals ($\bar{V}_{in}$, *Latch*, and the input data) must be asserted before the falling edge of ϕ_2 and held until after the falling edge of ϕ_2. Fig. 7.16 shows a timing diagram for input and output signals of the interface register. *Data*

is the data being returned by the router or arbiter: a new header and queue identifier from the router, and a queue identifier from the arbiter. For their queue manipulations, FSM_{fwd} and FSM_{rtr} only differentiate between the free list and ‘‘the queue’’ when accessing head / tail registers — it is the router and arbiter interface registers which specify which queue is being accessed. V is the valid signal — when it is high, a valid queue is being presented by the interface register. $\overline{V}_x$, a ‘‘strobe’’ signal which is asserted for a single clock cycle following the routing or arbitration, is used by the FSMs and the SYNC signal generator to implement the demand multiplexing.

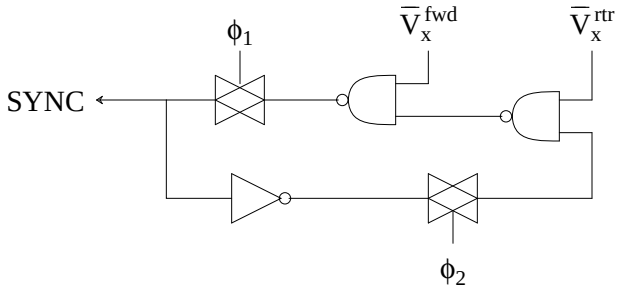


Figure 7.17: SYNC *Signal Generator*. The circuit which generates the signal which specifies which FSMs have control of the data path (*SYNC*).

Figure 7.17 is the schematic for the *SYNC signal generator*. As long as $\overline{V}_x^{rtr}$ and $\overline{V}_x^{arb}$ (the strobe signals from the the router and arbiter) remain high, *SYNC* is alternates high and low every other clock cycle. On any cycle that $\overline{V}_x^{arb}$ is low, *SYNC* will be high. Recall that FSM_{rtr} controls the data path on the cycles *following* SYNC being high (i.e. FSM_{fwd} controls the data path on the cycle that SYNC is high). Similarly, SYNC is forced low when $\overline{V}_x^{rtr}$ is low, unless $\overline{V}_x^{arb}$ is also low. Since the SYNC signal is taken as input by FSM_{fwd} and FSM_{rtr}, the alteration of SYNC’s value cannot be acted upon by the FSMs until the following

clock cycle. It was thus necessary to implement two additional mechanisms to support the ‘‘immediate’’ demand multiplexing which was our goal.

The first problem is how to allow an FSM to commence the queue manipulations associated with packet reception / transmission on the same clock cycle that the valid queue identifier becomes available. This is accomplished by masking the FSM output signals which control the data path. Fig. 7.15 shows output signals being driven either by inverters or by NAND gates. The NAND gates allow control signals to be masked by an enable signal. FSM_{rtr} and FSM_{fwd} each have an enable signal associated with them (EN^{rtr} and EN^{fwd}). The enable signal is asserted during the cycles that an FSM is controlling the reception / transmission of packets. The V signal from the interface register is asserted for the entire length of a packet reception / transmission. To allow the FSMs to initiate packet reception or transmission on the same cycle that the interface register asserts V (Fig. 7.16) (i.e. the cycle following a routing / arbitration), the FSMs actually assert the control signals which will initiate the queue manipulation while they are in the ‘‘ready’’ state, *before* V is asserted. These signals do not interfere with the other finite state machine, nor do they manipulate the data path prior to there being a valid queue identifier in the interface register, because these signals are masked until V goes high. When V^{rtr} for FSM_{rtr} or V^{arb} for FSM_{fwd} goes high, the enable signal goes high and the control signals are passed through to the data path. The FSMs also take V as input, notifying them that reception / transmission has begun.

The second problem is how to allow one FSM to ‘‘grab’’ the data path from the other FSM without waiting for the FSMs to change state. The previous

paragraph described how the grabbing FSM will assert its control signals on the clock cycle that V goes high; what is needed is the ability to mask the other FSMs control signals on that cycle. Once again, the enable signals and control signal masking are the key. In addition to generating a signal V indicating the presence of a valid queue identifier, the interface registers generate a strobe signal $\overline{V}_x$ which is asserted low for the first clock cycle that V is asserted (Fig. 7.16). Fig. 7.17 showed how these strobes are used to manipulate SYNC, which signals to the FSMs which cycles they control the data path. These strobes are also used to mask the FSM data path control signals for a single clock cycle —

$$EN_{rtr} = V^{rtr} \cdot \overline{V}_x^{arb}.$$

$$EN_{fwd} = V^{arb} \cdot (\overline{V}_x^{rtr} + \overline{\overline{V}_x^{arb}}).$$

If both strobes are asserted on the same cycle, FSM_{fwd}'s control signals are not masked.

Having masked some of the outputs of one FSM so that the other can usurp its control of the data path, we must ensure that the skipped operation will be subsequently executed. This is accomplished by having both FSM_{rtr} and FSM_{fwd} remain in every state for two consecutive clock cycles. The FSMs have control of the data path for the first of these clock cycles, and relinquish control for the second; the SYNC signal indicates when to move on to another state. If the data path is usurped from an FSM, it will occur on the first of the two cycles in a given state. When this happens, the SYNC signal will maintain the same value that it had had the previous clock cycle. Since the FSM has remained in the same state *and* SYNC has its same value, the FSM repeats the actions of the previous clock cycle

and ends up spending three consecutive cycles in that state before moving on. This mechanism works because only the signals which control the data path are state dependent; the remaining FSM outputs depend solely upon the inputs, and will thus not be repeated when a state is repeated.

We have thus developed an FSM architecture which supports demand multiplexing. Either FSM_{rtr} or FSM_{fwd} can usurp control of the data path from the other on any given clock cycle, *on the same clock cycle* that a valid output port becomes available. This ‘‘immediate’’ demand multiplexing is possible because (a) the FSM which is grabbing control is idle the cycles prior to gaining control of the data path and (b) the FSM which is losing control is simply delaying its queue manipulations for a single clock cycle, which does not impair its operation.

The mechanism which is instrumental to this implementation of demand multiplexing is the use of NAND-gate-masked outputs for the data path control signals. This allows an idle FSM to be asserting the control signals associated with the first queue manipulation operation on the cycles preceding the arrival of a valid queue signal. These signals are passed through to the data path on the same clock cycle that a new queue identifier becomes available. This also allows the control signals from the other FSM to be masked for a single clock cycle. The FSM whose outputs are thus masked is notified by the SYNC signal to re-assert the data path control outputs the following clock cycle. Thus, FSM_{rtr} and FSM_{fwd} immediately return to their pattern of utilizing the data path on alternating clock cycles.

7.2.4. The Floorplan

The previous sections have established that this implementation of the DAMQ buffer can operate at 50 MHz, and that it is the bandwidth of the memory array that defines this clock frequency. Thus, the DAMQ buffer operates at the same raw bandwidth as a FIFO buffer would (i.e. the complex control logic of the DAMQ buffer does not reduce its performance relative to the more simple FIFO buffer).

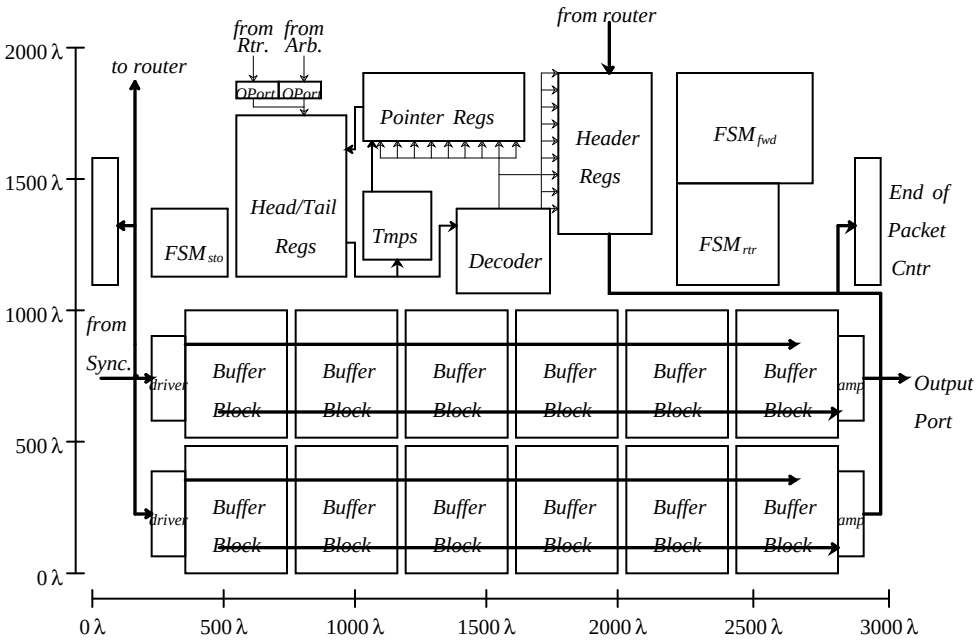


Figure 7.18: The floorplan for the DAMQ buffer.

The remaining issue, then, is a comparison of the DAMQ and FIFO layouts to evaluate their relative silicon usage. The floorplan for the DAMQ buffer is shown in Fig. 7.18. The 96 bytes of storage (slots for three thirty-two byte packets) are broken into two rows with separate read and write buses in order to reduce the latency of reads and writes and to produce a rectangular layout suitable for use as a building block in larger chips. This layout occupies approximately the same area

as a FIFO buffer with 128 bytes of storage (slots for four packets). Our simulations indicate that a network of DAMQ buffers with three packet slots outperforms one with FIFO buffers of four packet slots (Ch. 4), achieving throughputs 23% higher than the maximum throughput of the FIFO network. Furthermore, even in comparison with the maximum throughput of a network with eight-slot FIFO buffers, a network with three-slot DAMQ buffers saturates at 10% higher throughput and a network with four-slot DAMQ buffers saturates at 24% higher throughput. We conclude that the DAMQ buffer architecture, as exemplified by the DAMQ Buffer Chip, is a more efficient utilization of silicon than is a FIFO buffer.

7.3. Summary and Conclusions

The DAMQ Buffer Chip is a VLSI implementation of a multi-queue buffer for use in multiprocessor and multicomputer communication networks. As an interface chip in the node of a multicomputer, it asynchronously receives packets from and transmits packets to other DAMQ Buffer Chips. As an example implementation of the DAMQ buffer, it demonstrates that a non-FIFO buffer can be efficiently implemented in VLSI. In this chapter, we have presented the design of the DAMQ Buffer Chip. Details of the implementation are given, including the floorplan and critical path timing.

We have demonstrated that the DAMQ buffer can operate at high clock rates, despite its complex control. Hence, other factors, such as the inter-chip links, are more likely to limit the raw bandwidth. Further, the DAMQ Buffer Chip supports a minimum switching latency of four clock cycles. This latency is determined by

the serial dependencies of receiving the packet header, routing the packet, arbitrating the crossbar and then traversing the crossbar — the complexity of the DAMQ buffer does not add to this latency.

For a DAMQ buffer with four queues and a packet size of 32 bytes, it was shown that the DAMQ buffer will have one less packet slot available to it than a FIFO buffer occupying approximately the same chip area. Previous work has indicated that, with as few as two packet slots (64 bytes), a network with DAMQ buffers can outperform a FIFO buffer network which has an additional packet slot (Chs. 4 and 6). With three packet slots, the DAMQ buffer network will saturate at a throughput approximately 23% higher than a FIFO buffer network with four packet slots. The DAMQ buffer is thus shown to be a highly effective building block for packet switches, thus demonstrating that for VLSI switches, increased control complexity may lead to higher performance.

Chapter Eight
Flow Control

This dissertation addresses the design of high performance communication switches for scalable multicomputers. Previous chapters focused on the buffer architecture as being a critical functional unit for such a design. This chapter focuses on *flow control*. Flow control is the 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].

The primary task of a flow control mechanism is to prevent *uncontrolled* packet loss or corruption due to contention for network resources. Such contention may be handled by blocking, rerouting, or discarding and retransmitting packets. This chapter focuses on the potential of flow control mechanisms to optimize network performance.

Previous chapters of this dissertation examined the behavior of multistage interconnection networks transmitting uniformly-distributed traffic. However, a uniform traffic distribution cannot be guaranteed in the communication network of a multicomputer. The scalable computer system that we envision consists of hundreds or thousands of computing nodes linked together by a high throughput, low latency communication network. Multiple applications will execute on this system in a dynamic fashion; applications and processes will be spawned independently of one another, requiring system resources to be dynamically re-allocated to accommodate them. Due to the applications which will execute on this system exploiting fine-grained parallelism, and the computing nodes in the system

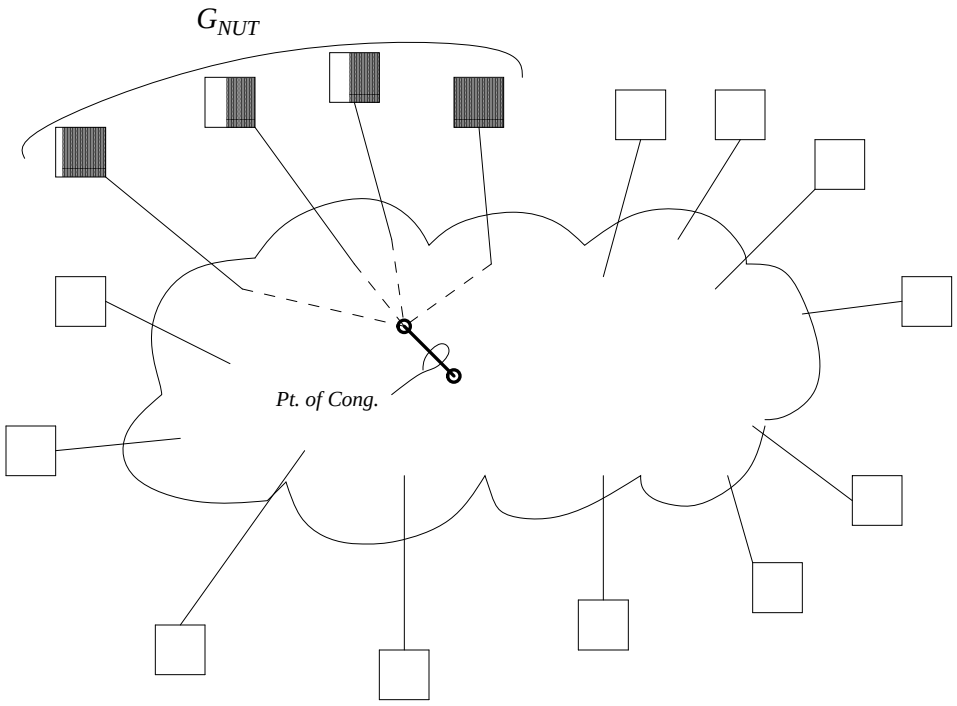


Figure 8.1: *Network Congestion.* A group of one or more senders (G_{NUT}) overload link(s) in the network (the *point(s) of congestion*) for some period of time.

being multitasking and / or multithreading, there will be a large number of packets traversing the network at any given point in time. In a system such as this, one cannot guarantee that communication traffic will remain evenly distributed or that there will be no attempt to over-utilize a network resource. Even if a single application can be analyzed to the extent that it is known to not overload any links within the network, contention between it and other applications for system resources cannot be predicted.

Figure 8.1 depicts such a situation. A group of nodes labeled G_{NUT} (the *non-uniform traffic* group) are transmitting packets with a non-uniform traffic pattern and at a throughput such that a *point of congestion* is created within the

communication network (a point of congestion is a communication link whose applied load is higher than its bandwidth).

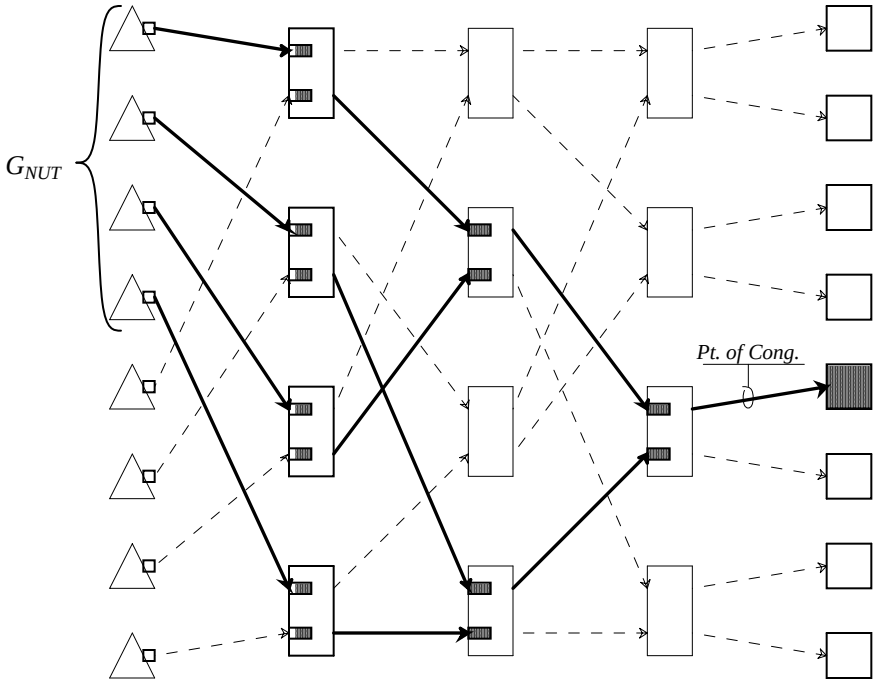


Figure 8.2: *Tree Saturation.* The senders in G_{NUT} overload a link in the network. The congestion propagates back to all senders (*tree saturation*) — including those senders not in G_{NUT}.

When a link is congested, the buffers of the switch which feeds the link become filled with packets waiting to traverse the link. When the buffers become full, the links which supply those buffers become congested, and the process may recur. We term this phenomena *congestion propagation*. When this occurs in a multistage interconnection network (MIN), and is due to the existence of a *hot spot* (a destination address which receives a higher percentage of packets than the other destinations in the network), then it is called *tree saturation* [Pfis85a]. In Fig. 8.2, the group of senders G_{NUT} has created a hot spot. As congestion propagates back

from the point of congestion (the terminal link which connects the network to the hot destination), it forms a tree whose leaves include all of the senders attached to the network. This includes those senders which are not members of G_{NUT}. While these senders may be transmitting no packets to the hot destination, the congestion has propagated to links which are on the path from these senders to other destinations.

Our focus in this chapter is on the ability of flow control to increase the performance of communication networks in the presence of congestion by reducing congestion propagation. In the example in Fig. 8.1, the point of congestion is the bottleneck for the communication performance of the members of G_{NUT}. Flow control cannot increase the link's bandwidth; without some mechanism for eliminating the congested link as a heavily-used resource for the members of G_{NUT}, the bandwidth of the point of congestion is a ceiling for G_{NUT}'s performance†. The key to improving network performance via flow control is to identify the senders which are ''causing'' congestion, and to limit the rate at which those senders can introduce packets to the network. This prevents the remaining senders from having their performance impacted by the congestion propagation.

For a flow control mechanism to prevent congestion from propagating, it must be able to react quickly to the creation of congestion. When a link becomes congested, packets contesting for the link are blocked and must be stored in the

† There are mechanisms which do address contention prevention or resolution. Examples of this are dynamic routing (packet mis-routing in MINs), hot spot detection and elimination at compile time, tree-structured barriers, etc. Some of these mechanisms may utilize flow control in some way — dynamic routing, for example, may use flow control to determine when to use an alternate route for a packet. This interaction is beyond the scope of this dissertation, but is possible future work.

switch's buffers. Since the packet buffers of multicomputer communication network switches will be small (Ch. 4), they will quickly become filled with packets contesting for the congested link. Kumar and Pfister, examining hot spots and tree saturation in MINs [Kuma86], determined not only that tree saturation occurs quickly after the onset of a hot spot (i.e. congestion will propagate quickly), but also that the tree saturation dissipates slowly once the hot spot has ended. Their results indicate that a quick reaction by the flow control mechanism is necessary to prevent congestion propagation and that failure to prevent the propagation will result in network performance degradation that outlives the cause of the congestion.

This chapter details our search for a flow control mechanism which (1) is scalable, (2) reacts quickly to the onset of congestion, (3) protects the performance of senders which are not ''causing'' congestion and (4) does not ''starve'' the point of congestion. The rest of this chapter is structured as follows. In the next section, a taxonomy for flow control mechanisms is established. Sec. 8.2 reviews some previously published flow control mechanisms. In Sec. 8.3 we discuss the characteristics of flow control mechanisms that are suitable for the interconnection network of a scalable multicomputer. In Sec. 8.4, we present the specific flow control mechanisms that are evaluated in this chapter. The evaluation is presented in Sec. 8.5. The simulation methodology and metrics for measuring the performance of the flow control mechanisms are described, and the simulation results presented and evaluated. This section concludes with a summary of the performance of the flow control mechanisms under consideration. Sec. 8.6 describes the hardware necessary to implement those flow control mechanisms

which promote high performance communication. This is all drawn together in Sec. 8.7, in which the costs and benefits of the flow control mechanisms are summarized and evaluated.

8.1. Flow Control Mechanism Taxonomy

In this section a taxonomy for flow control mechanisms is established. Three classifications for flow control mechanisms are defined. Secs. 8.2 and 8.3 use this taxonomy in their evaluation of flow control mechanisms.

One can classify flow control according to the action taken when a buffer becomes full. Under *discarding* flow control mechanisms, packets are allowed to be transmitted to full buffers, but these packets are then dropped from the network. Discarding flow control mechanisms have the ability to determine when a packet has not reached its destination and to retransmit the packet. Under *blocking* flow control mechanisms, on the other hand, packets are prevented from being transmitted to a full buffer. Unless an error occurs, every packet injected into a network with blocking flow control will eventually reach its destination.

A second characteristic by which flow control mechanisms are classified is by where in the network the flow control decisions are made. In [Zhan91], this characteristic as used to divide flow control mechanisms into three categories: *end-to-end*, *global* and *local*. End-to-end flow control schemes specify guidelines regarding individual traffic flows from the senders to the final destinations. With some end-to-end schemes, the destination nodes transmit flow control information directly to nodes which are sending them packets. Other end-to-end schemes restrict the senders without using any dynamic information from the destinations.

Global flow control mechanisms are those in which either flow control information is globally distributed or a centralized server makes flow control decisions for the entire network. Local flow control mechanisms are those in which each switch in the network makes flow control decisions independently, using locally available information. A similar categorization is made in [Gerl80]. In this work, end-to-end flow control is called *entry-to-exit* flow control. Global flow control is called *network access level* flow control, and defined as flow control in which some or all of the network state is used to determine whether packets can enter the network. Local flow control is referred to as *hop-level.*

The third characteristic by which flow control mechanisms are classified is the trigger used to initiate flow control actions. *Predictive* flow control mechanisms attempt to prevent congestion from occurring. They enforce guidelines that attempt to eliminate traffic patterns that are likely to lead to congestion. Flow control actions are triggered when such patterns are detected (e.g., a particular sender has transmitted its ‘‘quota’’ of traffic for a given time period). On the other hand, *reactive* flow control mechanisms are triggered by the onset of congestion. Their goal is to minimize the impact of congestion once it has been detected.

8.2. Selected Existing Flow Control Mechanisms

The previous section established a flow control taxonomy based upon three classifications. This section examines a cross-section of existing flow control mechanisms. The mechanisms are classified using the taxonomy of the previous section, and their applicability to scalable multicomputers is evaluated. The

selection of mechanisms presented is not exhaustive for any category. Rather, these mechanisms are just samples of a large, dense design space. They were chosen to facilitate the rest of the discussion in this chapter.

8.2.1. Hop-Level, Blocking Flow Control

The most basic flow control mechanism is *hop-level blocking flow control* (also called *blocking flow control*). Under this flow control, an input port will not accept packets when there is no room to store them. This flow control mechanism is widely used in scalable multicomputers: 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], all use blocking flow control. This flow control mechanism's sole function is to prevent packet loss due to overflowing packet buffers; it does not have a positive effect on performance. Indeed, due to the undirected nature of its back-pressure, blocking flow control is a fundamental cause of congestion propagation. However, this reactive, hop-level flow control can respond quickly to the congestion, allowing small buffers to be used without fear of packet loss.

Hysteresis has been explored by several researchers [Reis83, Yum83] as a hop-level flow control mechanism to improve network performance. In [Reis83], hysteresis is applied to a buffer with a single class of packets (i.e. a FIFO buffer) by implementing blocking flow control with a lower threshold value for restarting traffic flow than for halting traffic flow. Reiser found that the only benefit derived from using hysteresis in this context is a reduction in the number of times the flow control state for a particular buffer changes. In a communication network with

dedicated flow control lines, the amount the flow control state changes is not an issue (assuming that it changes at most twice per packet).

Yum and Yen [Yum83] implement hysteresis in a buffer with two classes of packets: high-priority and low-priority. In a multi-queue buffer, hysteresis is attached to a queue-oriented blocking flow control scheme similar to maximum usage flow control (discussed in Sec. 8.4). Both the Reiser and the Yum and Yen flow control mechanisms are hop-level blocking flow control mechanisms. Since they use a hysteresis mechanism, it is not clear whether they are predictive or reactive — they may throttle flow at times when there is no congestion, and not throttle at times when there is congestion.

8.2.2. End-to-End Flow Control

Another commonly used flow control mechanism is *windowing* [Klei80, Kerm80, Cerf74]. This scheme works by limiting the number of packets which can be traversing the network between each source / destination pair at any given point in time. It requires that a sender be notified when packets it has transmitted have reached their destination. A sender may transmit packets to a destination at any throughput until the number of unacknowledged packets equals the size of the window. At that point, the sender may not transmit further packets to the destination until an acknowledgment for previously transmitted packets is received. The window limits the amount of congestion propagation which can occur in the network by limiting the number of outstanding packets. The size of the window can be set in a number of ways: a system-wide static window size, prior agreement between source and destination (bandwidth reservation) and dynamic window

sizing based upon feedback from the destination [Kerm80].

Windowing is an end-to-end flow control mechanism. All flow control decisions are based upon the interaction between source and destination nodes — the network itself is treated as a black box. It is also a blocking, predictive flow control mechanism. When the number of outstanding packets reaches a threshold, further packets are blocked from entering the network. While the number of outstanding packets may be indicative of the existence of congestion, there is no direct detection of congestion. Windowing is used in conjunction with a hop-level blocking or discarding flow control mechanism which prevents packet loss when congestion occurs.

Leaky bucket [Turn86, Rath89] is an end-to-end flow control scheme based upon bandwidth reservation, rather than network feedback. When a sender reserves a path to a destination, it is assigned a maximum throughput. It is guaranteed that, if the sender does not exceed its maximum throughput to any destination, then the links in the network will not be over-utilized. If, however, this limit were to be strictly enforced, then the network would be under-utilized due to the bursty nature of computer communication. So, to increase the network utilization, the instantaneous throughput of the senders' communication is not restricted. Rather, when the sender is not using all of the bandwidth it has reserved it accumulates *credits* which allow it to use more than its reserved bandwidth at a later time. When the sender sends data, it can transmit at saturation throughput until it has used up all of its credit, at which point it becomes limited to the allocated bandwidth. To prevent a sender from accumulating too much credit (which would enable it to flood the network with packets), there is a maximum

number of credits which can be accumulated; credits accumulated beyond this are discarded (the leak in the bucket).

Leaky-bucket flow control, like windowing, is a predictive, blocking, end-to-end flow control mechanism. There are many such flow control mechanisms described in the literature: [Rama91, Mukh86, Kalm90, Clar88, Cher89]. End-to-end flow control mechanisms limit congestion by restricting the rate at which individual senders can inject packets into the network or the total number of packets from each sender which may be in the network at one time. The extent to which such schemes reduce congestion depends on the severity of restrictions on the senders. However, tight restrictions on the senders are likely to result in under-utilization of the network. Under most practical end-to-end policies, congestion internal to the network is still possible. Hence some other flow control mechanism is needed in addition to the end-to-end scheme.

An additional problem with end-to-end schemes is that each sender must maintain state information regarding its traffic to each of its destinations. This can limit scalability. In practice, with multicomputers, end-to-end flow control is often used at the application level. As stated earlier, this does not eliminate the need for flow control at the interconnection network level, especially (but not exclusively) if there are multiple independent applications running on the system.

VirtualClock [Zhan91] is an alternative to ‘‘straight’’ end-to-end flow control. It is a hybrid flow control mechanism, operating at both the end-to-end and the hop levels. This gives it the functionality of an end-to-end flow control mechanism with the ability to detect and prevent internal network congestion.

VirtualClock is implemented at the switch level; it determines whether or not

packets can proceed toward their destinations on a hop-by-hop basis. In order to perform end-to-end flow control, VirtualClock is built on top of virtual circuits. Each circuit has associated with it the maximum bandwidth that it will support. Every switch in the network maintains a separate virtual clock for every virtual circuit crossing the switch. The circuit's virtual clock is incremented each time a packet arrives at the switch on that circuit. The amount the clock is incremented depends upon the bandwidth assigned to the circuit — packets traversing circuits which have been allocated large bandwidths cause smaller increments than packets on low-bandwidth circuits. Flow is throttled if a circuit's clock advances ahead of a real-time clock. While flow control decisions are being made on a hop-by-hop basis, based upon information locally available, this is a blocking, predictive flow control mechanism operating on packet flows not unlike the end-to-end flow control mechanisms discussed earlier.

While VirtualClock has demonstrable strengths as a flow control mechanism for loosely-coupled systems [Zhan91], it has two drawbacks which make it inappropriate for a scalable multicomputer. First, it requires that significant computing resources exist at each switch, in order to maintain the virtual clock for each circuit which traverses the switch. Second, VirtualClock is a predictive flow control mechanism, and, as is described in Sec. 8.3, will potentially under-utilize a communication network. Thus, while VirtualClock is very effective in loosely-coupled distributed systems, there is reason to believe that it would be both unreasonably expensive and significantly less effective for scalable multicomputers.

8.2.3. Global Flow Control

In [Scot90], a global feedback flow control scheme is proposed for multistage interconnection networks (MINs) operating with a synchronous communication protocol (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.

This is a blocking, global, reactive flow control mechanism. This combination of characteristics imposes requirements on the system. First, because reactive mechanisms take action after congestion has occurred, and because there is a significant latency between detecting congestion and reacting to it, this mechanism is implemented in conjunction with hop-level blocking flow control to prevent packet loss. In addition, the last switching stage is equipped with large buffers to prevent congestion from propagating during the period between detection and reaction.

The global flow control mechanism proposed by Scott and Sohi has several strengths. First, it is simple to understand and to implement. Second, the scheme is effective for the size and type of system the authors simulated — multistage

interconnection networks with inputs and outputs numbering in the hundreds. The scheme does have weaknesses, though, which make it less appropriate for scalable multicomputer systems.

First, the feedback network itself is expensive. For a MIN, the feedback network will have approximately half the number of links that the communication network itself has. In addition, the scheme is oriented towards MINs. The problem with implementing their scheme over commonly used topologies such as hypercubes, fat trees, tori, and meshes is that these topologies have non-uniform distances between source / destination pairs. Implementing the flow control feedback network with these topologies presents significant technical difficulties. The long feedback latency of the flow control mechanism would be tolerable for nodes communicating over a great distance, but it would not work at all for nodes whose packets traverse a small number of hops.

A third issue which makes this scheme untenable is that this flow control mechanism is incapable of detecting NUT spots internal to the communication network. It is similar to the end-to-end protocols discussed previously, in that only the endpoints of the network are considered by the flow control mechanism. However, unlike the end-to-end flow control mechanisms, if a point of congestion was created on an internal link, the congestion would propagate indefinitely.

The crucial problem with MIN feedback flow control is that it is global, and thus not scalable (Sec. 8.3).

A global (network access) flow control scheme which does not attempt to broadcast data is *isarithmetic flow control* [Davi72]. The idea behind this scheme is that there is a pool of tokens available to network. For a node to transmit a

packet, it must first grab a token. The token traverses the network with the packet and is freed upon packet reception. Thus, the token pool limits the total number of packets in the network in a distributed fashion; this is a blocking, predictive, global flow control mechanism.

Kleinrock and Gerla [Gerl80] comment that isarithmetic flow control works well when the traffic is uniformly distributed, but that it ‘‘may lead to unnecessary throughput restrictions, and therefore, to poor performance in the case of nonuniform, time-varying traffic patters.’’ The problem is partly attributed to the fact that, when a source transmits a significant number of packets to a particular destination, that destination will accumulate a large fraction of the token pool which must then be redistributed. If the tokens are sent back to the source of the flow, then that source will end up capturing a large percentage of the total available network bandwidth. If they are not, then the sender will run out of tokens and will be starved.

There are a number of additional problems with isarithmetic flow control. First, if there are enough tokens in the pool to fully load the network under uniform traffic conditions, then it will still be possible to congest the network under non-uniform traffic conditions. Further, with a significant percentage of the tokens associated with packets that are involved in congestion, the non-congested areas of the communication network will be under-utilized (i.e. the negative feedback is not directed to the nodes involved in the congestion). Finally, there is the increased communication latency incurred by having to obtain a token for each packet transmitted. All of these problems are exacerbated as the communication network scales, making isarithmetic flow control inappropriate for a scalable

multicomputer.

8.2.4. Discarding, Hop-Level Flow Control

Dias and Kumar [Dias89] have proposed a packet switching flow control mechanism that displays some of the characteristics of a virtual circuit flow control mechanism (e.g. VirtualClock[Zhan91]). 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 in 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 sending switch re-queues the packet at the tail of the FIFO buffer from which it came (Fig. 8.3). Thus, destination-based flow control is a discarding, hop-level, reactive flow control mechanism. It provides an elegant two-pronged solution to tree saturation. By allowing only a single packet per destination per buffer, a hot spot cannot cause a buffer to become filled, and by requeueing rejected packets at the tail of the list, packets not addressed to the hot spot can bypass those that are.

Destination-based flow control has two key positive features. First, it is *directed* explicitly only towards packets addressed to hot destinations. Second, it is *selective* and exerts back-pressure only on packets addressed to the hot destination. Other packets are not restricted.

The ability to direct the back-pressure comes from the basis for flow control decisions. By using a comparison of destination addresses to make flow control determinations, this scheme assures that no matter how far from the point of congestion the back-pressure is felt, it is only packets which will traverse the point

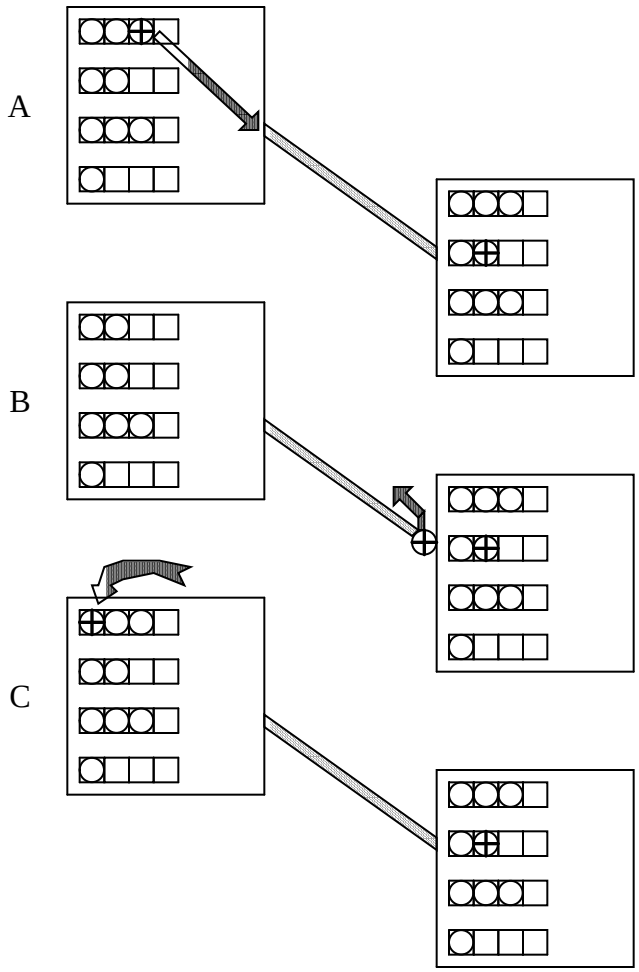


Figure 8.3: *Operation of Destination-Based Flow Control.* Only one packet per destination per buffer is allowed. If the destination of the incoming packet matches that of a packet already in buffer, the incoming packet is discarded (B) and requeued at the tail of the previous buffer (C).

of congestion which are throttled. The ability to avoid throttling those packets which are not involved in the congestion comes from requeueing packets at the tail of the buffer when they are rejected. Destination-based flow control can do this and still maintain the FIFO ordering of packets between each source / destination pair, because there will be at most one packet to a destination in a queue at any

point in time.

Due to its ability to direct back pressure, the behavior of destination-based flow control is similar to the behavior of schemes based on virtual circuits, such as VirtualClock. Functionally, however, destination-based flow control is entirely different from VirtualClock; it is hop-level (not end-to-end), packet switching (not based upon virtual circuits) and reactive (not predictive).

8.3. Flow Control Schemes for Tightly-Coupled Interconnection Networks

A large number of flow control mechanisms have been proposed in the literature; Sec. 8.2 presents only a small subset. The choice of a ‘‘good’’ flow control mechanism depends on the characteristics of the network. The focus of our investigation of flow control is on schemes which are appropriate for the interconnection networks of tightly-coupled multicomputers and multiprocessors. In this section we describe the key characteristics of such tightly-coupled networks. We then discuss the flow control mechanism design space (as described in Sec 8.1) and identify the class of flow control schemes that are most appropriate for tightly-coupled networks.

The property of being *tightly-coupled* indicates both support for and reliance upon low communication latencies. Fine-grained distributed applications require high bandwidth communication paths and communication latencies on the same order as access latencies to local primary memory. These goals must be met in conjunction with efficient support for small messages (tens of bytes). For example, in [Dall87b], it is suggested that the processes of a fine-grained distributed application will transmit a ten-byte packet every one hundred clock cycles. For the

system to be *scalable*, no single object or resource in the system can be a performance bottleneck. A general purpose scalable multicomputer should support multiple applications executing simultaneously, with new applications being initiated on the system at any given point in time. With multiple applications competing for shared resources, predicting resource utilization (including network resources) is not feasible.

Very few large-scale multicomputers implement discarding flow control [Crow85, Rett90]. The first problem with discarding flow control is that, when a packet is discarded, it must be retransmitted by the sender and then re-traverse all of the communication links that it had traveled prior to being discarded. This is contrary to the goal of minimizing transmission latency. It also creates a positive feedback for network congestion — when congestion occurs, packets are discarded more frequently and thus require *more* network bandwidth than they did at lower communication throughputs.

The second problem deals with returning acknowledgments to packet senders. Since the regular packets are often small, if each packet is acknowledged separately, the acknowledgement traffic will consume a significant fraction of network bandwidth. The acknowledgement traffic can be reduced by only acknowledging groups of packets (i.e., windowing). However, this increases the average packet latency since there will be longer delays for the retransmission of discarded packets. A third problem with discarding flow control is that it may fail to maintain FIFO packet ordering between sender and destination. This requires packet reordering by higher levels of the communication protocol.

Destination-based flow control is a discarding flow control scheme which

behaves similarly to blocking flow control — FIFO packet ordering is maintained and communication latencies are kept low by performing discarding / retransmission on a hop-by-hop basis. For these reasons, an evaluation of destination-based flow control is included in this chapter, even though it is, technically, a discarding flow control mechanism. Enhancements to the destination-based flow control described in Sec. 8.4 minimize link re-traversals, making it even more similar to blocking flow control.

Predictive flow control mechanisms attempt to prevent congestion by throttling traffic which has the potential to cause congestion. In a general-purpose multicomputer, traffic patterns may be highly irregular. If the predictive flow control severely restricts traffic, it may unnecessarily reduce the available network bandwidth. However, if the predictive flow control is less restrictive, it may fail to significantly reduce the probability of congestion. Thus, in this case, the utility of the predictive flow control is in doubt and a different ‘‘good’’ flow control mechanism is needed in addition to the predictive scheme.

Section 8.2.2 discusses some of the problems of implementing end-to-end flow control in a scalable multicomputer. The problems generally stem from the conflict between low latency and scalability. As a system scales, destinations move further away from sources, increasing the latency of end-to-end feedback. Furthermore, most end-to-end flow control mechanisms are not capable of explicitly detecting and reacting to congestion that occurs in the middle of the network. There are end-to-end flow control schemes that operate without any feedback of dynamic network activity. However, as discussed earlier, such predictive schemes are not likely to perform well in the dynamic environment of a

general-purpose multicomputer.

Since they must either process all flow control information centrally or broadcast changes in flow control state, global flow control mechanisms scale poorly, as well.

Hop-level flow control displays the characteristics desired in a scalable multicomputer. Since flow control decisions are made independently by each switch, there is no increase in complexity as the system scales. Similarly, because all decisions are based upon locally available information, scaling does not impact the latency of the mechanism. A distributed flow control mechanism is also more robust in the face of faults; the failure of a single link or switch will only impact those nodes adjacent to the failure. Finally, there are the benefits of having the same flow control hardware and algorithm repeated throughout the network — the system is easier to design, build and maintain (e.g. MIT's J-Machine[Dall90a]).

8.4. Hop-Level Flow Control

The previous section of this chapter discussed the reasons that end-to-end and global flow control mechanisms are unsuitable candidates for implementation in a scalable, tightly-coupled computer network. Hop-level flow control, on the other hand, displays a number of features which mark it as appropriate for such a system. With hop-level flow control, each sender and switch in the system makes flow control decisions based upon information that is locally available. Thus, the complexity, cost and / or requirements of each flow control calculation is independent of the network size. Second, flow control information is propagated through the network via *back pressure* [Gerl80]. This avoids the use of a broadcast

or a multicast flow control feedback, which, as was discussed in the previous section, does not promote scalability. The remainder of this section describes the hop-level flow control mechanisms evaluated in this chapter.

8.4.1. Blocking Flow Control

Being the flow control mechanism used in so many of the current multicomputers, hop-level blocking flow control was an obvious choice for evaluation. One of the key choices in the design of blocking flow control is whether (1) to support blocking of transmissions after only part of a packet has been transmitted or (2) to restrict blocking to occur only between packets. In the former case, a buffer accepts packets right up to the point where the buffer is completely filled. If the buffer is in the middle of receiving a packet at that point, the reception / transmission is halted, and the packet is stored for some period of time in two (or more) switch buffers. *Wormhole routing* [Dall86] requires this form of blocking flow control since the buffers are too small to store an entire packet in a single node. With wormhole routing the nodes *must* support *virtual cut-through* [Kerm79], i.e., be able to begin forwarding a packet before all of it is received.

In the second case — restricting blocking to between packets — buffers must halt flow if there is not enough space in the buffer to store an entire packet. While this style of blocking flow control restricts the maximum packet size in a network, it also ensures that once a packet transmission begins, it will complete without interruption. This form of blocking flow control can be used with nodes that support virtual cut-through as well as with nodes that do not. We evaluated the

second type of blocking flow control with nodes that do support virtual cut-through. Specifically, we assume a maximum packet size of thirty-two bytes so flow is halted when there are less than thirty-two bytes available in a buffer.

8.4.2. Maximum Usage Flow Control

In their survey of flow control schemes, Kleinrock and Gerla [Gerl80] discuss *channel queue limits.* This is a mechanism which limits the amount of buffer space a particular class of packets can occupy in a buffer pool. In the case of hop-level flow control, the packets are classified according to which output port they will be leaving the switch on. There are a number of ways in which the buffer memory usage can be limited: each output port can have a maximum memory allowance (sharing with maximum queues — SMXQ), each can be guaranteed a minimum availability (sharing with minimum availability — SMA), SMXQ and SMA can be combined, or the buffer space an be strictly partitioned (see the SAMQ switch, Ch. 3). In their discussion of channel queue limits, Kleinrock and Gerla refer primarily to [Irla78].

In [Irla78], Irland found that, under non-uniform traffic, complete partitioning (SAMQ buffers) performed slightly worse than the other channel queue limit mechanisms, while at high offered loads (saturation), all channel queue limit mechanisms supported approximately the same throughput. He also found that switches with unrestricted sharing performed significantly worse than those with some form of channel queue limits, *even under uniformly distributed traffic.*

This dissertation refers to channel queue limits as *maximum usage flow control* (the networks we evaluated do not have channels). Under maximum usage

flow control, each queue of a buffer has a bit of flow control state associated with it. When the length of the queue (the number of *buffer blocks* occupied by the queue) exceeds a threshold (or fills the queue, in the case of static buffer allocation), then that queue's flow control signal is asserted. Similarly, when the length is equal to or less than the threshold, the signal is dropped. If there are fewer than thirty-two bytes (four blocks) available in the buffer, then all of the queues' flow control signals are asserted.

The use of maximum usage flow control implies a dynamically allocated buffer (there is sharing of storage space between queues of packets destined to different output ports). As discussed in Ch. 3, implementation issues direct one to implement switch buffers at the input ports. Thus, maximum usage flow control is evaluated using dynamically allocated, multi-queue buffers located at the input ports — the DAMQ buffer.

We also simulated *static buffer allocation*; this is a form of maximum usage flow control in which the buffer is strictly partitioned (statically allocated) among the queues. Chs. 4 and 6 examine static buffer allocation as an implementation option for non-FIFO buffers: the SAMQ and SAFC buffers. Their performance under non-uniform traffic conditions is presented as part of the evaluation of maximum usage flow control.

Maximum usage flow control requires that packets be pre-routed. *Pre-routing* is the determination of which output port *of the next switch* a packet will traverse. Thus, for each hop through a network using maximum usage flow control, each packet must be routed twice: once to determine which output port the packet is destined for, and once to determine which queue the packet will be appended to in

the next switch. If the packet at the head of a queue will be append to a queue on the next switch which is currently blocked, then the crossbar arbiter will not connect the buffer to that output port; the packet is blocked until the destination queue unblocks. In our evaluation of maximum usage flow control, we looked at blocking thresholds of six blocks and ten blocks and static buffer allocation (both SAMQ and SAFC buffers), all with 128 byte buffers.

8.4.3. Two-Counter Flow Control

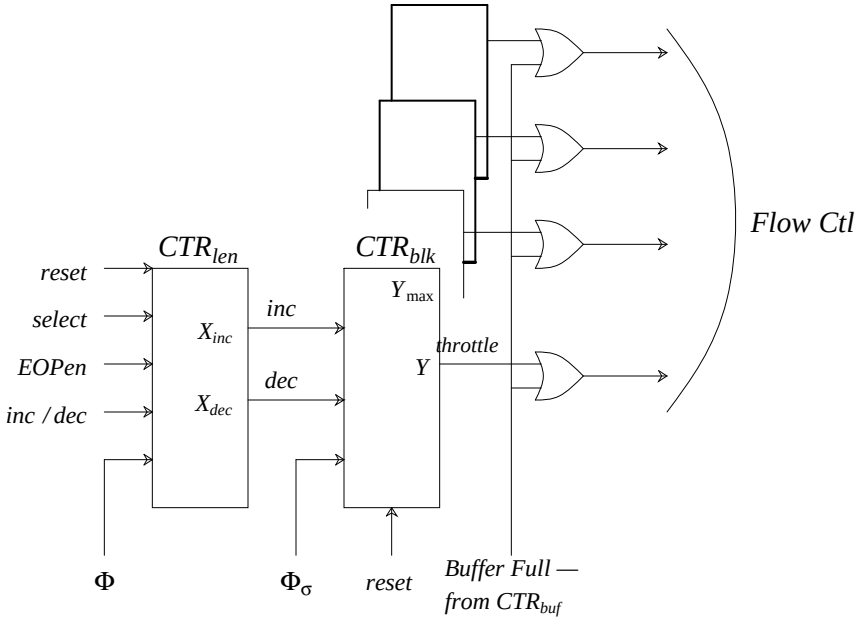


Figure 8.4: *Block Diagram, Two-Counter Flow Control.* CTR_{len} indicates how many blocks of the buffer are occupied by the queue. CTR_{blk} is incremented / decremented once every σ clock cycles, depending on whether CTR_{len} is above X_{inc} or below X_{dec}. Flow to the queue is throttled when the value of CTR_{blk} is above Y, and is throttled to all queues when the buffer is full. The maximum value of CTR_{blk} is $Y_{\max}$.

Two-counter flow control is a general implementation of hysteresis flow

control (it subsumes the hysteresis flow control mechanisms discussed in Sec. 8.2). It is similar to maximum usage flow control (channel queue limits), in that it limits the amount of buffer space a single packet queue can occupy. Two-counter flow control is more complicated than the hysteresis schemes mentioned above, in that it provides three separately configurable mechanisms for hysteresis. A block diagram of the flow control mechanism for a single queue is shown in Fig. 8.4. Each queue has two counters associated with it. CTR_{len} keeps track of the current length of the queue. CTR_{blk} maintains the current flow-control state for the queue. When the value of CTR_{blk} reaches the threshold Y, the queue blocks (i.e. does not accept packets). When its value drops below Y, the queue unblocks. CTR_{blk} is updated every σ clock cycles (ϕ_σ, shown in Fig. 8.4, is the system clock divided by σ). Whether CTR_{blk} is incremented, decremented or left unchanged on an update cycle depends upon the number of buffer blocks occupied by the queue. When the flow control state is updated, CTR_{blk} is incremented if more than X_{inc} buffer blocks are occupied by the queue, and decremented if less than X_{dec} are occupied.

σ	Interval between CTR_{blk} updates.
X_{inc}	CTR_{len} threshold, beyond which CTR_{blk} is incremented.
X_{dec}	CTR_{len} threshold, below which CTR_{blk} is decremented.
Y	CTR_{blk} threshold, at which packets to the queue are blocked.
$Y_{\max}$	Maximum value of CTR_{blk}; further increments have no effect.

Table 8.1: *Two-Counter Flow Control Parameters.* See Fig. 8.4 for a block diagram of the two-counter flow control hardware.

The behavior of the two-counter flow control mechanism is controlled by the values of the full and not full thresholds (X_{inc} and X_{dec}), the CTR_{blk} threshold at which the queue blocks (Y), the maximum CTR_{blk} value and rate of ϕ_σ (see

Tab. 8.1). Being a flexible and fully parameterized flow control mechanism, two-counter flow control is an ideal test-bed for hysteresis flow control. For example, if the blocking threshold and maximum value of CTR_{blk} are both set to one ($Y = Y_{MAX} = 1$), σ is set to one ($\phi_\sigma = \phi$ — the system clock), and $X_{inc} = X_{dec} - 1$, the resulting flow control behaves like the maximum usage flow control described above. If the above parameters are changed such that $X_{inc} \geq X_{dec}$, then then the flow control is identical to previously published hysteresis flow control schemes, which depend strictly upon separate blocking and unblocking thresholds [Reis83, Yum83].

Table 8.2 displays the two-counter flow control parameter configurations which we evaluated, with an explanation of the intended behavior of each of the parameter settings. The goal in choosing these five versions (H, T, tH, A and aH) of two-counter flow control is to broadly explore the behavior-space of this flow control mechanism. While this does not represent an exhaustive search of the two-counter parameter space, it is expected that evaluating these five parameter settings will broadly indicate which aspects of two-counter flow control improve network performance. We found that assigning more ‘‘extreme’’ values to the parameters reduced the performance of the resulting network under a variety of traffic loads.

name	settings	explanation
H	$\sigma = 1$ $X_{inc} = 8$ $X_{dec} = 4$ $Y = 1$ $Y_{\max} = 1$	Hysteresis flow control, as presented in [Reis83, Yum83]. The queue is throttled when there are more than eight buffer blocks in the queue; flow is restored when there are less than four blocks in the queue.
T	$\sigma = 20$ $X_{inc} = 6$ $X_{dec} = 6$ $Y = 1$ $Y_{\max} = 1$	Infrequent sampling. The queue is throttled when there are more than six buffer blocks in the queue; flow is restored when there are less than six buffer blocks in the queue. CTR_{len} is sampled once every twenty clock cycles.
tH	$\sigma = 20$ $X_{inc} = 8$ $X_{dec} = 4$ $Y = 1$ $Y_{\max} = 1$	Hysteresis with infrequent sampling.
A	$\sigma = 20$ $X_{inc} = 8$ $X_{dec} = 8$ $Y = 2$ $Y_{\max} = 4$	Accumulator hysteresis. CTR_{blk} increments every 20 clock cycles that $CTR_{len} > 8$, decrements every 20 cycles that $CTR_{len} < 8$. CTR_{len} will be > 8 for at most 40 cycles before flow is halted, and < 8 for at most 40 cycles before flow is resumed.
aH	$\sigma = 20$ $X_{inc} = 8$ $X_{dec} = 4$ $Y = 2$ $Y_{\max} = 4$	‘‘Straight’’ and accumulator hysteresis combined.

Table 8.2: *Parameter Settings for Two-Counter Flow Control.* See Tab. 8.1 and Fig. 8.4 for explanations of the function of each parameter.

8.4.4. Destination-Based Flow Control

The destination-based flow control described by Dias and Kumar in [Dias89] was implemented with FIFO buffers. Since the superiority of DAMQ buffering has already been demonstrated (Chs. 4 and 6), we simulated destination-based flow

control with DAMQ buffers. One could also evaluate destination-based flow control with SAMQ or SAFC buffers; in the context of this chapter, this represents a combining of multiple flow control mechanisms (static buffer allocation and destination-based flow control). Sec. 8.5.10 examines the performance of combined flow control mechanisms.

In our simulator, every time a buffer using destination-based flow control receives a packet, it responds to the sending switch with either a positive or a negative acknowledgment (ACK / NACK). The sending buffer does not de-allocate the buffer memory occupied by the packet until the ACK is received. When a packet is rejected, the sending switch receives the NACK within six clock cycles of the initiation of transmission, which is considerably less than the time required to send the maximum-length packet (thirty-two bytes, in our simulator). The packet is re-linked to the tail of the queue, the sending buffer is considered to be ‘‘busy’’ for the time required to transmit the entire packet, and the communication link is ‘‘busy’’ for the time required to transmit the entire packet plus the inter-packet idle period (Sec. 6.1.1).

We enhanced our implementation of destination-based flow control by aborting transmissions as soon as a NACK is received (since Dias and Kumar worked with a synchronous network, this optimization was not applicable to their work). This enhancement has the potential to reduce the amount of time spent by buffers and output ports transmitting already-rejected packets. In the case of simulating SAMQ and SAFC buffers using destination-based flow control with this extension, we do not account for the overhead of moving the packet from the head of the queue to the tail, nor do we consider the possible conflict of a packet arriving

while this movement is taking place. For the case of implementing the scheme on a DAMQ buffer, the buffer cannot transmit for a period of time equal to two clock cycles for each buffer block occupied by the packet, to account for the time required to manipulate the linked lists. We refer to the enhancement of aborting transmissions and requeueing as soon as a NACK is received as *abortive transmissions.*

We also designed a second extension to destination-based flow control. This extension operates on packets which have been rejected from a succeeding switch, and requeued in a preceding switch. When the packet comes back to the head of its queue, whether or not it is transmitted depends upon whether the succeeding switch has indicated that the packet which blocked this packet on its previous transmission attempt has been transmitted from the succeeding switch. If the *BLOCKER* packet has not been transmitted, then the *BLOCKEE* packet is re-linked at the tail of its queue without attempting to transmit it. The idea behind this second enhancement is to not transmit packets which are known will be discarded by the next switch, with the goal of reducing the number of times a packet is transmitted and rejected. This extension is referred to as *restricted retry*.

8.4.5. Path-Based Flow Control

Path-based flow control is identical to destination-based flow control, except for the basis it uses for throttling traffic. Instead of rejecting packets whose destination matches the destination of a packet residing in the buffer, path-based flow control rejects packets whose next n hops are identical to the next n hops of a packet residing in the buffer. The idea behind path-based flow control is to achieve

the functionality of destination-based flow control, with the additional ability to detect congestion which is not associated with a particular destination.

Path-based flow control can be implemented in an identical fashion to destination-based flow control, including the enhancements. For the four stage omega network of our simulator, setting $n = 4$ causes path-based flow control to behave identically to destination-based flow control. Setting $n = 1$, on the other hand, causes the buffers to behave like SAMQ buffers which can store one packet per queue. The simulation results of path-based flow control presented in this dissertation use $n = 2$.

8.5. Flow Control Performance Evaluation

This section presents an evaluation of the performance of the flow control mechanisms which were discussed in the previous section. First, the network simulator that was used for the evaluation is characterized. Then, the methodology of the evaluation is described. Finally, the simulation results are presented and used to evaluate the flow control mechanisms' performance.

8.5.1. The Simulator

The flow control mechanisms were evaluated using the asynchronous network simulator presented in Ch. 6. Thus, the flow control mechanisms were simulated in a computer network supporting virtual cut-through, variable-length packets and multi-packet messages. The simulator was configured as a 256×256 multistage interconnection network in the omega topology. The network was composed of four stages of sixty-four 4×4 switches. For the simulations, each switch was

implemented with a 128-byte buffer† at each input port (unless otherwise stated). The senders generated messages with geometric interarrival times, as described in Ch. 6. The maximum packet size allowed in the network was thirty-two bytes; when senders generated messages longer than thirty-two bytes, the messages were broken into multiple packets which are transmitted sequentially at maximum throughput.

The *applied load* of the system is characterized by two components: the distribution of message lengths and the distribution of message interarrival times. As stated above, the message interarrival times are geometrically distributed, so for each sender there is a fixed probability of a message being generated on each clock cycle. We specify this probability as *messages per sender per clock cycle* (m/sc). Throughput (thpt) is measured in *bytes per link per clock cycle* — since the links are one byte wide, and can transmit one byte per clock cycle, throughput is presented as the fraction of available bandwidth utilized (i.e. $0 \le thpt \le 1$). Average latency (lat) is the average number of clock cycles from packet creation to the first byte of the packet reaching its destination. See Ch. 4 for a more detailed discussion of the simulator.

---

† Flow control mechanism functionality improves with larger buffers. We restricted our buffers to 128 bytes (storage for four thirty-two byte packets) because, in Ch. 6, we established that larger buffers provided negligible performance improvement under conditions of uniformly distributed traffic, and the cost of additional buffer memory is linear. The diminishing benefits of additional buffer space is a well-known result [Dias81].

8.5.2. Evaluation Methodology

In the introduction to this chapter, it was mentioned that network congestion is caused by a group of senders (G_{NUT}) attempting to over-utilize one or more links within the network. Further, we asserted that, unless some mechanism such as dynamic routing reduces the dependency of G_{NUT} on the critical link(s), the communication throughput of G_{NUT} is determined by the bandwidth of the critical link(s), and no flow control mechanism is capable of improving it. Flow control can, however, improve the performance of the communication network as a whole by preventing the congestion caused by G_{NUT} from reducing the network bandwidth available to the other nodes in the system. Thus, we evaluated the flow control mechanisms by performing network simulations in which a set of senders was designated to be G_{NUT}. These senders were directed to transmit packets in a manner that would create congestion in the network. Various aspects of the performance of the network was measured under these conditions to determine the degree with which each flow control mechanism could preserve network performance.

Congestion within a network can display a range of characteristics which are determined by (a) the relative locations of the senders which comprise G_{NUT} and (b) with what non-uniform traffic pattern does G_{NUT} create congestion. We have developed a suite of network benchmarks in order to explore the performance of flow control mechanisms under a variety of congestion patterns. This suite is intended to stress the communication network in a variety of ways, with the idea that a flow control mechanism that can successfully handle a wide variety of stochastically-generated congestion patterns in the simulator will be effective

against application-generated congestion when implemented in an actual multicomputer.

The benchmarks specify the senders which will comprise G_{NUT} and the pattern with which they will transmit data. For each benchmark, G_{NUT} creates a point or points of congestion within the network by transmitting packets at saturation throughput in the traffic pattern associated with that benchmark. The remaining senders in the system are partitioned into groups according to their interaction with G_{NUT}. The applied load of the senders in these groups is varied from 0.0 m/sc to saturation.

We used three performance metrics to evaluate the flow control mechanisms. The first is the maximum throughput of the network. This statistic reflects the ability of the flow control mechanisms to protect the performance of the ''innocent'' senders (the members of $G_0, G_1, \cdots$) from the network congestion caused by members of G_{NUT}. The second metric is the average latency of G_{NUT} and G_i per throughput. Finally, the utilization of the critical network resources was examined. Since the point(s) of congestion are the performance bottleneck for G_{NUT}, it is highly desirable that these links be fully utilized. For the benchmarks with a static point of congestion, we measured the throughput of data through this link. For the benchmarks with dynamic congestion patterns, we measured the total network throughput to evaluate the critical resource utilization.

Ideally, a flow control mechanism ensures that the nodes which are not members of G_{NUT} are not negatively impacted by the congestion created by G_{NUT}. Thus, the performance of a blocking flow control under uniform traffic conditions is the case against which the flow control mechanisms' performances under the

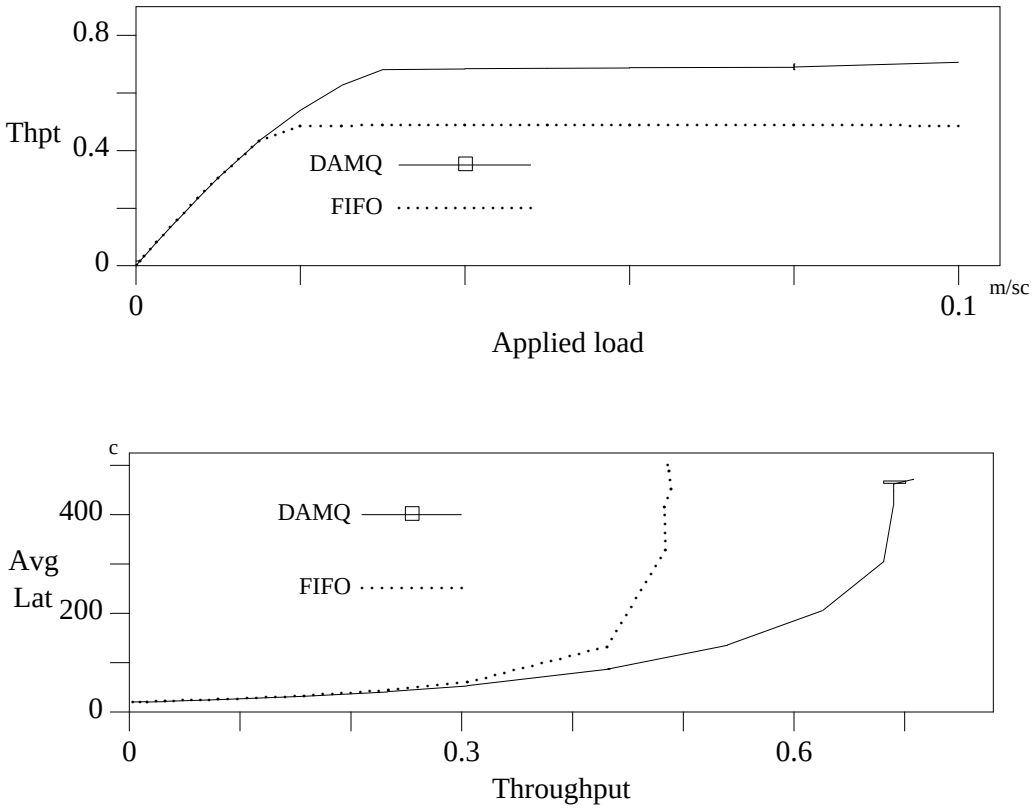


Figure 8.5: *Throughput and Latency, Uniform Traffic Distribution.* 256×256 omega network, composed of 4×4 switches. DAMQ and FIFO buffers. Thirty-two byte packets. Packet destinations randomly chosen, uniform distribution. The rectangles associated with the DAMQ buffer results are the confidence interval measures — see Sec. 8.5.3.

benchmarks is compared. Figure 8.5 shows both throughput vs. applied load and average latency vs. throughput for networks of FIFO and DAMQ switches. All senders in these simulations generate thirty-two byte packets whose destinations are randomly chosen from a uniform distribution. As can be seen from the throughput vs. applied load graph, at lower applied loads the throughput is directly proportional to the applied load, indicating that there is no congestion in the

network. For both FIFO and DAMQ switch networks, as saturation throughput is approached, the lines quickly become horizontal (i.e. additional applied load results in no additional throughput). Similarly, the average latency vs. throughput graph shows only slight increase in the average latency until the saturation throughput is approached, at which point the lines become vertical (i.e., additional applied load results in additional latency, but no additional throughput).

These graphs demonstrate two desirable features which are present under uniform traffic conditions and which a flow control should preserve in networks with non-uniform traffic conditions. The first is that, until saturation throughput is approached, the network is transparent to the senders. That is to say, from the senders' point of view, it appears that each sender is directly connected to all of the destinations, with no network in between. The network is transparent when it imposes no back pressure upon the senders and imparts a negligible latency upon the packets traversing it. The second desirable trait is that, when the applied load approaches and moves beyond the point of maximum throughput, the throughput of the network does not drop. Networks which experience a reduction in throughput when an excessive load is applied are termed *reactive*. Non-uniform traffic conditions can cause a network to be reactive. Reactivity is an undesirable characteristic, in that network throughput is reduced at exactly that point in time when it is most needed — when communication bandwidth is the performance bottleneck.

8.5.3. Confidence Intervals

The simulation results presented in this chapter required extensive CPU cycles to generate. This prevented the generation of confidence intervals for every point on every line in every graph presented. Instead, confidence intervals were generated for three points in a representative subset of lines. The lines which have confidence intervals are indicated in the graph legends by a rectangle across the middle of the line in the legend, and the confidence intervals themselves are presented as rectangles in the graph. The vertical dimension of the rectangle represents the 95% confidence interval of the y-value of that point in the line, and the horizontal dimension reflects the x-value. Tab. 8.3 presents the lines for which confidence intervals exist.

The confidence figures were generated by running the simulator for a ‘‘warm-up period’’ and then executing it for five ‘‘intervals’’. Each interval was of the same length as the simulation runs which generated the other points in the graphs. After each interval, the performance statistics were recorded and then reset for the next interval. The rectangles in the graphs provide quantitative information as to the reliability of the results represented by that data point — there is a probability of 0.95 that, if the results of an infinite number of intervals were collected, the data point representing their average would fall within the rectangle. The rectangles also provide qualitative information as to the reliability of the remaining points in the graphs. All of our confidence intervals are ‘‘tight’’ — there is little variance in the simulation results. Thus, while a confidence interval cannot be generated for a single simulation result, we are confident in the accuracy of the data points which were generated from single simulation runs.

Figure	Line
8.5	DAMQ
8.8	DAMQ
8.9	Blocking
8.10	tH
8.13	DAMQ
8.14	Blocking
8.16	Path-Based
8.19	MU-6
8.20	aH
8.24	MU-6
8.27	FIFO
8.28	SAFC
8.29	A
8.33	Dst-Based, MU-22
8.34	Dst-Based, SAFC
8.36	Dst-Based
8.37	Dst-Based, MU-22
8.37	Dst-Based, SAFC

Table 8.3: *Confidence Interval Map.* The figures and results for which confidence intervals have been generated.

It should be emphasized that there are *three* points for which a confidence interval exists for each of the results specified in Tab. 8.3. For many of the points, particularly those generated at lower network throughputs (for which there is less variance in performance), the rectangles representing the confidence intervals are too small to be seen.

8.5.4. Benchmark I: Static Hot Spot, Serial Requests

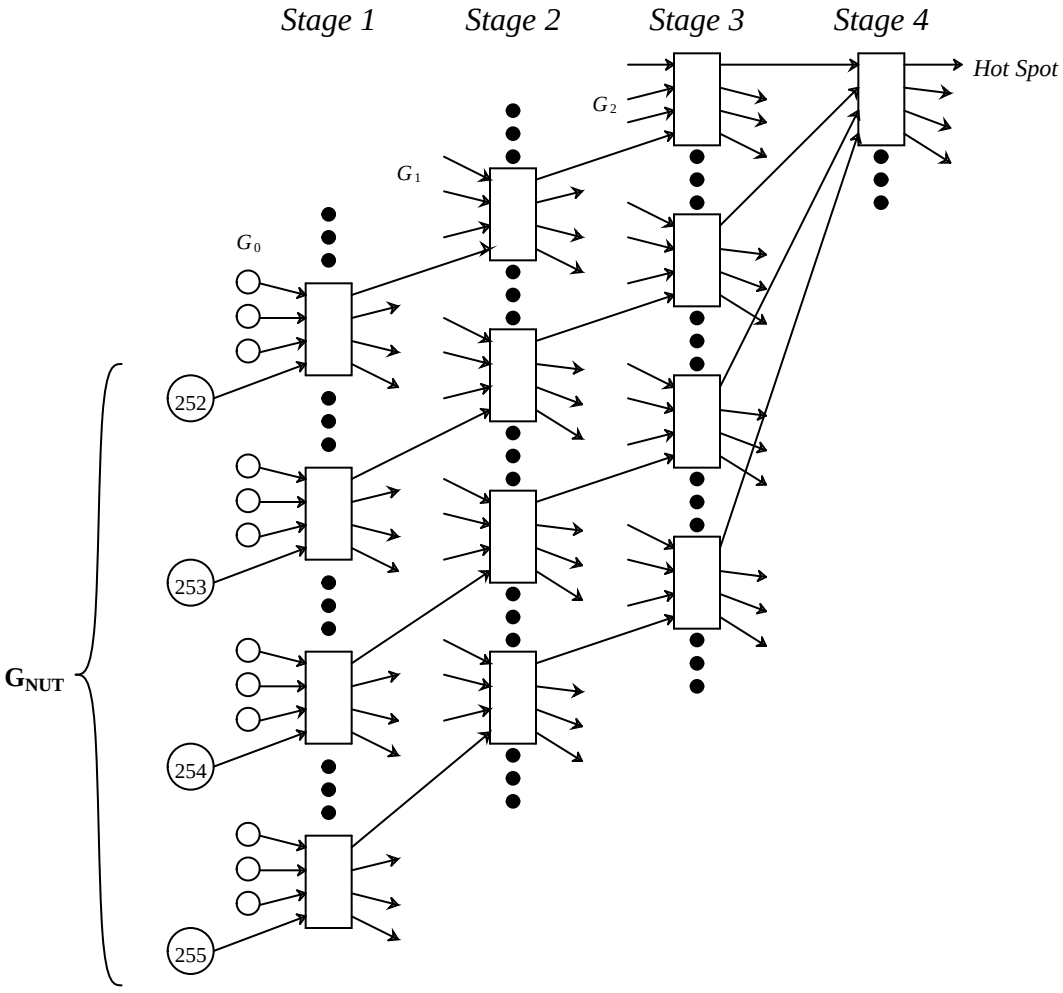


Figure 8.6: *Benchmark I: Hot Spot, Serial Requests.* G_{NUT} (<252>, <253>, <254>, <255>), transmitting all packets to the ''hot'' destination, creating a point of congestion at *output port* #0 of *switch* #0 of the fourth switching stage.

In this benchmark, the members of G_{NUT} transmit thirty-two byte packets to a single destination. There are four senders in G_{NUT} — their addresses (*senderIDs*) are <240>, <244>, <248> and <252>. These particular senders were chosen so that the path from each member of G_{NUT} to the hot spot would not intersect with that of

any other other member of G_{NUT} until the final link to the hot destination; thus there is no contention between the members of G_{NUT} prior to the hot spot itself. The senders of G_{NUT} transmit 100% of their packets to the hot spot at saturation throughput.

Name	Function	Explanation
G_{NUT}	*senderID* & 0xFC = 0xFC	Four senders whose paths intersect at final switching stage.
G_0	*senderID* & 0x3C = 0x3C	Senders whose paths intersect G_{NUT} at first switching stage (i.e. they share a first stage switch with a member of G_{NUT}).
G_1	*senderID* & 0x0C = 0x0C	Senders whose paths intersect G_{NUT} at the second switching stage.
G_2	TRUE	All remaining switches — their paths intersect with the paths of a member of G_{NUT} at the third switching stage.

Table 8.4: *Group Definition, Benchmark I.* The functions listed use the routing characteristics of the omega network to determine which senders are members of which sender groups. The parameter *senderID* is the source address.

The remaining 252 sending nodes are divided into three groups, according to the degree to which they interact with G_{NUT} (Fig. 8.6). G_0 comprises the twelve senders which share a first-stage switch with a member of G_{NUT}. G_1 is the forty-eight senders which can reach the same second-stage switches as G_{NUT} (and G_0). Finally, G_2 is the 192 senders (all remaining) which intersect G_{NUT} (and $G_{0,1}$) at the third switching stage. Tab. 8.4 demonstrates how these groups were defined in the simulator using bit operations on the *senderIDs*. These three groups generated uniformly-distributed traffic with the same applied load as each other. This applied

load is graduated from 0.0 m/sc to saturation.

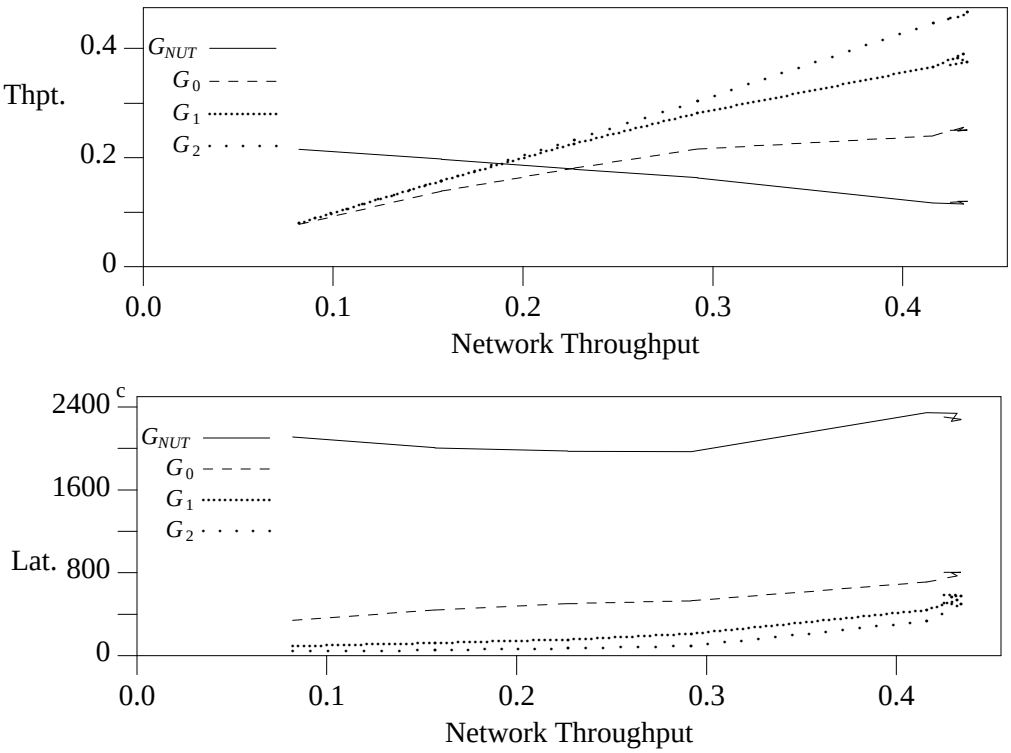


Figure 8.7: *Benchmark I: FIFO Buffers, Blocking Flow Control.* Hot spot traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0–2}$). Switches with FIFO buffers, blocking flow control. Throughput and average latency vs. total network throughput, G_{NUT} and $G_{0–2}$.

Figure 8.7 shows the throughput of each sender group vs. the total network throughput and the average latency vs. total network throughput, for a network of FIFO switches using blocking flow control. The throughput of a sender group is its average per-sender throughput. The lowest network throughput occurs when the applied load for groups $G_{0–2}$ is 0.0 m/sc, the highest when all groups are at saturation throughput. The throughput of G_{NUT} drops as the applied load of the

other three groups is increased because of the increasing contention in the network. At low applied loads, the throughput of G_{0-2} is proportional to the applied load. As the applied load (and, hence, throughput) is increased, G_0 is the first group impacted by the G_{NUT} traffic; at saturation, its throughput barely exceeds 0.20. The congestion caused by G_{NUT} has only minor impact upon G_2 — a network of FIFO switches under uniformly distributed traffic achieves 0.49, compared to 0.45 of G_2 in this benchmark.

This graph presents two of the performance objectives of interest. The first is the throughput degradation experienced by the sender groups G_{0-2}. In particular, G_0 achieves a saturation throughput which is a fraction of that achievable under uniform traffic (Fig. 8.5, FIFO data). We are looking for a flow control mechanism which will reduce the impact of congestion traffic on senders not participating in the congestion. The second performance objective is to maintain the throughput of data to the point of congestion (the hot spot, in this benchmark). When the applied load of $G_{0-2} = 0.0\ m/sc$, G_{NUT} operates a throughput of 0.235, which implies that the throughput of data to the hot spot is 0.94; since our transmission protocol requires that each link be idle for two clock cycles between each packet, with thirty-two byte packets the maximum throughput across a link is 0.941. When G_{0-2} are also operating at saturation throughput, the total throughput of data to the hot spot decreases to 0.893.

Figure 8.8 displays the same information as Fig. 8.7, but for DAMQ buffers with blocking flow control. A quick comparison of the two shows that the addition of non-FIFO buffering provides a significant boost in network performance. There is still some reduction of the performance of G_{0-2} due to the congestion caused by

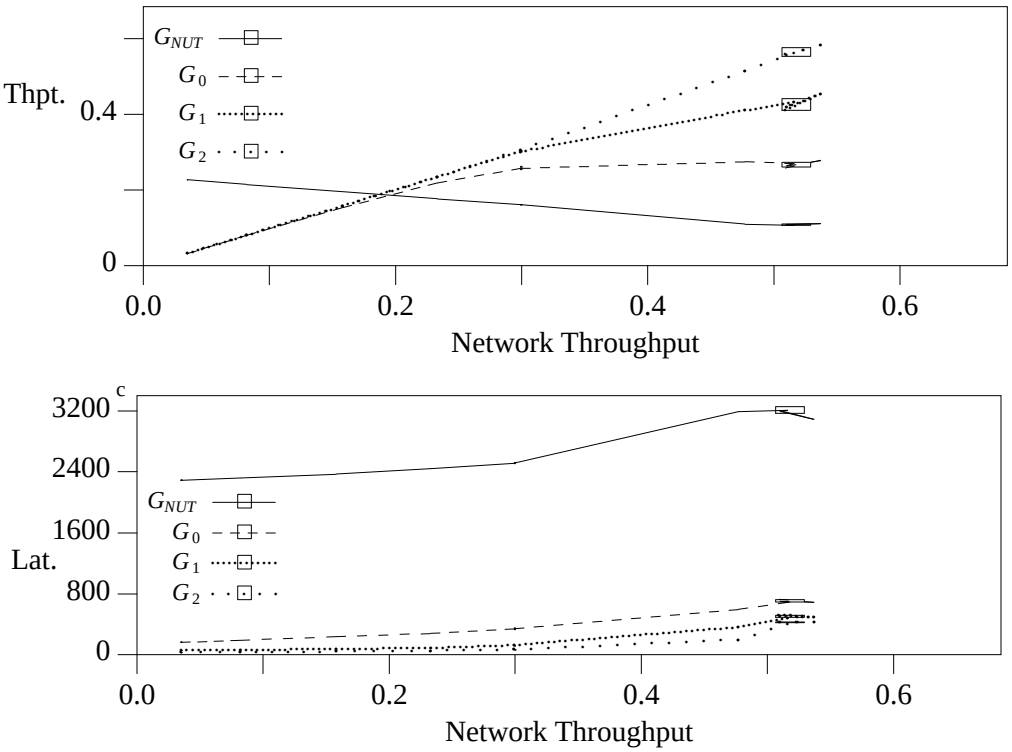


Figure 8.8: *Benchmark I: DAMQ Buffers, Blocking Flow Control.* Hot spot traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0–2}$). Switches with DAMQ buffers, blocking flow control. Throughput and average latency vs. total network throughput, G_{NUT} and $G_{0–2}$.

G_{NUT}. Under uniform traffic conditions, a network of DAMQ switches with blocking flow control can achieve a throughput of 0.71; in this benchmark, G_2 is limited to 0.58, while G_0 only reaches 0.28. Further, the latency of G_0 is extraordinarily high. Even at low throughputs, where average latency of G_2 is 39 c (the minimum latency for four hops is 20 c), G_0 experiences an average latencies of $\approx$ 200 c for DAMQ buffered networks, 450 c with FIFO buffers. Transmission latencies of this order could significantly degrade the performance of fine-grained

distributed programs.

While the saturation throughput of the network of DAMQ-buffered switches is higher than that of the network of FIFO-buffered switches, G_0 does not achieve significantly higher throughputs with DAMQ buffers than with FIFO. The throughput of the members of G_0 is determined by the available bandwidth of the output port of the switches in the first switching stage which leads to the hot spot, and the ability of packets from G_0 to compete with the hot spot packets for that bandwidth. DAMQ buffers do not and cannot help to resolve this — once the buffers of the first-stage switches connected to the members of G_0 become filled with packets destined for the congested output port, their throughput drops nearly to that of switches with FIFO buffers. The average latency of packets in the DAMQ buffer network is much lower, because packets routed to the other output ports are not queued behind the packets in the hot output port queue. The saturation throughput of the DAMQ buffer network is higher, because the back-pressure from the succeeding switch stages is lower than it is in the FIFO buffer network — the DAMQ buffer network maintains a throughput to the hot spot of 0.94.

Figure 8.9 shows the G_0 throughput vs. network throughput and G_0 latency vs. network throughput for switches with DAMQ buffers with blocking flow control, static partitioning (SAMQ and SAFC buffering), and maximum usage flow control with blocking thresholds of six and ten buffer blocks. This graph shows a significant improvement in G_0 throughput for both static allocation and maximum usage flow control mechanisms over blocking flow control (0.46 vs. 0.28). The throughput is still significantly lower than the throughput under uniform traffic,

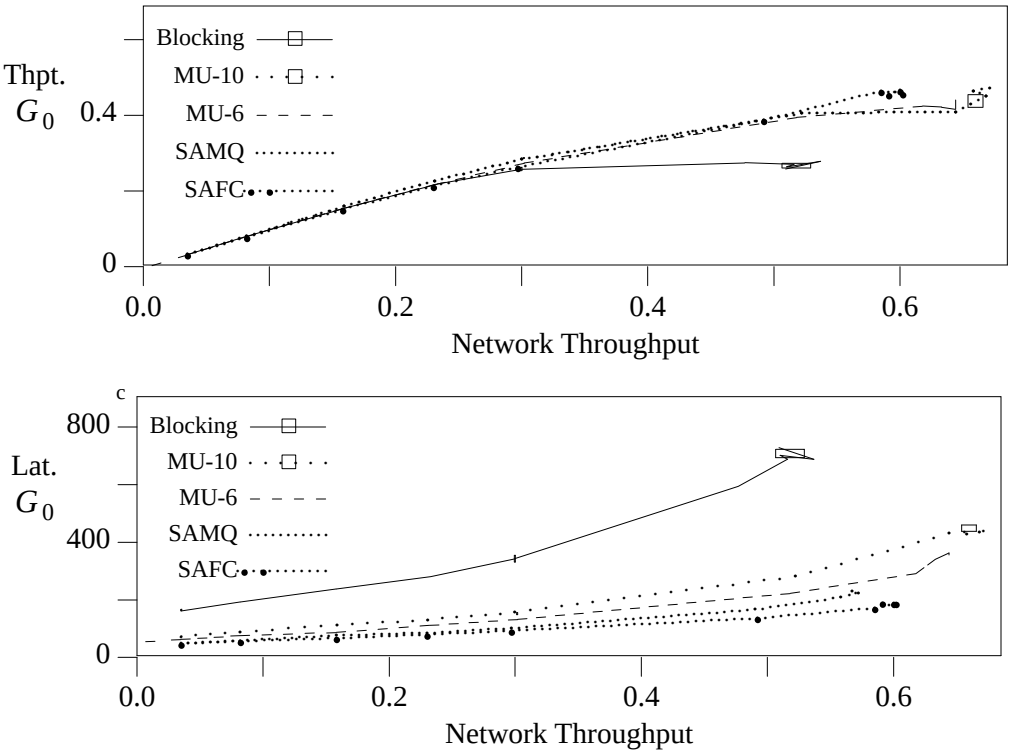


Figure 8.9: *Benchmark I: Maximum-Usage Flow Control.* Hot spot traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0–2}$). Shown is the G_0 throughput vs. network throughput and G_0 latency vs. network throughput for blocking flow control, static buffer allocation and maximum-usage flow control (blocking thresholds of six and ten buffer blocks).

however. The positive results achieved by maximum usage flow control under this benchmark are due to the fact the packets transmitted by the senders of G_0 conflicted with those from G_{NUT} at the output ports of the first switching stage. The flow control prevents the buffers in the second switching stage from becoming completely filled with packets destined to the hot spot, dramatically improving the performance of the G_0 senders. The heightened activity of the G_0 senders, in turn,

increases the contention experienced by G_{NUT}, reducing its throughput and the network contention it causes. Thus, under this benchmark, the ‘‘hot’’ output port of the first-stage switches which are connected to G_0 and G_{NUT} senders are *the* critical points for flow control decisions.

Static buffer partitioning provides a lower G_0 average latency than does maximum usage flow control and a similar saturation throughput for G_0. However, it does not achieve as high a network saturation throughput. G_0 benefits not from the superior flow control properties of static buffer allocation, but from the fact that it causes the buffer space to be used less efficiently than does maximum usage flow control. The throughput of the G_1 and G_2 traffic is unduly throttled by this inefficient buffer utilization; while the throughput of G_0 increases due to the resulting reduction in network contention, the throughput of the network as a whole is reduced.

Fig. 8.10 shows latency vs. throughput for networks using two-counter (hysteresis) flow control, with a variety of parameter settings (these settings are presented in Sec. 8.4.3 and Tab. 8.2). Comparing these results to each other and to those in Fig. 8.9, it can be seen that (a) the parameter settings for two-counter flow control make almost no difference in the performance of the network in this benchmark and (b) the hysteresis in two-counter flow control provides no improvement in performance over maximum usage flow control. This may be due to the static nature of the congestion in this benchmark.

Figure 8.11 shows the throughput vs. network throughput and average latency vs. network throughput results of G_0 for destination- and path-based flow control. Destination- and path-based flow control both achieve a higher network throughput

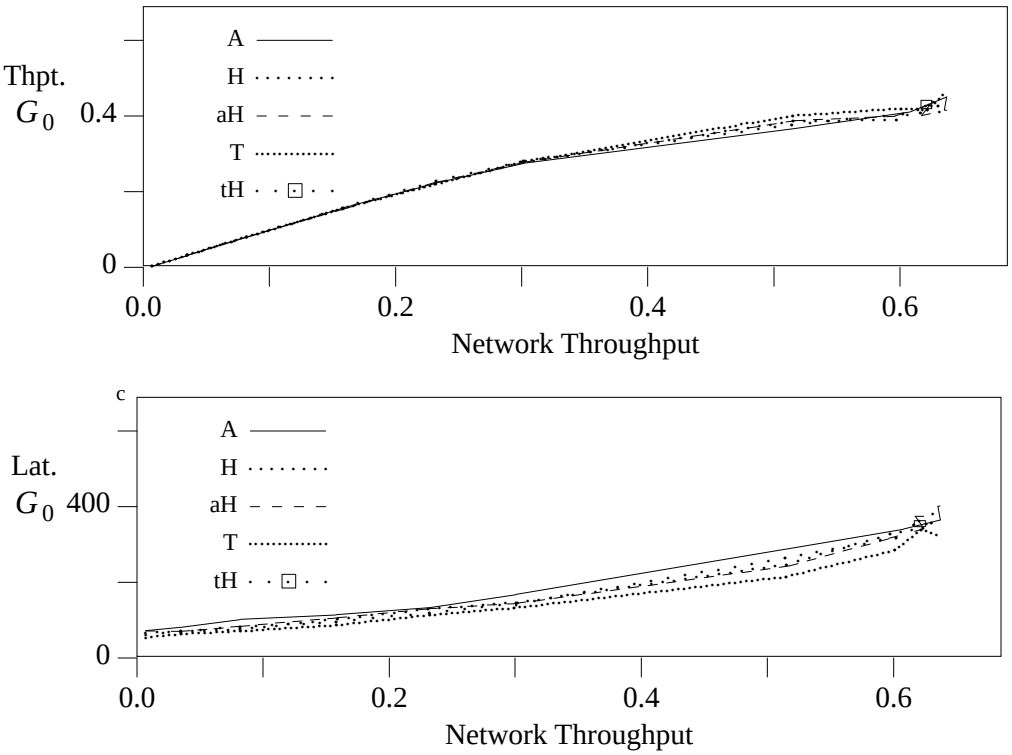


Figure 8.10: *Benchmark I: Two-Counter Flow Control.* Hot spot traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0–2}$). Shown is the throughput vs. network throughput and latency vs. network throughput of G_0 for two-counter flow control with a variety of parameters (see Tab. 8.2 for details of the parameters).

and a higher G_0 throughput than do the other flow control mechanisms. Destination-based flow control achieves the best results for G_0, saturating at a throughput of 0.80 for G_0. This throughput is higher than the saturation throughput of a blocking DAMQ buffer network under uniform traffic conditions. Destination-based flow control achieves this because the back-pressure from congestion in this flow control scheme is strongly directed — the *only* packets from G_0 which conflict with the G_{NUT} traffic are those addressed to the hot

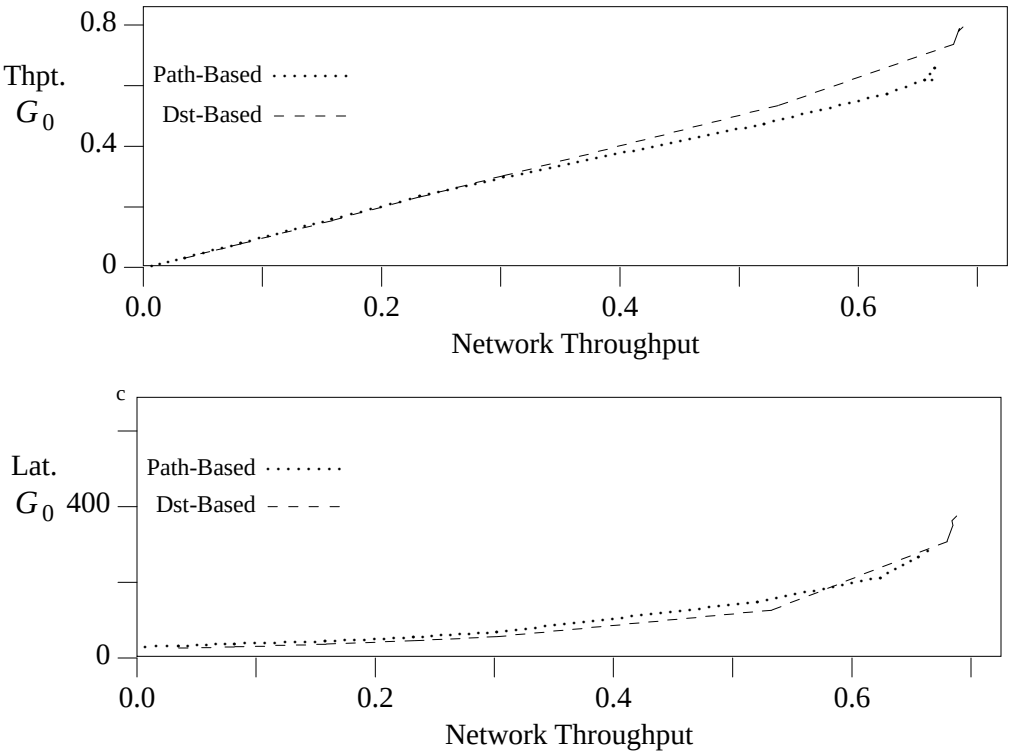


Figure 8.11: *Benchmark I: Link-Based Flow Control.* Hot spot traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0–2}$). Shown is the G_0 throughput vs. network throughput and G_0 latency vs. network throughput for destination-based and path-based.

destination. Since destination-based flow control severely restricts the throughput of G_{NUT}, G_0 experiences a situation where the first stage switches have four output ports but only three active input ports (the senders of G_{NUT} do not contend in a meaningful way for the output ports of the first stage switches). Thus, G_0 experiences less contention in this network than do the senders of a network transmitting uniformly-distributed traffic.

Under destination-based flow control, when all senders are transmitting at

saturation throughput, G_{NUT} is reduced to a throughput of only 0.06. This is as compared to 0.10 for blocking flow control and 0.08 for maximum usage flow control, threshold of ten blocks. Despite the tight reigns on G_{NUT}, however, the throughput to the hot spot from all senders, when all senders are operating at saturation, is 0.93 for destination-based flow control, due to the high throughput of the other sender groups.

Under this benchmark, the four senders which comprise G_{NUT} create a point of congestion associated with a terminal link in the network (a hot spot). This congestion can be characterized as being static and associated with a destination node. These are ideal conditions for destination-based flow control, which is explicitly designed to handle hot spot conditions. Path-based flow control does well, but by comparing the next two hops of the packets to determine collisions, it mistakenly categorizes some ‘‘innocent’’ packets as contributing to the congestion, and thus throttles G_{0-3} traffic unnecessarily. The static nature of the congestion completely negates the effects of hysteresis — its performance is similar to maximum usage flow control. Maximum usage and static allocation flow controls are both improvements over blocking flow control, due to their ability to moderate the congestion in the first two switching stages of the network.

8.5.5. Benchmark II: Static Hot Spot, Many Participants

The previous benchmark examined network performance in the presence of a static hot spot caused by a small number of senders (four of the 256) sending 100% of their traffic to a single ‘‘hot’’ destination at saturation throughput. In this benchmark, a static hot spot is caused by directing half of the senders (G_{NUT}) to

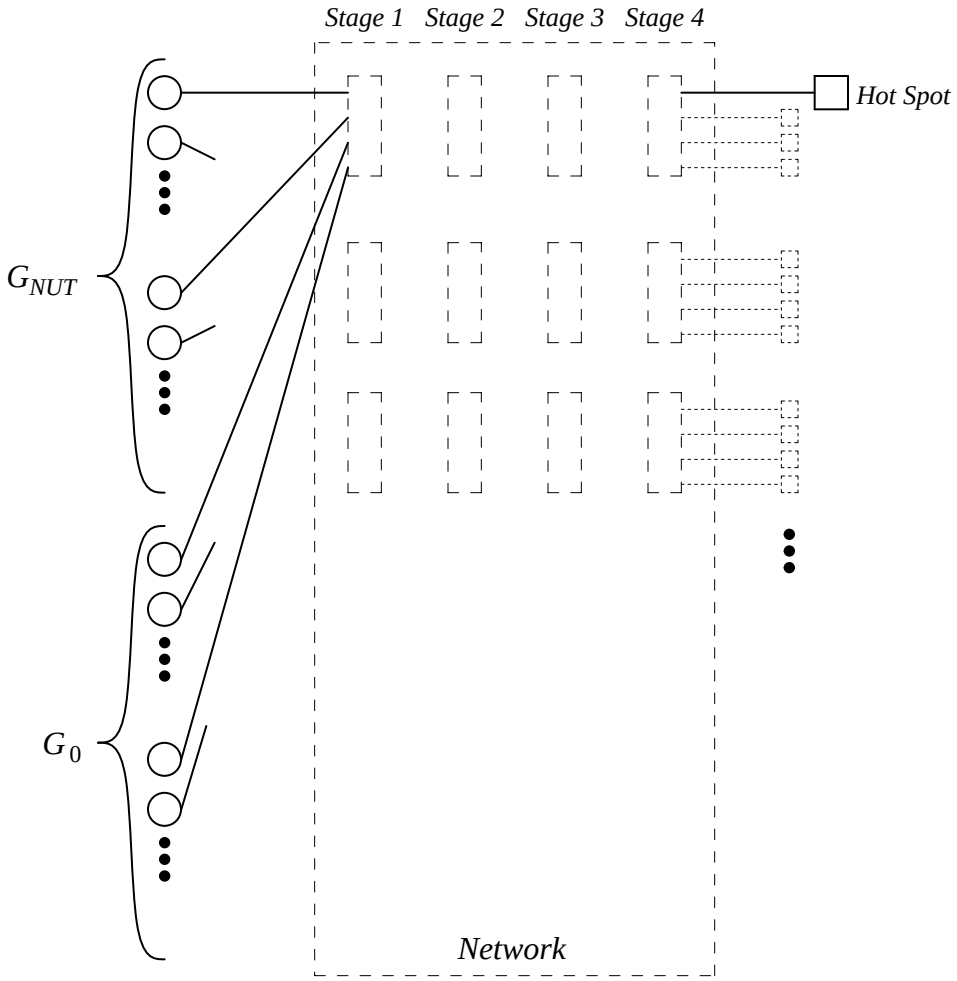


Figure 8.12: *Benchmark II: Static Hot Spot, Barrier Traffic.* Half of senders (G_{NUT}) operating at saturation, assigning uniformly-distributed destinations to 97% of packets generated, with the other 3% being sent to the hot spot. Other half of senders (G_0) transmit uniformly distributed traffic with graduated applied load. G_{NUT} and G_0 ''mix'' in first stage of omega (perfect shuffle) network.

transmit 3% of their packets to the hot destination, with the destinations of the other 97% being chosen from a uniform distribution of all possible destinations. Senders <0-127> comprise G_{NUT}, and senders <128-255> comprise G_0 (Fig. 8.12). Due to the perfect shuffle topology of the omega network, these two groups of

senders ‘‘mix’’ at the first switching stage — every switch in the first switching stage is connected to two members of G_{NUT} and two members of G_0. All messages are thirty-two bytes long, and G_{NUT} transmits at saturation throughput. Since there are 128 senders in G_{NUT}, each sending 3% of their packets to the hot spot, and the bandwidth to the hot spot is 0.94, an upper bound for the throughput for G_{NUT} (T_{NUT}) can be found by the equation

$$0.94 \geq 0.97 \cdot (128/256) \cdot T_{NUT} + 0.03 \cdot 128 \cdot T_{NUT}.$$

The saturation throughput of G_{NUT} is less than or equal to 0.23.

There are three significant differences between the setup of this and the previous benchmark. The first difference is that contention for the hot spot occurs in all stages of the network in this benchmark. In the previous benchmark, G_0 absorbed the brunt of the performance penalty in the first switching stage. Second, all senders which are not members of G_{NUT} experience the same amount of contention, hence the single group of senders transmitting uniformly-distributed traffic (G_0). Third, G_{NUT} transmits packets to destinations other than the hot spot, and thus competes for some of the bandwidth of links not on the path to the hot spot.

Figure 8.13 shows the G_{NUT} and G_0 results for networks composed of FIFO and DAMQ switches with blocking flow control. The fact that throughput results for FIFO and DAMQ buffers are nearly identical highlights the fact that non-FIFO buffering does not prevent congestion propagation. The G_0 average latency is significantly lower for the network composed of DAMQ buffered switches than it is for the FIFO network. Multi-queue buffers prevent packets which are not

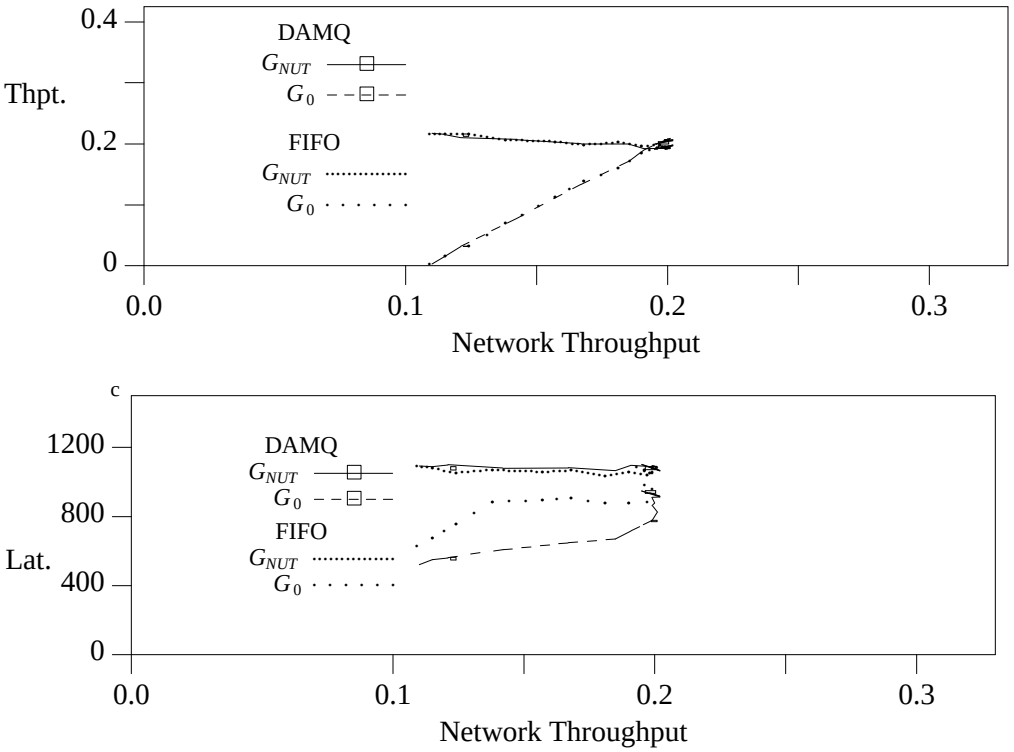


Figure 8.13: *Benchmark II: FIFO and DAMQ Buffers, Blocking Flow Control.* Hot spot traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_0). FIFO and DAMQ Switches, blocking flow control. Throughput and average latency vs. total network throughput, G_{NUT} and G_0.

contending for the hot output port of a switch from being queued behind packets which are contending for the hot output port.

These results suggest three areas in which network performance could be improved under the congestion pattern of this benchmark: (a) the maximum throughput of G_0 can be increased (ideally to 0.71), (b) total network throughput can be increased and (c) average latency for G_0 can be reduced. One area which will not be significantly improved is hot spot utilization — if $T_{NUT} = T_0$, then the

point of congestion is fully utilized when the throughput for G_{NUT} and G_0 is 0.20, which is achieved by blocking flow control.

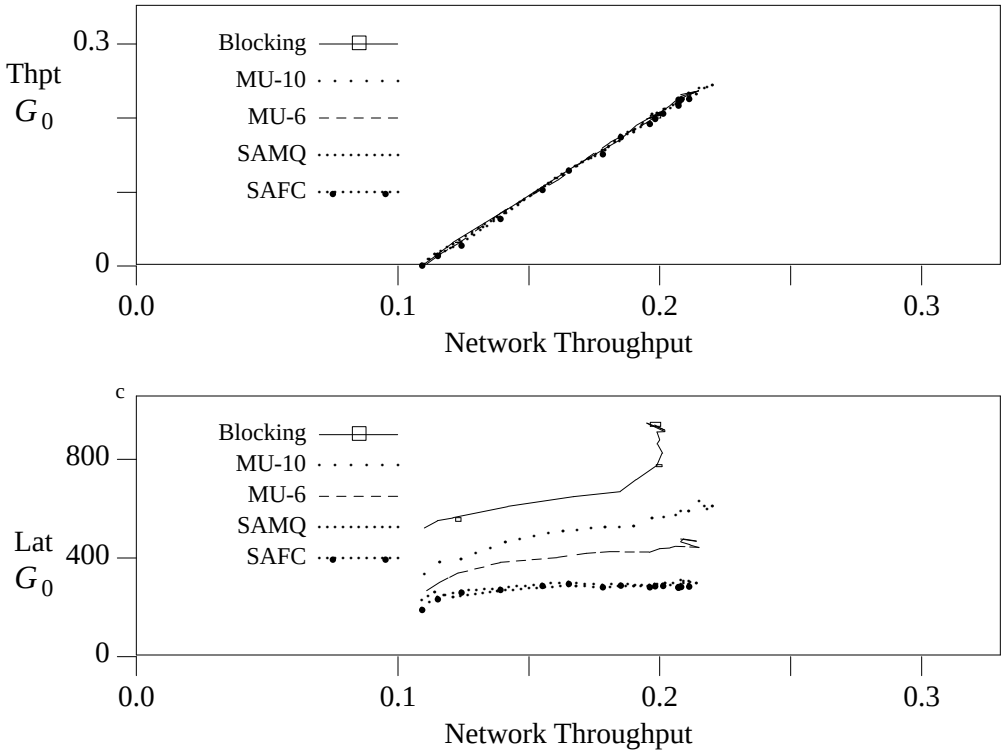


Figure 8.14: *Benchmark II: Maximum-Usage Flow Control, Group 0.* Hot spot traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_0). DAMQ switches with blocking flow control, maximum usage flow control, and statically allocated buffers. Throughput and average latency vs. total network throughput, G_0.

Figure 8.14 shows the performance of the G_0 traffic for a network of DAMQ switches with maximum usage flow control with thresholds of ten and six buffer blocks, and for a network of multiqueue buffers with a static buffer allocation (SAMQ and SAFC buffers). In the first benchmark, we noted that these flow control mechanisms caused a measurable increase in the saturation throughput of

G_0 and significantly reduced its average latency. That is not the case for the current traffic pattern; while these flow control mechanisms do still reduce the latency, there is little improvement to the saturation throughput of G_0 or of the network as a whole.

In Benchmark I, the packets transmitted by the senders of G_0 conflicted with those from G_{NUT} at the first switching stage. At this point, because there were only four senders in G_{NUT}, the congestion traffic was ‘‘diluted’’, making the first switching stage *the* critical point for flow control decisions. In this benchmark, on the other hand, half of the senders are members of G_{NUT}, and there are points of congestion throughout the network, including at the final switching stage. Since the congestion is occurring at a distance from the senders and maximum usage flow control directs its back-pressure for only two hops, this traffic pattern results in tree saturation.

Figure 8.15 presents G_0 performance figures for two-counter (hysteresis) flow control. Results are given for a variety of two-counter parameter values. The behavior of each of the parameter assignments is presented in Tab. 8.2. As can be seen from Fig. 8.15, these behaviors all result in the same network performance. This stems from the fact that Benchmark II is a static congestion pattern. Hysteresis is a mechanism for differentiating between short-term variance and long-term changes in the value of a random function. Since this benchmark, like Benchmark I, creates a static congestion pattern, there are few dynamics upon which hysteresis can operate.

Figure 8.16† shows the G_0 throughput and latency vs. total network

† Note the change in scale between the graphs in Fig. 8.16 and the graphs in the previous figures in this section.

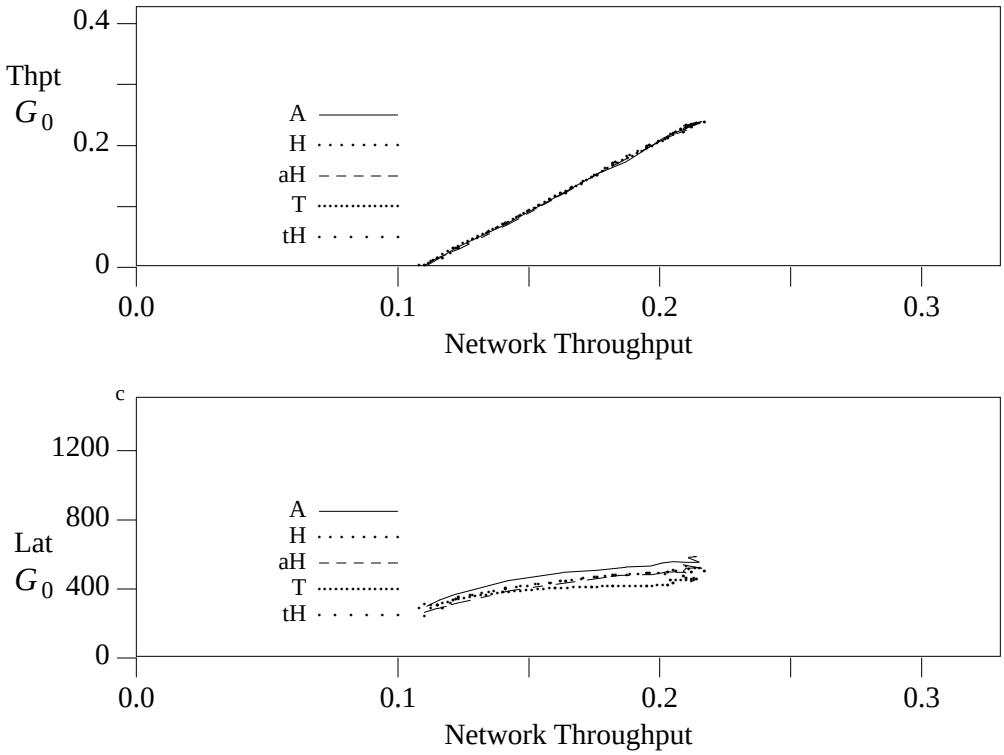


Figure 8.15: *Benchmark II: Two-Counter Flow Control, Group 0.* Hot spot traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_0). Two-counter flow control, variety of parameters (Tab. 8.2). Throughput and average latency vs. total network throughput, G_0.

throughput for networks using destination-based flow control and path-based flow control. In the group throughput vs. network throughput graph, the destination-based and path-based flow control throughputs are identical up to a network throughput of 0.26. At that point, the network using path-based flow control saturates.

A result which surprised us is the degree to which destination-based flow control out-performed path-based flow control. Not only did destination-based

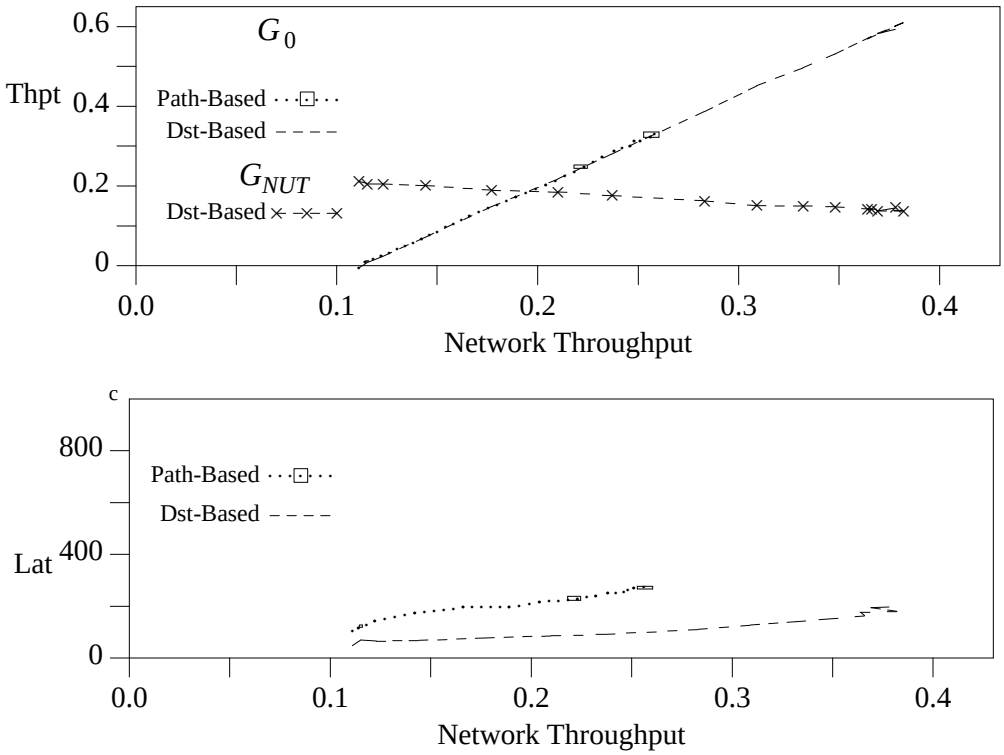


Figure 8.16: *Benchmark II: Link-Based Flow Control, Group 0.* Hot spot traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_0). Shown is the G_0 throughput vs. network throughput and G_0 latency vs. network throughput for destination-based and path-based flow control. In the top graph, G_0 throughputs for destination-based and path-based flow controls are equal up to a network throughput of 0.26.

flow control support a G_0 saturation throughput *two times greater* than path-based flow control, but the average latency of G_0 traffic under destination-based flow control *at saturation throughput* (0.3 +) was lower than the average G_0 latency under path-based flow control *at throughputs approaching 0.2.* Given such a tremendous difference between the performance of G_0 traffic under destination-based flow control and any other flow control mechanism, one must ascertain

(a) that the total network throughput was not sacrificed to provide this level of G_0 performance and (b) that the point of congestion was not under-utilized.

Figure 8.16 shows that the total network throughput under destination-based flow control was also significantly higher than under the other flow control mechanisms. The throughput through the point of congestion (T_{PC}) when the network is operating at its maximum throughput can be calculated from the throughputs of G_0 and G_{NUT} at that point (0.60 and 0.15, respectively).

$$T_{PC} = \frac{128}{256} \cdot 0.60 + \frac{128}{256} \cdot 0.97 \cdot 0.15 + 128 \cdot 0.03 \cdot 0.15$$

$$= 0.94.$$

Thus, for this benchmark, the hot link is fully utilized under destination-based flow control.

Benchmark II presented markedly different results than did Benchmark I. The most significant difference was the ineffectiveness of all of the flow control mechanisms except destination-based flow control in this benchmark. Even path-based flow control, which performed comparably to destination-based flow control under Benchmark I, and which outperformed all other flow control mechanisms under this benchmark, fell significantly short of destination-based flow control. The reason for this was two-fold. First, the congestion was destination-oriented (a hot spot). Since destination-based flow control keys on packets' addresses to detect congestion, it has a clear advantage in terms of being able to detect the congestion in this benchmark. Second, the point of congestion was four hops away from the senders. For flow control mechanisms such as maximum usage,

hysteresis and static buffer allocation, which only provide directed flow control for a single hop, having the point of congestion be this far from the senders allows tree saturation to occur.

The difference between the performances of destination-based and path-based flow control stems from the fact that path-based flow control suffers from ‘‘false blocking’’. Since it considers multiple packets traversing the same next n hops as being congested (i.e. it does not allow two such packets in a single buffer memory), and, for our simulations, n was set to two, all packets whose paths shared two or more links with the hot spot traffic were blocked as through they *were* hot spot traffic. This significantly reduced path-based flow control's performance relative to destination-based flow control.

8.5.6. Benchmark III: Static NUT Spot

The first two benchmarks explored the utility of flow control mechanisms in situations where congestion was caused by contention for a terminal link of the network (i.e. a hot spot). Under those conditions, packets which are ‘‘involved’’ in the congestion can be identified by their destination address. Destination-based flow control does exactly that, and thus achieves higher performance for senders not participating in the hot spot (members of G_i). In this benchmark, G_{NUT} creates a non-uniform traffic spot internal to the network. This is done by directing senders <207>, <223>, <239> and <255> (G_{NUT}, for this benchmark) to transmit thirty-two byte messages to destinations randomly chosen from the range $(0 \cdots 15)$. The members of G_{NUT} were chosen such that their paths to any destination intersect at the second switching stage. By restricting the destinations

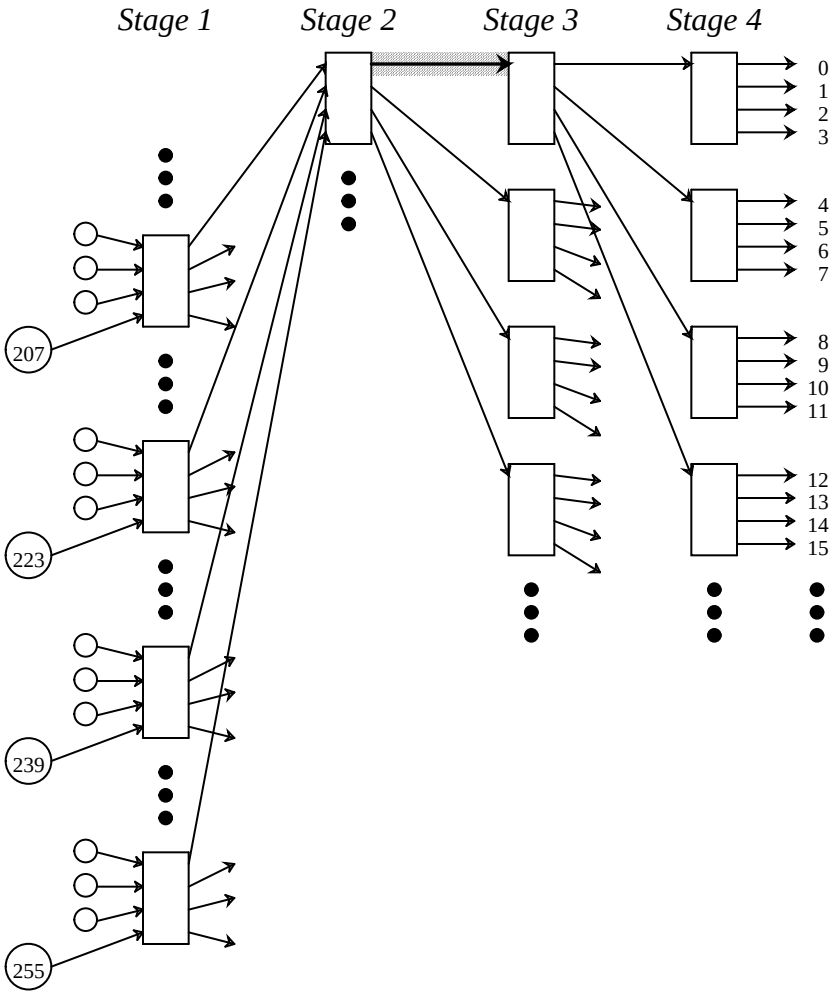


Figure 8.17: *Benchmark III: Non-Uniform Traffic Spot.* G_{NUT} (<207>, <223>, <239>, <255>), transmitting all packets to destinations $(0 \cdots 15)$, create a non-uniform traffic spot at *output port* #0 of *switch* #0 of the second switching stage.

of packets transmitted by G_{NUT} to the range $(0 \cdots 15)$, a non-uniform traffic spot is created at *output port* #0 of *switch* #0 of the second switching stage (Fig. 8.17). The NUT spot is traversed by packets addressed to destinations $(0 \cdots 15)$ from twelve other senders; these senders comprise G_0. The remaining senders (G_1) cannot reach the congested link, and are thus not negatively impacted by the

congestion (in fact, the performance of the remaining senders is improved, since the existence of the congestion will limit the throughput of the senders in G_{NUT} and G_0).

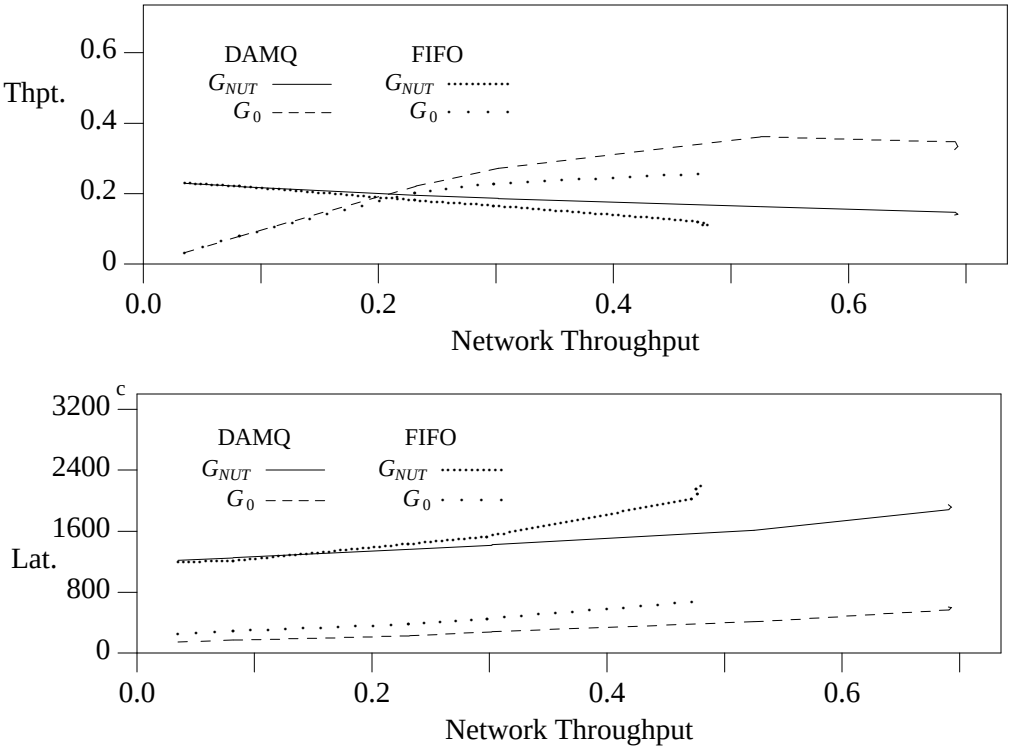


Figure 8.18: *Benchmark III: FIFO and DAMQ Buffers, Blocking Flow Control.* Congestion traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0,1}$). FIFO and DAMQ Switches, blocking flow control. Throughput and average latency vs. total network throughput, G_{NUT} and G_0.

Fig. 8.18 shows the results for networks of FIFO and DAMQ switches with blocking flow control (the results for G_1 are not shown, since G_1 does not interact with the congestion). As with Benchmark I, the point of highest throughput for G_{NUT} is when the applied load for $G_{0,1}$ is 0.0 m/sc — as the applied load for the

other groups is increased, G_{NUT} loses network bandwidth and experiences higher latencies. This graph shows G_0 severely impacted by the NUT spot; G_0's throughput saturates at 0.36 and experiences significant degradation as the applied load is increased beyond that point (i.e. as the applied load is increased beyond the point of maximum throughput for G_0, the throughput of G_0 drops). The throughput degradation is due to contention between G_0 and G_1. Since G_1 is not impacted by the NUT spot, it achieves a higher packet throughput, as evidenced by the high total network throughputs achieved by both buffer types. Packets from G_0 do, however, interact with packets from G_1 at the third and fourth switching stages. Thus, at top throughputs, G_1 uses significant fractions of the available bandwidth at those stages, reducing the throughput of G_0.

Figure 8.19 shows the performance of networks using maximum usage flow control mechanisms in comparison to blocking flow control. The graphs present the throughput and latency of G_0 vs. total network throughput, as this provides the key performance metrics (total network throughput and degree of protection offered G_0). The figure shows maximum usage flow control significantly improving the throughput of G_0. This degree of success is due to the proximity of the point of congestion to the senders. The point of congestion is a link connecting the second and third switching stages; it is thus two hops from the senders. Maximum usage flow control, which determines whether or not to transmit packets based upon the output port of the *next switch* that the packet will traverse, provides *directed back pressure* (Sec. 8.4) for only a single hop. It is, thus, much more effective in situations where the senders which are causing the congestion are a small number of hops away from the point of congestion. Under this benchmark,

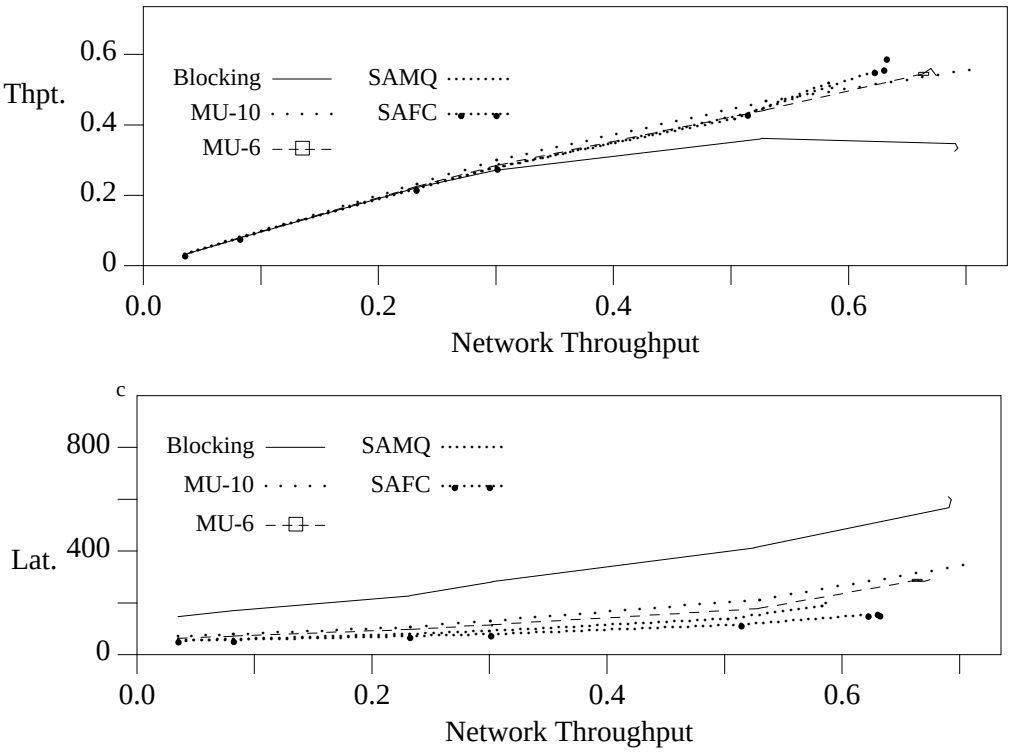


Figure 8.19: *Benchmark III: Maximum Usage Flow Control, Group 0.* Congestion traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0,1}$). Maximum-usage flow control with thresholds of ten and six buffer blocks and statically allocated buffers. DAMQ buffers with blocking flow control presented for comparison. Throughput and average latency vs. total network throughput, G_0.

switches in the first switching stage can distinguish between packets which are destined for the hot switch in the second stage and will traverse the congested link, and those destined for the hot switch but will not contest for the congested link.

Static buffer allocation and maximum usage flow control with a threshold of six buffer blocks, however, have significantly lower maximum network throughputs than does blocking flow control. This reflects blocking flow control's

superior performance under uniform traffic conditions (i.e. G_1 performance), as will be seen in Sec. 8.5.9. Maximum usage flow control with a threshold of ten blocks demonstrates clear performance advantages over the other mechanisms in the figure; it achieves the highest network throughput, the highest G_0 throughput *and* an average G_0 latency as low as any of the four flow control mechanisms shown.

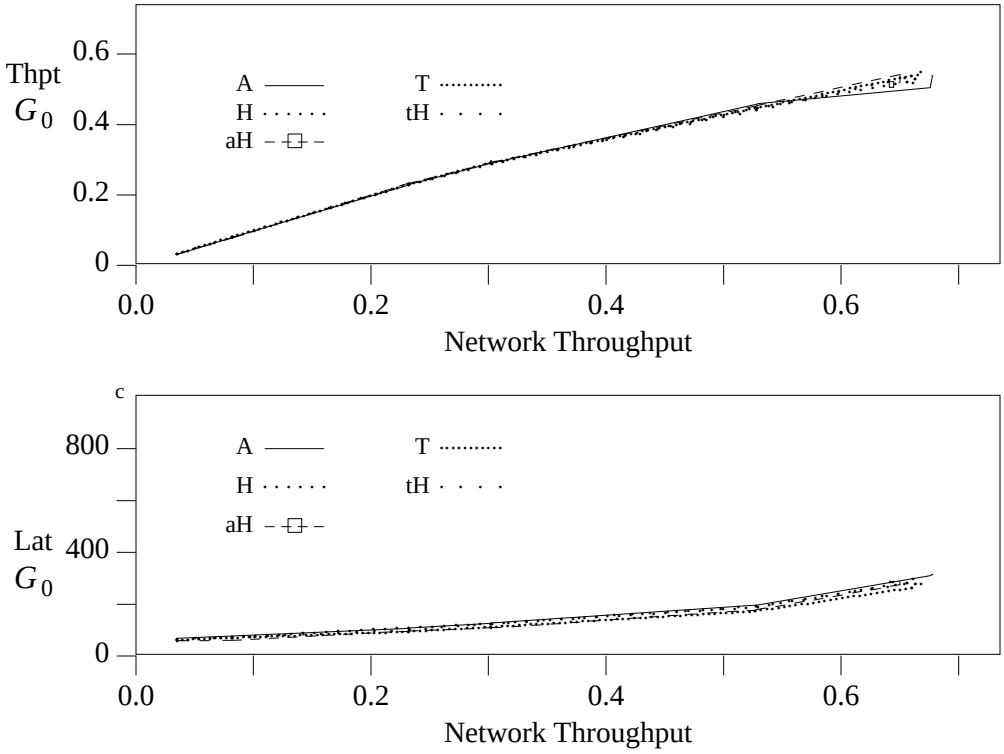


Figure 8.20: *Benchmark III: Two-Counter Flow Control, Group 0.* Congestion traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0,1}$). Two-counter flow control with a variety of parameters (Tab. 8.2) Throughput and average latency vs. total network throughput, G_0.

Figure 8.20 shows the performance for two-counter flow control with a

variety of parameters (see Tab. 8.2). This being a static congestion pattern, we did not expect hysteresis to positively impact the performance of the flow control mechanisms. As the graphs show, the two-counter flow control mechanism behavior is similar to maximum usage flow control behavior, with the maximum network throughput falling short of that achieved by maximum usage, threshold of ten blocks.

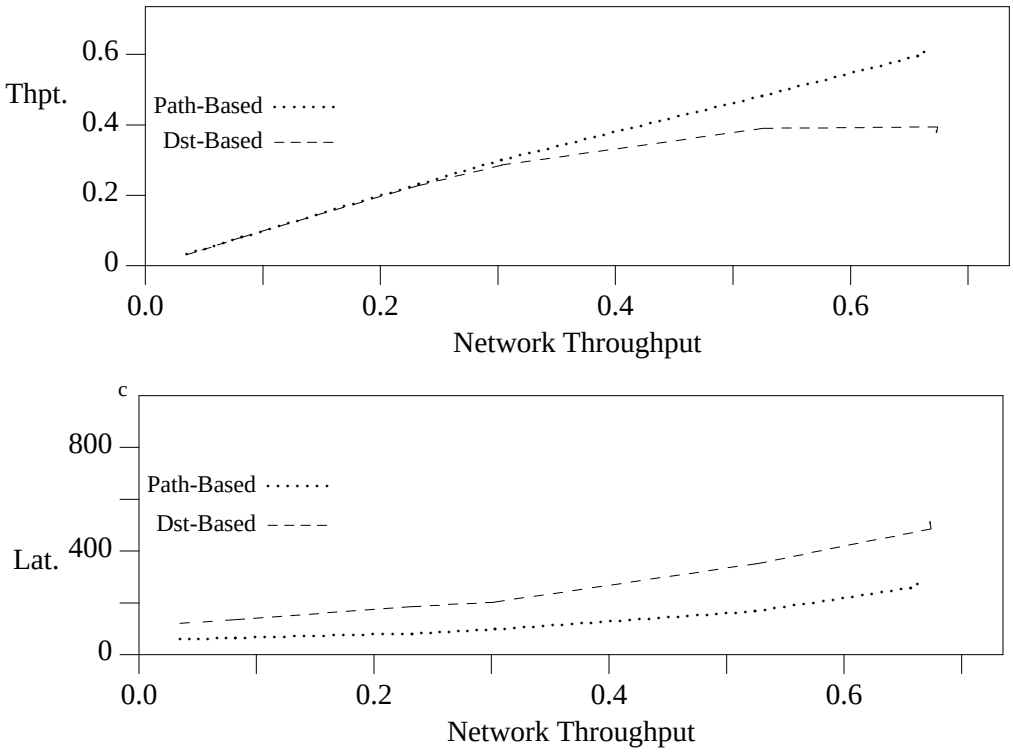


Figure 8.21: *Benchmark III: Link-Based Flow Control, Group 0.* Congestion traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0,1}$). Destination- and path-based flow control. Throughput and average latency vs. total network throughput, G_0.

Figure 8.21 shows the G_0 throughput and average latency vs. network

throughput curves for destination-based and path-based flow control. Destination-based flow control performance is similar to that of blocking flow control; it achieves a slightly higher throughput for G_0 and a slightly lower maximum network throughput. This is due to the fact that destination-based flow control bases its flow control decisions on the destination addresses of packets. Since the congestion is not correlated to individual destinations, the flow control is ‘‘blind’’ to the congestion. Path-based flow control performs as well in this benchmark as in Benchmarks I and II — that is to say, it provides good protection for G_0, but does suffer some network throughput degradation relative to other flow control mechanisms. This follows from the fact that path-based flow control detects link contention, independently of where the link is located in the network. Thus, the congestion occurring in this benchmark is indistinguishable from the congestion experienced in the first two, from path-based flow control’s viewpoint.

This benchmark provided insights into the behavior of each of the flow control mechanisms under investigation. It demonstrated the reliance of destination-based flow control on there being a correlation between congestion and a particular destination address. If such a correlation does not exist, then destination-based flow control will be ineffective. Path-based flow control remains effective, due to its use of packets’ paths as a basis for flow control decisions. It suffers, however, from the same fault as static buffer allocation; it too often throttles flow when throttling is unnecessary, which negatively impacts the total network throughput. Buffer-based flow control schemes such as maximum usage and two-counter flow control are effective in situations where the point of congestion is physically close to the cause of the congestion, as is the case with this

benchmark.

8.5.7. Benchmark IV: Four Senders with Long Messages

In this benchmark, the members of G_{NUT} transmit long (1024-byte) messages to randomly chosen destinations. These messages are broken into a sequence of thirty-two thirty-two byte packets — the packets which constitute a message are transmitted back-to-back at saturation throughput, and in this benchmark the messages themselves are generated with no idle time in between (saturation throughput). By attempting to utilize 100% of the available bandwidth on the path to their destination, each long message will cause congestion. There are four members of G_{NUT}: senders <240>, <244>, <248> and <252>. Due to the senders which were chosen to constitute G_{NUT}, two packets from G_{NUT} senders will contend for the output port of a switch only at the forth (final) switching stage. In other words, intra-G_{NUT} contention only occurs when two members of G_{NUT} transmit messages to the same destination in overlapping time periods. Thus, for the majority of the simulation time, there will be four independent heavily used paths through the network (one from each member of G_{NUT}).

The remaining senders are divided into three groups based upon their interaction with G_{NUT}. The twelve senders whose traffic intersects G_{NUT}'s traffic at the first switching stage are G_0, the forty-eight whose traffic intersects at the second stage are G_1, and the remaining 192 senders, whose traffic intersects G_{NUT}'s at the third switching stage, are G_2; this is identical to the sender grouping for Benchmark I. This benchmark is intended to test the flow control mechanisms in a dynamic environment, where the cause of congestion is transient. Also,

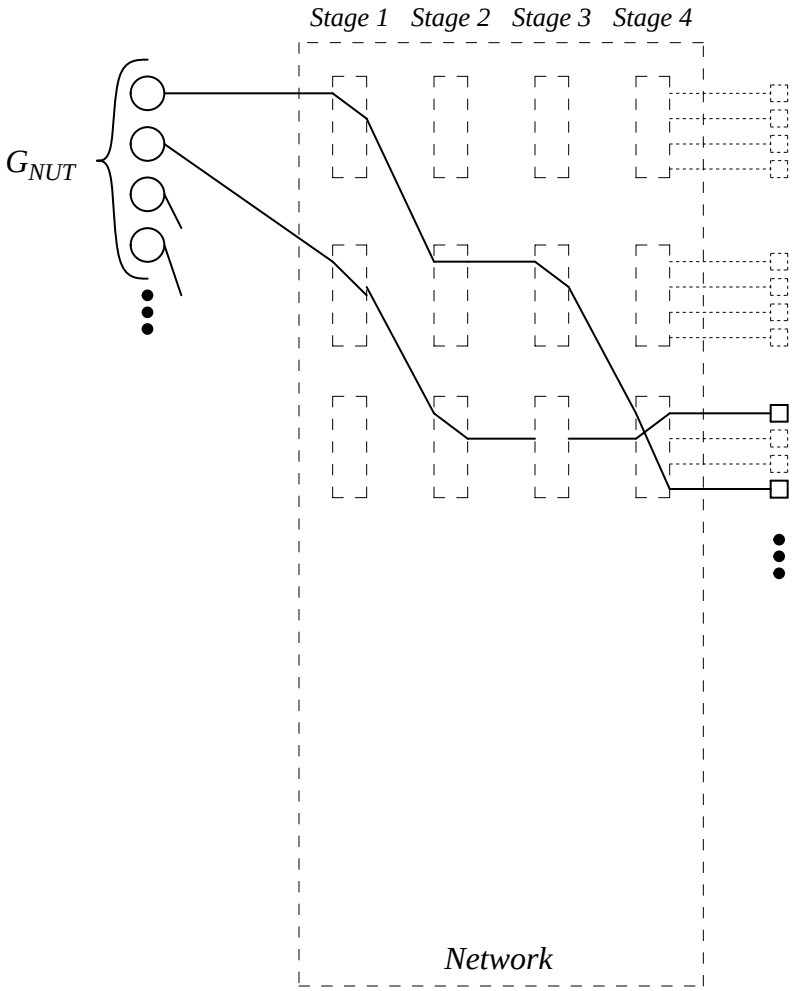


Figure 8.22: *Benchmark IV: Dynamic Serial Transmission.* Four senders (G_{NUT}) operating at saturation, assigning uniformly-distributed destinations to 1024 byte messages.

creating streams of packets from a source to a particular destination mimics events which occur in multicomputers such as remote paging, cache flushing or file I/O.

Figure 8.23 shows the throughput and the average latency of traffic from each of the four groups of senders for a network of DAMQ switches using blocking flow control. The results suggest that the congestion caused by this benchmark degrades the performance to the network as a whole far less than did the static hot or NUT

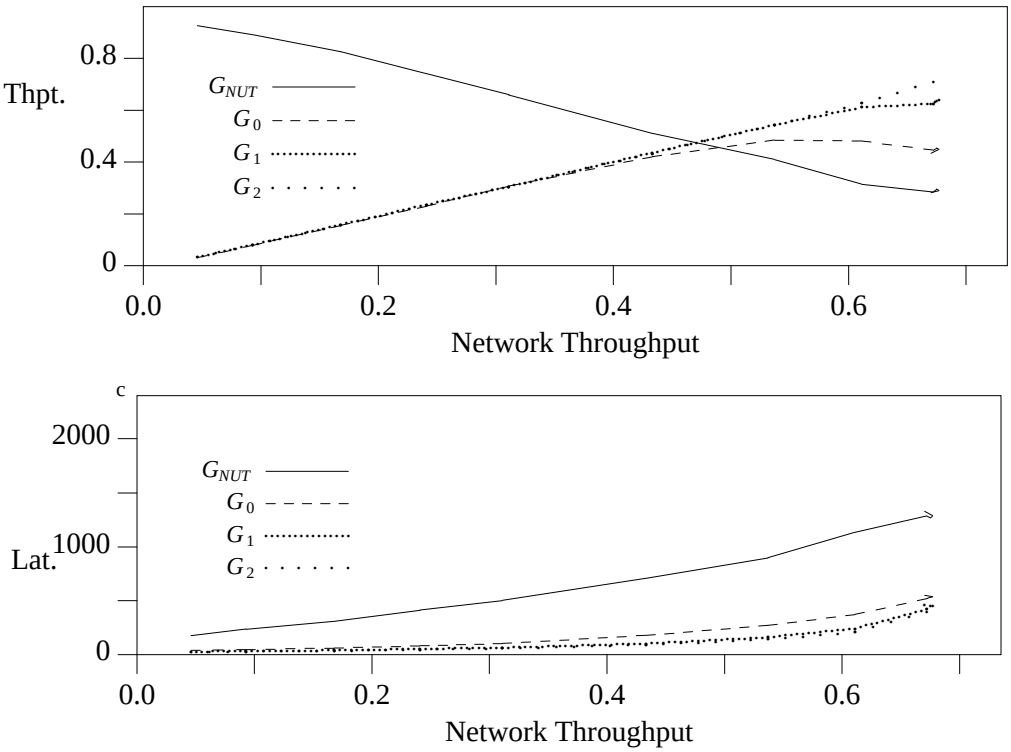


Figure 8.23: *Benchmark IV: DAMQ Buffers, Blocking Flow Control.* Serial traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0–2}$). Switches with DAMQ buffers, blocking flow control. Throughput and average latency vs. total network throughput, G_{NUT} and $G_{0–2}$.

spots of the previous benchmarks. The graphs show G_0 suffers the greatest impact, with its saturation throughput reduced to a maximum of 0.49, a forty percent reduction compared to the throughput of a system with a uniform traffic distribution. In addition, the point of maximum G_0 throughput does not correspond to the maximum network throughput; beyond the point at which the G_0 throughput peaks, contention from $G_{2,3}$ (which compete with $G_{0,1}$ for network resources only *after* the first switching stage) reduces the G_0 throughput.

G_1 is minimally impacted by G_{NUT}, with its throughput dropped to 0.64, and G_2 is not impacted at all, achieving a throughput of 0.71. The reason for this is that, while G_{NUT} reduces the throughput of G_0, G_0 does the same to G_{NUT}. Congestion at the first switching stage reduces the applied load of G_{NUT} traffic on the second switching stage, decreasing its impact on G_1. Similarly, contention between G_{NUT} and G_1 at the second switching stage reduces G_{NUT} traffic to the point that G_2's throughput is not effected.

Figure 8.24 compares the G_0 results of blocking flow control to results for maximum usage flow control (with thresholds of ten and six buffer blocks) and to static buffer allocation (SAMQ and SAFC buffering). Static buffer allocation and maximum usage flow control provide similar protection for the performance of G_0; the maximum throughput of G_0 is increased to ≈ 0.58 (slightly higher for SAFC buffers, slightly lower for SAMQ buffers and maximum usage flow control), and the average latency of G_0 is cut to nearly half of what it is for blocking flow control. By limiting the amount of buffer space which can be occupied by a single queue, these forms of flow control prevent the buffers of the second-stage switches from being filled with G_{NUT} packets, thus allowing the G_0 traffic which will take a different second-stage output port to bypass the congestion at that switching stage. This improvement is not enough, however, to offset the performance degradation caused by static buffer allocation under uniform traffic conditions; SAMQ and SAFC buffering support significantly lower maximum network throughputs under this benchmark than does DAMQ buffers with blocking flow control. Maximum usage flow control, on the other hand, increases the maximum throughputs of both G_0 and the network as a whole relative to blocking flow control; the impact of

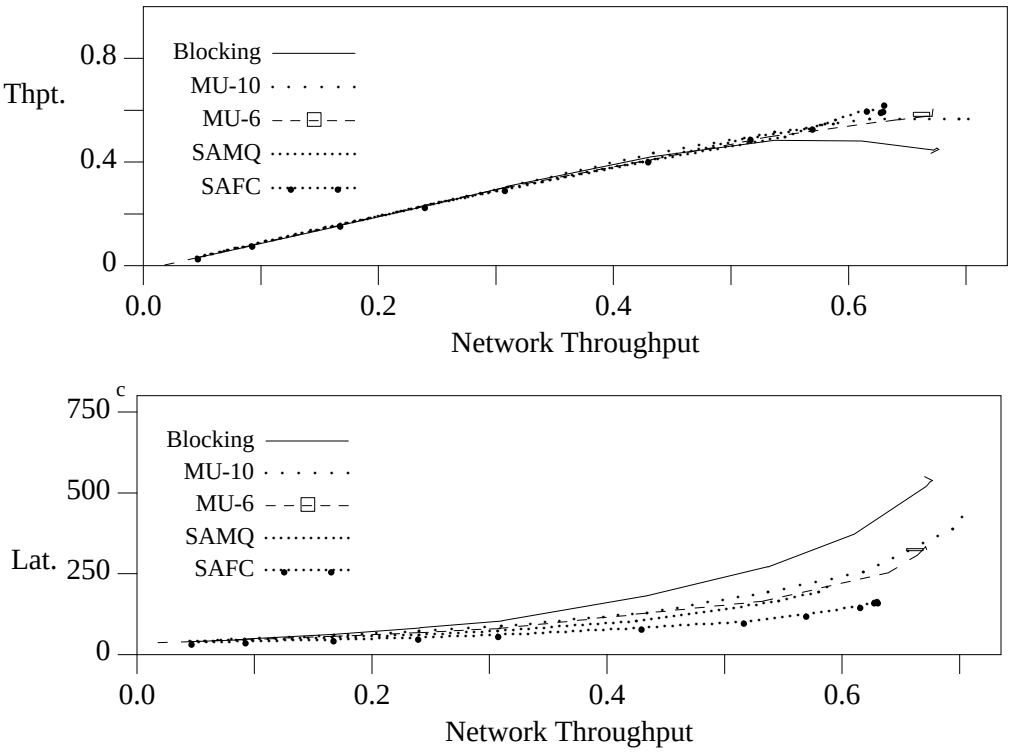


Figure 8.24: *Benchmark IV: DAMQ Buffers, Maximum-Usage Flow Control.* Serial traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_{0-2}). Blocking, maximum-usage flow control and statically allocated (SAMQ and SAFC) buffers. Throughput and average latency vs. total network throughput, G_0.

maximum usage flow control on uniformly-distributed traffic is presented in Sec. 8.5.9.

The performance of two-counter (hysteresis) flow control is shown in Fig. 8.25. Simulations were run for this benchmark using a variety of parameter values (Tab. 8.2). As was seen in the previous benchmarks, there is little difference in the degree to which two-counter flow control with the different parameter values promote the performance of the groups of senders transmitting

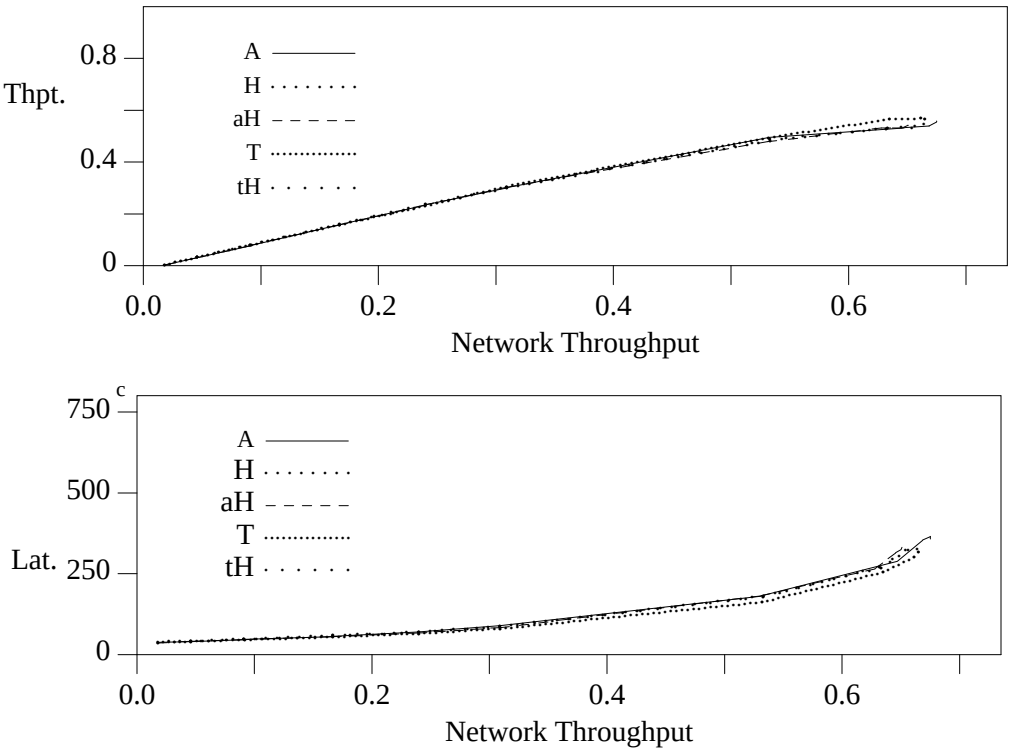


Figure 8.25: *Benchmark IV: DAMQ Buffers, Two-Counter Flow Control.* Serial traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0–2}$). Shown is the throughput vs. network throughput and latency vs. network throughput of G_0 for two-counter flow control with a variety of parameters (see Tab. 8.2 for details of the parameters).

uniformly-distributed traffic. In this benchmark, the best two-counter flow control results are similar to that of maximum usage flow control with a threshold of six buffer blocks. With the previous three benchmarks the lack of impact of hysteresis on network performance was not surprising, due to the static nature of the congestion in those benchmarks. Benchmark IV does create dynamic congestion patterns. The lack of impact of hysteresis here indicates that, perhaps, the congestion patterns or not ‘‘dynamic enough’’ for hysteresis to be helpful. This

possibility is examined further with Benchmark V, where the congestion patterns are more dynamic.

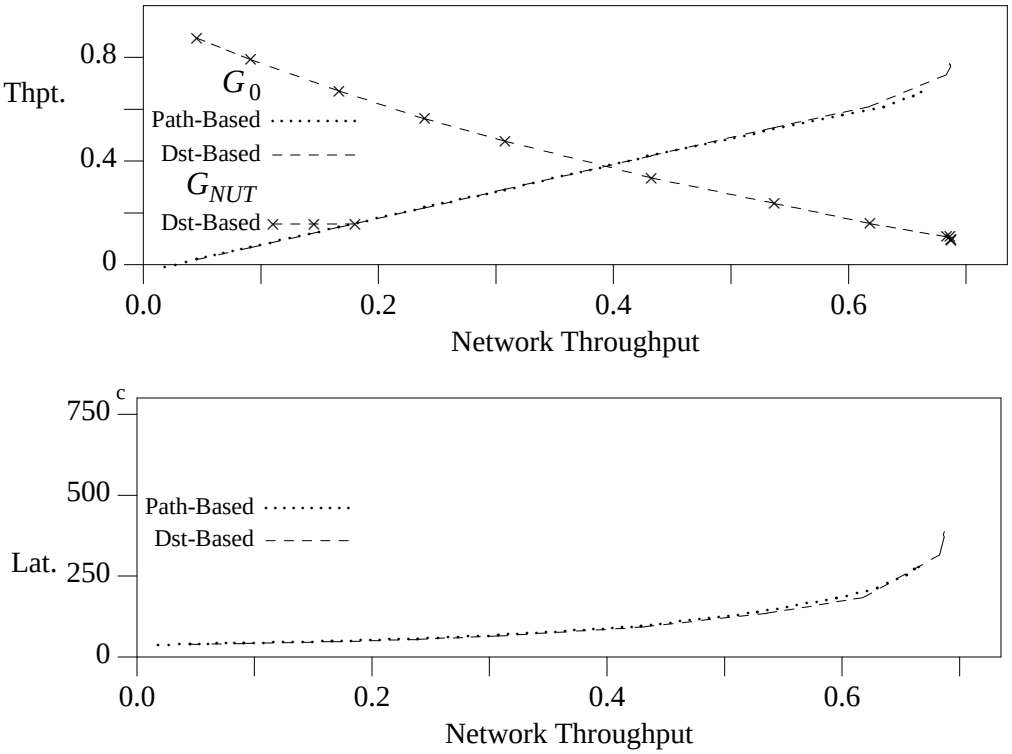


Figure 8.26: *Benchmark IV: DAMQ Buffers, Link-Based Flow Control.* Serial traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0–2}$). Destination- and path-based flow control. Throughput and average latency vs. total network throughput, G_0.

Figure 8.26 shows the performance of G_0 for networks implemented with destination-based flow control and path-based flow control. Path-based flow control allows G_0 to transmit data at a throughput of ≈ 0.69, and destination-based flow control supports a G_0 saturation throughput of 0.79. These flow control mechanisms achieve high throughputs for G_0 by throttling the throughput of G_{NUT}. As Fig. 8.26 shows, destination-based flow control throttles the G_{NUT} traffic as low

as 0.11 in response to contention with traffic from the other sender groups. Whether or not throttling G_{NUT} to this degree creates a ‘‘fairness problem’’ is an application-specific question; in the context of this evaluation, the total network throughput must be the basis for comparing the flow control mechanisms. While the high throughput achieved by G_0 under both destination- and path-based flow control almost offsets the severe throttling of G_{NUT}, the maximum total network throughput is lower for these flow control mechanisms than it is for maximum usage flow control with a threshold of ten buffer blocks.

This benchmark was a departure from the previous three benchmarks on two fronts. First, it created only mild network congestion — DAMQ buffers with blocking flow control achieve a network throughput of 0.68, only a bit shy of the 0.72 achieved under uniform traffic conditions. Benchmark III also minimally impacted network throughput, but for a very different reason. In Benchmark III, the point of congestion was unreachable by three fourths of the senders. Here, the congestion impacts every sender; it simply does not impact them very much. Second, the traffic pattern was dynamic. This caused points of congestion to be created and destroyed over the course of the simulation. In theory, one of the flow control characteristics this benchmark was to demonstrate was their ability to adapt under dynamic conditions.

This benchmark exposed the tendency of destination-based and path-based flow controls to over-throttle serial communication. This extreme throttling was a positive feature in Benchmarks I and II, but under traffic conditions in which high-throughput serial transmissions can coexist with the uniformly distributed traffic, this can cause a drop in performance.

In contrast, maximum usage flow control with a threshold of ten buffer blocks minimally throttles G_{NUT} and achieves the highest maximum network throughput of any of the flow control mechanisms for this benchmark. This mechanism depends upon congestion between G_{NUT} and G_0 at the first switching stage to throttle G_{NUT}, which allows G_{NUT} to utilize as much network bandwidth as it can. Further, by preventing any single packet queue from occupying an entire buffer, G_0 is not severely impacted by the G_{NUT} traffic (as it is under blocking flow control).

8.5.8. Benchmark V: 128 Senders Transmitting Multi-Packet Messages

For the fifth benchmark, G_{NUT} consisted of 128 (half) of the senders in the network. These senders cause congestion by transmitting 256-byte messages to uniformly distributed destinations at saturation throughput. Each message is split into eight-packet sequences. The other half of the senders transmit thirty-two-byte packets to uniformly-distributed destinations with a graduated applied load. As with Benchmark II, the senders are chosen such that G_{NUT} and G_0 ''mix'' in the first switching stage; each switch in *stage #1* receives input from two members of G_{NUT} and two members of G_0.

In this benchmark, there are two sources of congestion. The first is that the 256-byte messages generated by G_{NUT} can completely fill a buffer if the packet sequence encounters contention for a link. The second source is that the packet sequences which constitute a 256-byte message heavily utilize individual links for extended periods of time. This is similar to the congestion in Benchmark IV, except that (1) the congestion pattern in this benchmark is more dynamic and

(2) messages from G_{NUT} can collide with one another at any point in the network, and do so more frequently in this benchmark than in the previous. As with the previous benchmark, the three metrics used to evaluate the flow control mechanism's performance are the maximum total network throughput (the throughput of G_{NUT} and G_0 combined), average latency of the uniform traffic vs. network throughput, and the saturation throughput of G_0.

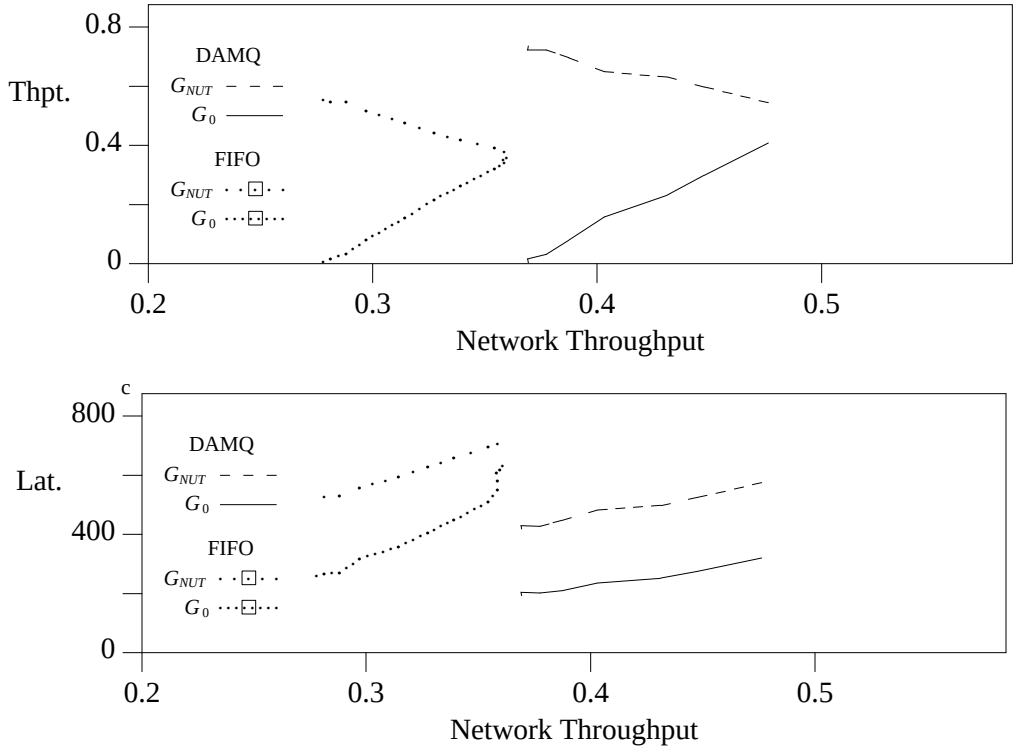


Figure 8.27: *Benchmark V: Blocking Flow Control.* Congestion traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_0). Switches with FIFO and DAMQ buffer, blocking flow control. Throughput and average latency vs. total network throughput, G_0 and G_{NUT}.

Fig. 8.27 shows the throughput and average latency vs. system throughput of

G_{NUT} and G_0 for networks of FIFO and DAMQ switches with blocking flow control. As previously mentioned, G_{NUT} operates at saturation throughput while the applied load of G_0 is graduated, G_{NUT} achieves its highest throughput and lowest latency when the applied load of G_0 is 0.0 m/sc (G_0 throughput equals 0.0 b/ls). As the applied load of G_0 is increased, the G_{NUT} traffic encounters more contention, and its throughput drops. Unlike the first four benchmarks, the G_{NUT} throughput remains higher than that of G_0 under blocking flow control, even when G_0 is at saturation throughput.

As can be seen from the figure, DAMQ buffers provide superior performance over FIFO buffers for both G_0 and G_{NUT} senders. Since the network congestion is highly dynamic, a buffering scheme which reduces the effect of output port contention makes a significant impact on network performance. The network of DAMQ buffers achieves a total network throughput of 0.48, compared to 0.36 for the network of FIFO buffers. While the DAMQ buffered network improves the throughput of both G_0 and G_{NUT} traffic, it does *not* provide a significant improvement in latency for the traffic of either group. Further, both the total network throughput and the G_0 throughput fall well short of the 0.71 throughput achievable under uniform traffic conditions.

Fig. 8.28 shows G_0 throughput vs. system throughput and G_0 latency vs. system throughput for DAMQ switches with blocking flow control, maximum usage flow control with thresholds of ten and six blocks, and static buffer allocation (SAMQ and SAFC buffers).

SAMQ buffering provides a lower G_0 average latency and slightly higher G_0 throughput than DAMQ buffering with blocking flow control. The graph indicates,

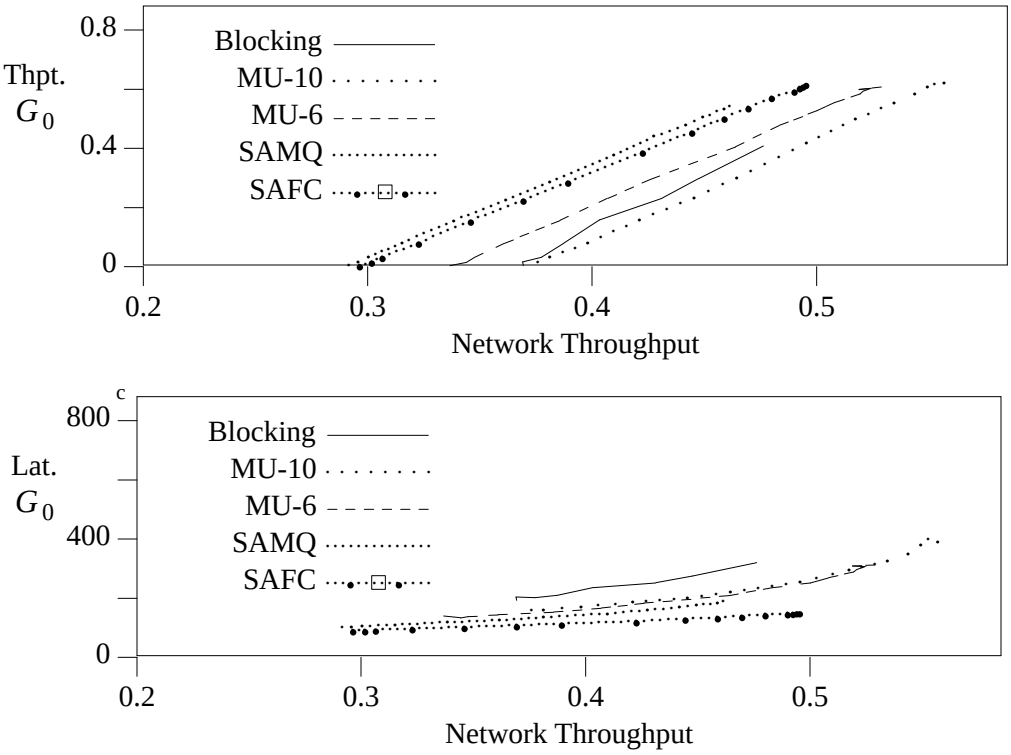


Figure 8.28: *Benchmark V: Maximum Usage Flow Control.* Non-uniform traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_0). Shown is the G_0 throughput vs. system throughput and G_0 latency vs. system throughput for blocking, static allocation and maximum usage flow control (six and ten buffer maximums).

however, that the SAMQ switch network has a lower network throughput — it has a much lower saturation throughput for G_{NUT}, which lowers network maximum throughput. The SAMQ switch network performs as well as the DAMQ switch network on G_0 traffic only because it so strongly throttles the G_{NUT} traffic, reducing the contention that the G_0 traffic encounters from the G_{NUT} messages.

While exhibiting better performance, the SAFC switch displays behavior

similar to that of the SAMQ switch. By over-throttling G_{NUT}, the network of SAFC switches achieves a maximum throughput for G_0 traffic equal to that for maximum usage flow control, and the average latency is significantly lower (Fig. 8.28). However, the total network throughput is lower for the SAFC switch network (0.50 vs. 0.55 for maximum usage flow control). If one considers the maximum network throughput to be *the* metric of interest, SAFC switch performance falls measurably short of the mark. If the latency of G_0 traffic is the primary concern, then SAFC buffering is preferable over maximum usage flow control for this benchmark.

Maximum usage flow control performance is superior to blocking flow control for this benchmark; the average G_0 latency is lower, and the maximum throughputs for both G_0 and the network as a whole are higher for maximum usage than for blocking flow control. Comparing maximum usage blocking thresholds, the ten-block congestion threshold supports higher total network throughput than a threshold of six buffer blocks while they both achieve approximately the same maximum G_0 throughput. Thus, it is the throughput of G_{NUT} traffic which benefits the most from the higher threshold. The higher threshold reduces the number of switches that the packets which constitute a message occupies; this, in turn, increases the probability that a G_{NUT} sender will be able to inject a message into the network in its entirety, without halting. The throughput and latency of G_0 seem to be indifferent to whether the threshold is ten or six blocks.

If there is a condition under which hysteresis is beneficial to a hop-level flow control mechanism, it is under dynamic congestion conditions such as this benchmark. We found, however, that two-counter (hysteresis) flow control is less

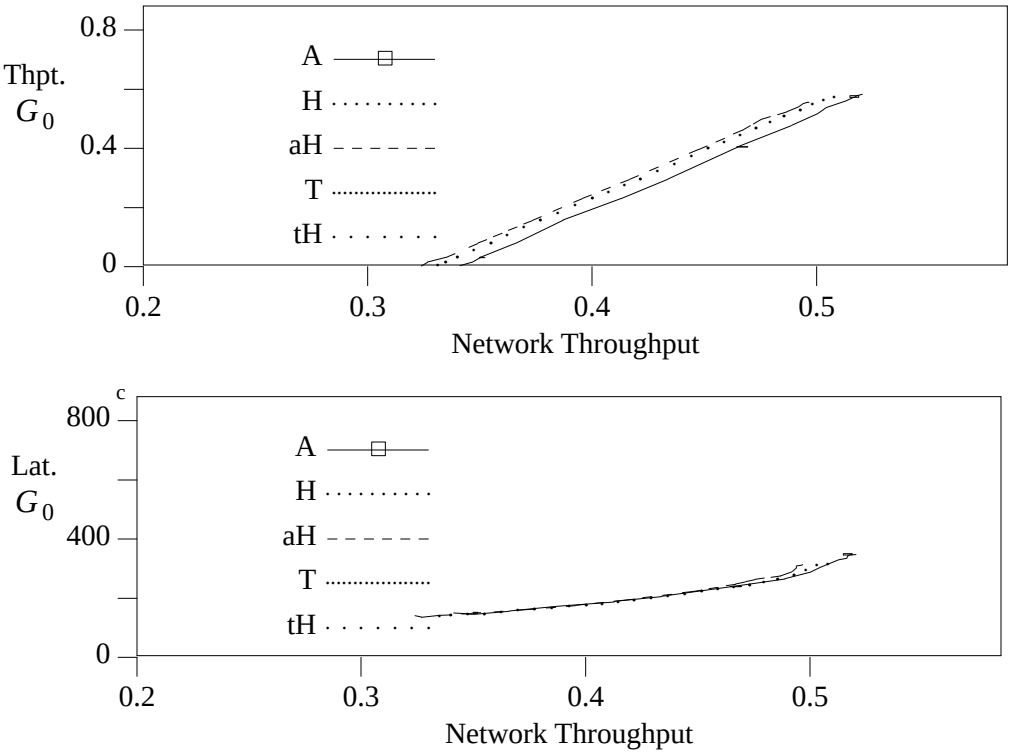


Figure 8.29: *Benchmark V: Two-Counter Flow Control.* Non-uniform traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_0). Shown is the G_0 throughput vs. system throughput and G_0 latency vs. system throughput for two-counter flow control with a variety of parameters.

effective than maximum usage. Fig. 8.29 shows the results for networks using two-counter flow control with a variety of parameters (see Tab. 8.2 for an explanation of parameter settings). G_0 performance for two-counter flow control is almost as good as that for maximum usage, but G_{NUT} throughput falls short of the throughputs achieved by maximum usage flow control (as can be seen by the reduction in the total network throughput).

As has been previously discussed, the concept behind hysteresis flow control

is that one wants to change flow control states only when the network state (specifically, the point of congestion) has ‘‘really’’ changed, and thus avoiding both throttling flow unnecessarily and allowing flow when one should not. Since these are situations which only arise under dynamic congestion conditions, we expected this benchmark to be where hysteresis flow control demonstrated its worth. Our results indicate that either (a) ‘‘false’’ throttles and releases do not significantly effect performance, (b) adding hysteresis to the flow control does not prevent ‘‘false’’ throttles and releases or (c) one must carefully match the thresholds of the hysteresis mechanism to the dynamics of the traffic pattern, and we did find the match. If any of those hypotheses are true, it suggests that hysteresis is not a beneficial characteristic of a flow control mechanism for a general purpose communication network.

Figure 8.30 shows simulator results for the link-based flow control mechanisms (destination- and path-based flow control). The link-based flow control mechanisms achieve very high throughputs for G_0 traffic — destination-based flow control supports a G_0 throughput of 0.85. Further, at a network throughput of 0.50, the average latency of G_0 packets is ≈ 150 c, markedly lower than the average G_0 latency for maximum usage flow control operating at 0.50 (≈ 260 c).

The saturation throughput of G_0 under destination-based flow control in this benchmark is significantly higher than the saturation network throughput obtainable under uniform traffic conditions — how destination-based flow control is able to achieve such a high throughput for G_0 is evident from the fact that the maximum network throughput (which, under this benchmark, is the average of the

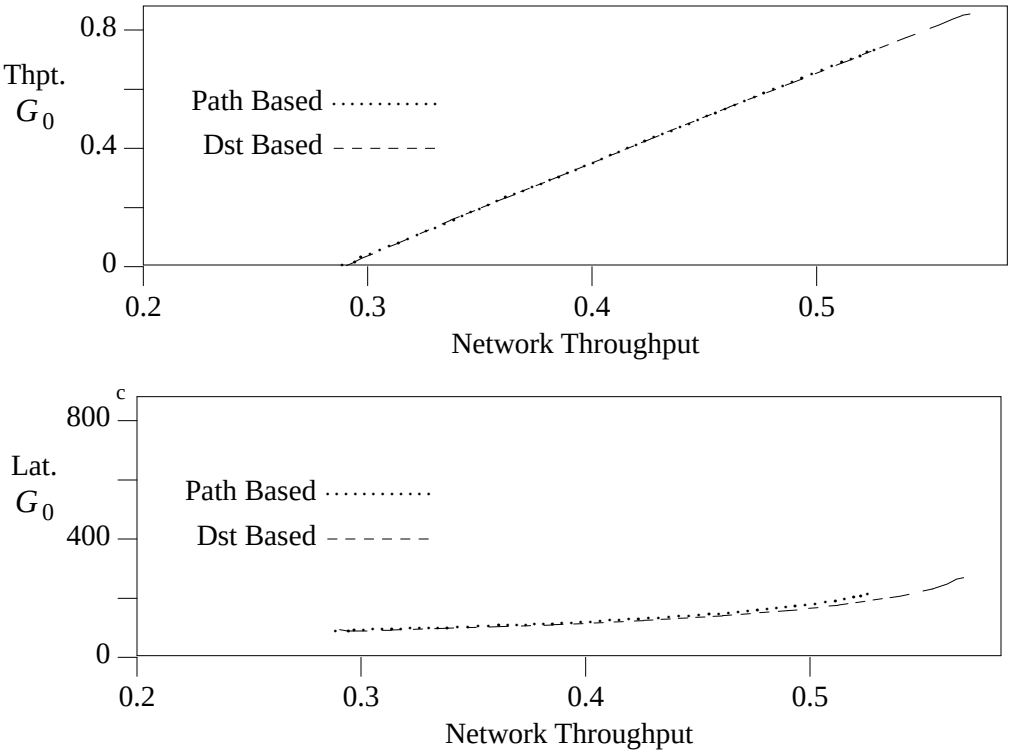


Figure 8.30: *Benchmark V: Link-Based Flow Control.* Non-uniform traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_0). Shown is the G_0 throughput vs. system throughput and G_0 latency vs. system throughput for destination- and path-based flow control.

G_{NUT} and G_0 maximum throughputs for this benchmark) is significantly lower than the maximum G_0 throughput. Destination-based flow control heavily penalizes the G_{NUT} traffic in favor of the G_0 traffic. G_{NUT} is restricted to 0.29 when G_0 is operating at saturation ($[0.85 + 0.29]/2 = 0.57$). Whether or not this behavior is desirable depends upon the applications being executed; while this flow control may seem to be ‘‘unfair’’ in the degree to which it is throttling G_{NUT} traffic in favor of G_0, the saturation throughput for the network as a whole is 0.57, which

is slightly higher than the total network saturation throughput for maximum usage flow control with a ten-block threshold (0.55).

While path-based flow control throttles G_{NUT} traffic as much or more than does destination-based flow control, G_{NUT} achieves a higher saturation throughput under path-based flow control. This is due to the fact that path-based flow control throttles G_0 more heavily than does destination-based (throttling on the basis of the next two hops to be traversed as opposed to the destination address). Thus, G_{NUT} encounters less network contention from G_0 packets under path-based flow control and achieves a higher throughput. The net result is that path-based flow control has a lower total network throughput than either destination-based or maximum usage flow control.

8.5.9. Benchmark VI: Uniform Traffic

Our final benchmark is to apply uniformly distributed traffic to networks implemented with different flow control mechanisms. The purpose of this benchmark is to measure the degree to which latency is increased and saturation throughput is reduced by the flow control mechanisms when there are no NUT spots in the network. For this benchmark, all senders transmit thirty-two byte messages, and each message's destination address is randomly selected from a uniform distribution of all possible destinations. Since the senders are not divided into separate groups, the performance metrics are the maximum (total) network throughput and the average latency per throughput.

Figure 8.5, presented in Sec. 8.5.2, showed latency vs. throughput for networks of FIFO and DAMQ switches with blocking flow control. The network

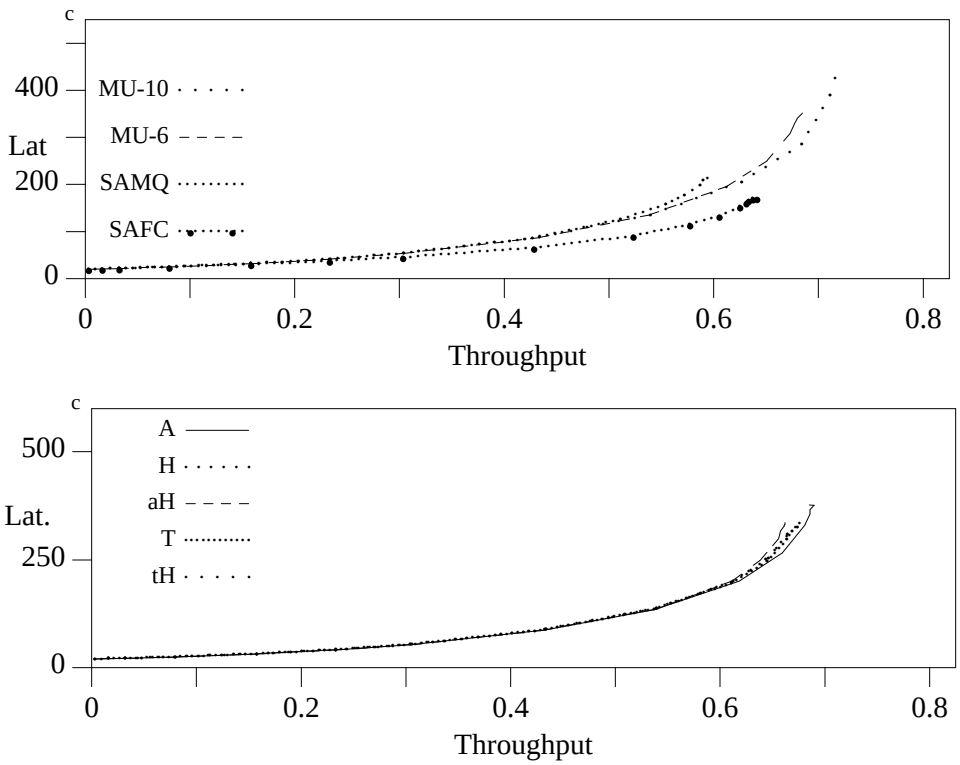


Figure 8.31: *Benchmark VI: Maximum Usage and Two-Counter Flow Control.* All senders transmitting 32-byte messages with uniformly distributed destinations. The average latency vs. throughput is presented; top graph shows static allocation and maximum-usage flow control (six and ten buffer maximums), bottom graph shows two-counter flow control with a variety of parameters (see Tab. 8.2 for explanation of parameters).

of FIFO switches saturates at a throughput of 0.49, while the network of DAMQ switches saturates at 0.71. At 0.20, both networks display an average latency of $\approx 35\,c$ (the minimum latency through the four-stage omega network is $20\,c$). Fig. 8.31 presents two graphs. The top graph shows the average latency vs. throughput for networks with static buffer allocation (SAMQ and SAFC buffers) and maximum usage flow control (thresholds of six and ten buffer blocks). The

bottom graph shows the performance of networks with two-counter flow control with a variety of parameter values (see Tab. 8.2 for explanation of parameter settings). For networks operating below saturation throughput, under uniform traffic conditions, these flow control mechanisms do not impact communication latency. The results indicate that the more restrictive flow control policies do reduce the saturation throughput. The flow control policy which significantly reduces the saturation throughput is static buffer allocation; SAMQ buffering drops the maximum throughput to 0.60, SAFC buffering drops it to 0.63.

Maximum usage flow control with a ten-block threshold actually improved the saturation throughput of uniformly distributed traffic. This mechanism achieved a network throughput of 0.72. An improvement of 0.01 is insignificant, and is within the error margin of the simulator (97.5% confidence intervals for simulation results of the two networks overlap), but one can understand why this mechanism would perform as well or better than blocking flow control under uniformly distributed traffic. The only time it throttles traffic is when a queue already has three packets in it — it does not cause output ports to be under-utilized, and it prevents a single queue from occupying the entire buffer, which eliminates the multi-queueing function of the buffer.

Two-counter flow control and maximum usage flow control with a threshold of six buffer blocks perform slightly worse than blocking flow control, saturating in the region of 0.68 (with some variance among the two-counter parameterizations). These mechanisms either throttle traffic more often then blocking flow control, or unblock blocked queues more slowly, and thus reduce the network throughput slightly.

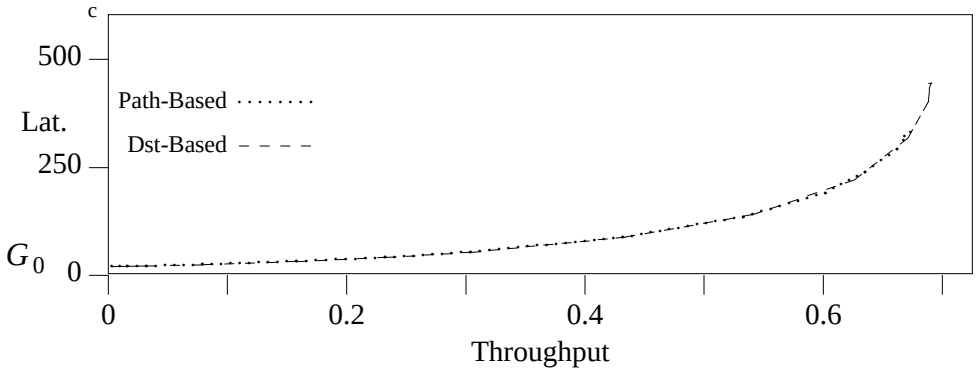


Figure 8.32: *Benchmark VI: Link-Based Flow Control.* All senders transmitting 32-byte messages with uniformly distributed destinations. Shown is the latency vs. throughput for Destination- and path-based flow control.

Figure 8.32 shows the performance of path-based and destination-based flow control. Destination-based flow control achieves a saturation throughput just slightly lower than that of blocking and maximum usage flow controls (0.69). Path-based flow control is slightly worse (0.67) — this mechanism is more likely to commit ‘‘false throttling’’ (throttling traffic when there is no congestion) than is destination-based flow control.

8.5.10. Combining Hop-Level Flow Control Mechanisms

In Secs. 8.5.4 through 8.5.9, a set of hop-level flow control mechanisms were evaluated. The results of this evaluation indicate that different flow control mechanisms protect the performance of G_{0-2} senders and the throughput of the network as a whole under different congestion patterns. For Benchmarks I and II (the hot spot benchmarks), destination-based flow control outperforms every other

flow control mechanism examined, both in terms of total network throughput and in terms of protection provided for G_0. On Benchmarks IV and V, destination-based flow control provides a greater degree of protection for the G_0 senders than does maximum usage flow control, but destination-based flow control and maximum usage flow control with a ten-buffer-block threshold achieve approximately the same maximum network throughput.

In Benchmark III, a static non-uniform traffic spot not associated with a single destination was generated in the network. Since only sixteen of the 256 senders were impacted by this congestion, there was little difference between the maximum total network throughput achieved by the different flow control mechanisms. Under destination-based flow control, however, the G_0 traffic was severely impacted by this congestion. As was discussed in Sec. 8.5.6, destination-based flow control is ineffective against congestion which is not destination-oriented. The buffer-based flow control mechanisms (maximum usage, hysteresis and static buffer allocation) were all significantly more effective than destination-based flow control in this case.

Benchmark VI addressed the performance of the flow control mechanisms under uniform traffic conditions. Static buffer allocation was the only flow control mechanism which significantly impacted network throughput under uniform traffic conditions; the rest of the mechanisms we examined had little impact. SAMQ buffers reduced throughput and had no redeeming features; SAFC buffers had a lower maximum throughput than the other flow control mechanisms, but it also supported a lower average latency at high throughputs.

The flow control mechanisms which we have examined are not all mutually

exclusive. That is to say, one can implement a flow control protocol which combines two or more of the mechanisms which we have discussed. In this section, we explore the results of combining destination-based flow control with maximum usage flow control and with SAFC buffering. Destination-based flow control is an obvious candidate for combining, as it provided the highest performance on most of the benchmarks. Maximum usage flow control also did well on many of the benchmarks, including Benchmark III — the benchmark that destination-based flow control did so poorly on. Although the SAFC buffer failed to achieve throughputs as high as the other flow control mechanisms, it was consistently able to maintain low average latencies.

Combining destination-based flow control and maximum usage flow control into a single flow control protocol is relatively straight forward. As with destination-based flow control, if a packet attempts to enter a buffer in which there is already a packet with the same destination, the incoming packet is rejected. Rejected packets are moved to the tail of the queue from which they were transmitted. Rejected packets are marked as BLOCKED, and packets which ‘‘cause’’ incoming packets to be rejected are marked as BLOCKER. When a BLOCKED packet moves to the front of a packet queue, it is moved to the tail without attempting to transmit it. When a BLOCKER packet is transmitted, this information is sent to the previous switch which clears the BLOCKED mark from all of the packets queued for the BLOCKER’s buffer. As with maximum usage flow control, any of the individual queues within a buffer may block reception. Full / not full signals for each of the queues within a buffer are fed back to the neighboring switch which transmits to that buffer (as opposed to a single full / not

full signal for the entire buffer). Switches pre-route packets to determine which queue they will be appended to in the next switch, and compare this information with the per-queue flow control to determine whether or not packets can be transmitted. Under this flow control combination, if the packet at the front of a queue is destined for a full queue and so cannot be transmitted, the packet is moved to the tail of the queue. This mechanism is called *blocked-packet-requeueing*. We did not provide blocked-packet-requeueing for plain maximum usage flow control, as (a) it requires additional hardware and firmware to support moving packets from the head to the tail of their queues and (b) maximum usage flow control would no longer support a FIFO packet ordering between source / destination pairs. Since destination-based flow control requires the presence of this hardware / firmware and preserves the FIFO ordering in the presence of re-queueing, it only makes sense to evaluate the performance of this flow control protocol *with* this optimization.

The implementation of destination-based flow control with SAFC buffers is identical to the implementation with maximum usage flow control just described. Combining destination-based flow control and SAFC buffering only makes sense, however, if the buffers are large enough that more than one packet can be in a single queue. For the network simulations presented previously in this chapter, the switches each have 128 bytes of buffer memory at each input port. Statically allocating a buffer memory of this size gives each queue within the buffer thirty-two bytes — enough memory to store a single packet. Destination-based flow control is not compatible with a buffer architecture in which the maximum queue length is a single packet, so the results presented in this section are from network

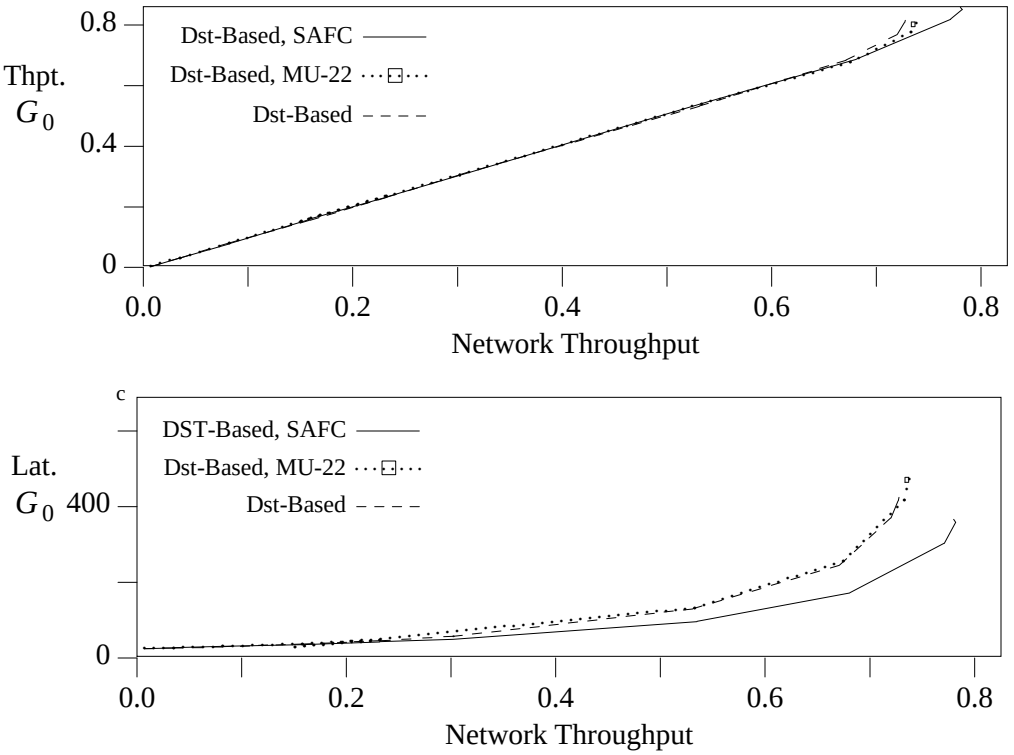


Figure 8.33: *Benchmark I: Group 0 Performance, Combined Flow Control.* Hot spot traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_{0-2}). Switches with 256 bytes per buffer. Shown is the G_0 throughput vs. network throughput and G_0 latency vs. network throughput for destination-based flow control alone, combined with maximum-usage flow control (threshold of twenty-two buffer blocks), and implemented with SAFC switches.

simulations in which each buffer has 256 bytes of storage.

Figures 8.33-8.37 present the throughput and average latency of G_0 vs. the network throughput for destination-based flow control combined with maximum usage flow control with a throttling threshold of twenty-two buffer blocks and destination-based flow control implemented on SAFC buffered switches. In order to compare the performance of these combined flow control protocols, results for

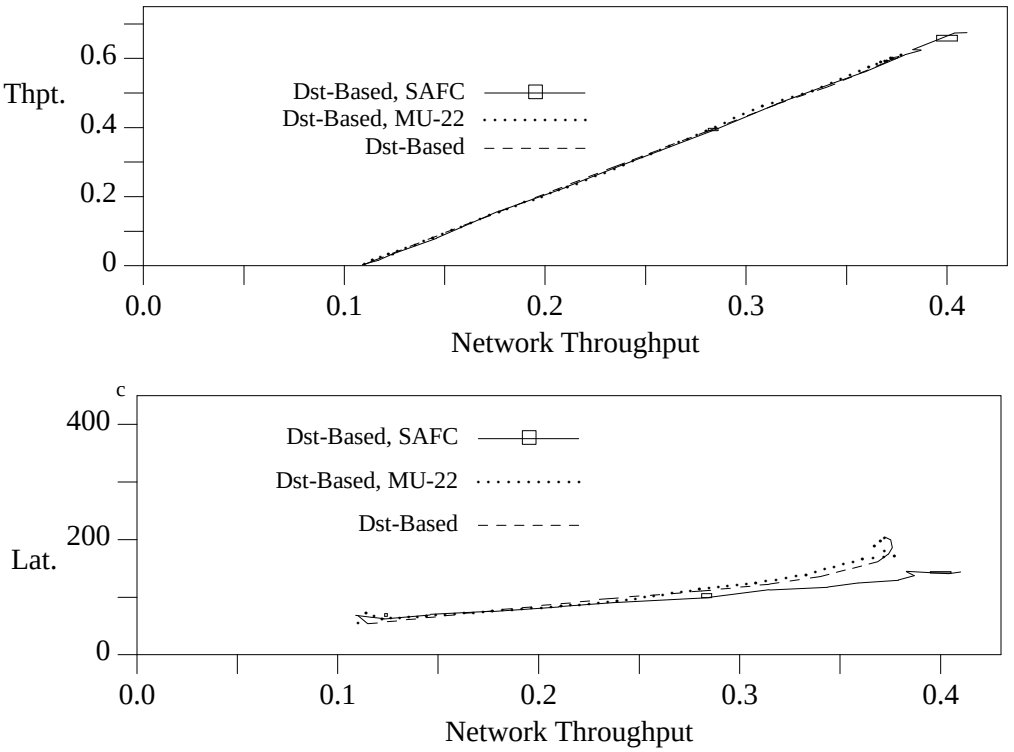


Figure 8.34: *Benchmark II: Combined Flow Control, Group 0.* Non-uniform traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_0). 256-byte buffers. Shown is the G_0 throughput vs. network throughput and G_0 latency vs. network throughput for destination-based flow control alone, combined with maximum-usage flow control (threshold of twenty-two blocks), and implemented with SAFC switches.

DAMQ switches with plain destination-based flow control are included in the figures, and Fig. 8.35 (results for Benchmark III) also shows the results for DAMQ switches with plain maximum usage flow control with a threshold of twenty-two blocks. Finally, Fig. 8.38 compares the performance of the combined flow control mechanisms to plain destination-based, maximum usage and blocking flow control under uniformly distributed traffic (Benchmark VI).

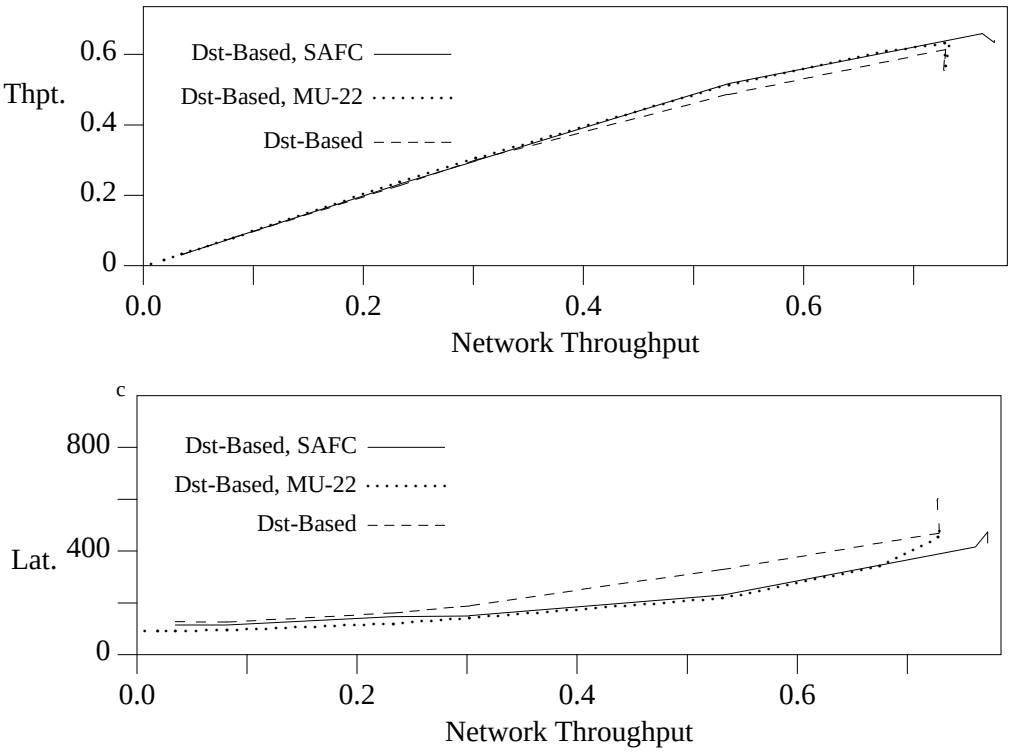


Figure 8.35: *Benchmark III: Combined Flow Control, Group 0.* Congestion traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0,1}$). 256-byte buffers. Destination-based flow control alone, combined with maximum-usage flow control (threshold of twenty-two blocks), and implemented with SAFC switches. Throughput and average latency vs. total network throughput, G_0.

The results of our simulations of combined flow control mechanisms can be succinctly summarized. Combining destination-based flow control with SAFC buffering provides superior network performance under uniform traffic conditions and in the presence of a wide variety of congestion patterns. The high performance of this combination has two sources. The first is the fact that it combines flow control mechanisms which make flow control decisions based upon different

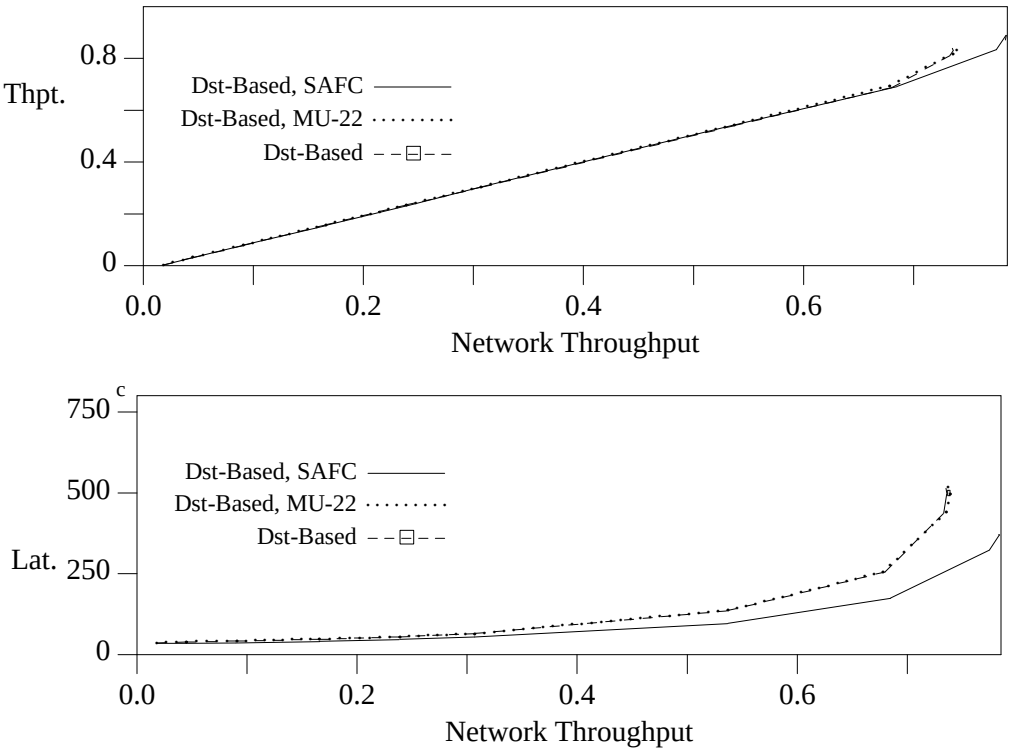


Figure 8.36: *Benchmark IV: Combined Flow Control, Group 0.* Congestion traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic ($G_{0–2}$). 256-byte buffers. Destination-based flow control alone, with maximum-usage flow control (threshold of twenty-two buffer blocks), and with SAFC switches. Throughput and average latency vs. total network throughput, G_0.

aspects of communication traffic. Static buffer allocation provides feedback as to the current buffer utilization; by limiting the number of packets in any one queue, the number of queues per buffer that have packets in them is increased, which in turn increases the link utilization. Destination-based flow control provides a predictive mechanism which (a) detects potential congestion multiple hops before the point of congestion and (b) provides highly directed back-pressure. The

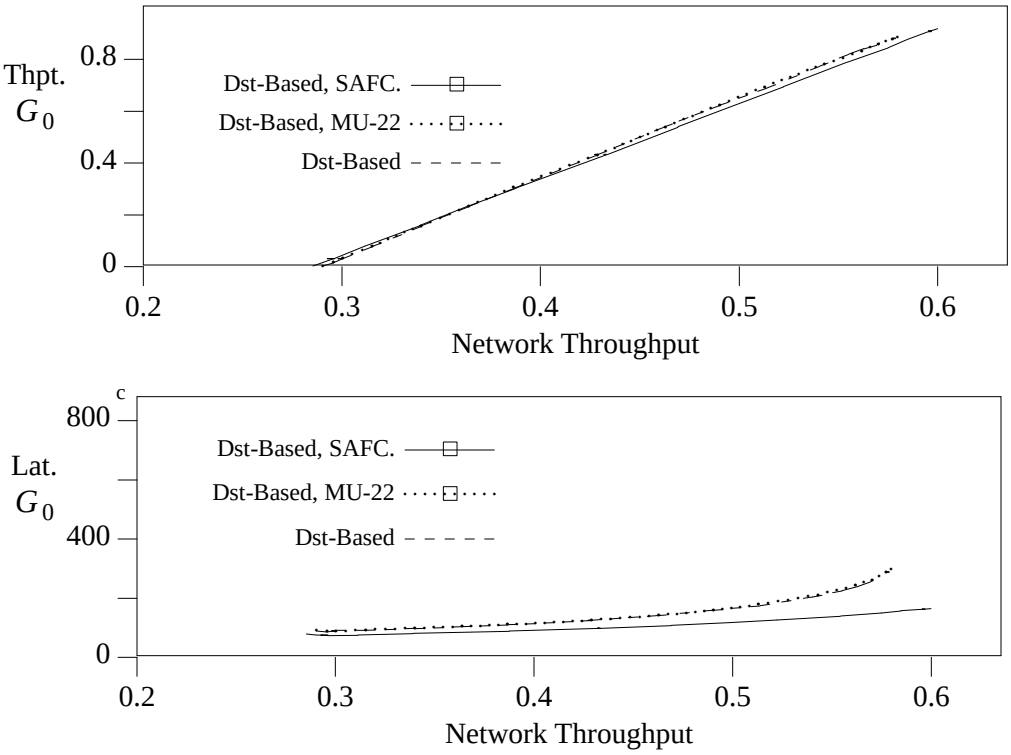


Figure 8.37: *Benchmark V: Combined Flow Control.* Non-uniform traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_0). 256-byte buffers. Shown is the G_0 throughput vs. system throughput and G_0 latency vs. system throughput for destination-based flow control alone, combined with maximum-usage flow control (threshold of twenty-two blocks), and implemented with SAFC switches.

combination of the two flow control mechanisms creates a congestion ‘‘firewall’’ which provides superior protection to the ‘‘well-behaved’’ senders in the system.

The second source of this combination’s high performance is shown in Fig. 8.38 (Benchmark VI, uniform traffic). Under uniform traffic conditions, destination-based flow control with SAFC buffers supports as high a saturation throughput as *any* flow control mechanism / buffer architecture we have examined,

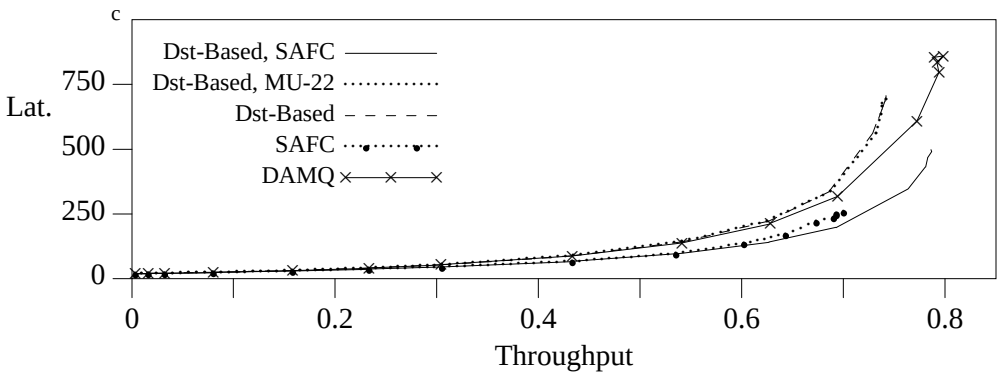


Figure 8.38: *Benchmark VI: Combined Flow Control.* All senders transmitting 32-byte messages with uniformly distributed destinations. 256-byte buffers. Shown is the latency vs. throughput for destination-based flow control alone, combined with maximum-usage flow control (blocking threshold of twenty-two blocks) and implemented with SAFC switches. Also shown are DAMQ and SAFC switches with blocking flow control.

and does so with a significantly lower average packet latency. Thus, even if it is only equally effective as the other flow control mechanisms which we have examined for preserving network performance in the presence of congestion, its superior performance under uniform traffic conditions would result in superior performance under congestion conditions.

Figure 8.38 suggests that destination-based flow control and SAFC buffers have a synergistic relationship — the combination results in a network whose performance is superior to that of either SAFC-buffered networks with blocking flow control or destination-based flow control with DAMQ buffers. In Chs. 4 and 6, we presented simulation results which indicate that DAMQ buffers provide performance superior to that of SAFC buffers under uniform traffic conditions. It is clear that SAFC buffers provide a greater degree of connectivity between buffers

and output ports, which gives SAFC buffers the potential to achieve a higher throughput than DAMQ buffers. However, they utilize their buffer memory less efficiently than do DAMQ buffers, which results in SAFC-buffered networks having a lower throughput under uniform traffic conditions than do networks with DAMQ buffers, for buffers up to 1024 bytes in size (Fig. 6.4).

The inefficient buffer utilization of SAFC buffers (and other statically allocated buffers) stems from the fact that queues can become full when there is still free memory in the buffer. If the packet at the head of a queue in an SAFC buffer (Q^0) is destined for a full queue in the next switch (Q^1), under blocking flow control Q^0 cannot transmit until Q^1 transmits (frees some of its buffer memory). With blocked-packet-requeueing, the packet at the head of Q^0 can be moved to the tail of Q^0; while the next packet in the queue is destined for the same buffer that the requeued packet was, it is not necessarily destined for Q^1, and thus may be transmittable.

The resulting increase in the utilization of SAFC buffers is shown by the fact that both the saturation throughput *and* the average latency at the saturation throughput are higher for SAFC buffers with destination-based flow control than for SAFC buffers with blocking flow control. Comparing the points of saturation throughput for SAFC-buffered networks with blocking and with destination-based flow control, we see that the average number of packets existing within the network† increases from 1445 to 3154.

† The average number of packets stored in the network equals the throughput per link times the number of links connected to destinations times the average latency divided by the number of bytes per packet.

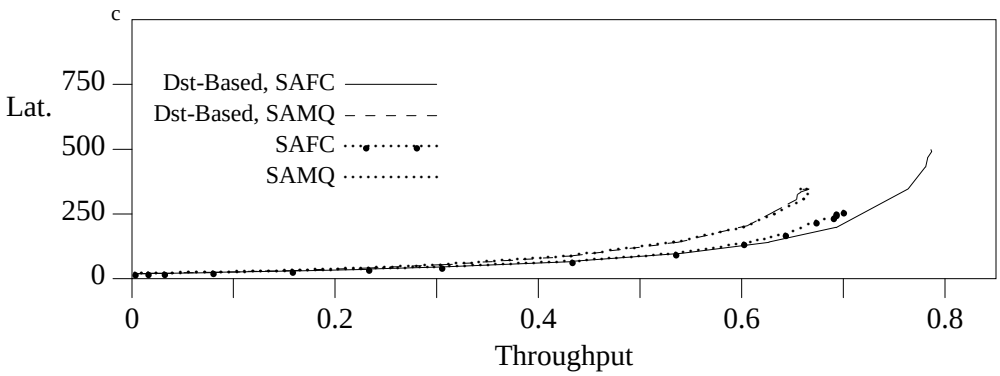


Figure 8.39: *Benchmark VI: Destination-Based Flow Control with SAMQ Buffers.* All senders transmitting 32-byte messages with uniformly distributed destinations. 256-byte buffers. Shown is the latency vs. throughput for SAFC- and SAMQ-buffered networks with blocking and destination-based flow control.

The phenomena of blocked-packet-requeueing does not totally explain the superior performance of destination-based flow control with SAFC buffers, however. Fig. 8.39 shows the average latency vs. throughput under uniform traffic conditions for 256-byte SAMQ- and SAFC-buffered networks, with both blocking and destination-based flow control. If the change in performance of the SAFC buffers was solely driven by the ability to move blocked packets to the rear of queues, then the network of switches with SAMQ buffers would also experience higher buffer utilization with destination-based flow control. The figures for average latency and saturation throughput suggest that SAMQ buffers received little or no increase in buffer utilization or network throughput under destination-based flow control. Thus, it requires more than blocked-packet-requeueing to improve network throughput under static buffer allocation.

The difference between the impact of destination-based flow control on

SAMQ and SAFC buffers stems from the number of packets which are available for transmission at any point in time. With SAMQ buffers, only the packets in buffers which are currently idle are considered. With SAFC buffers, on the other hand, every queue associated with a particular output port can be considered for every crossbar arbitration that the output port is available, since multiple queues within an SAFC buffer can transmit simultaneously. Since destination-based flow control moves blocked packets to the tail of queues, under SAFC buffering, when an output port becomes available, *every* packet currently on the chip destined for that output port is a candidate for transmission. This both increases link utilization relative to SAMQ buffers *and* reduces the average transmission latency.

In examining the results presented in Figs. 8.38 and 8.39, it is seen that, under uniform traffic conditions, destination-based flow control (a) increases the saturation throughput of SAFC buffers, (b) has little effect on SAMQ buffers and (c) reduces the saturation throughput of DAMQ buffers. We have explained the increased performance of the SAFC-buffered network, and the negligible impact destination-based flow control has on SAMQ-buffered networks. What remains to be explored is the negative impact destination-based flow control has on networks of switches implemented with DAMQ buffers.

Destination-based flow control *significantly* reduces the throughput of the DAMQ-buffer network transmitting uniformly-distributed traffic. While it impacts performance throughout the network, the most significant effect of destination-based flow control occurs in the final switching stage of the network. There, the output ports of the switches are connected directly to the destination nodes. Since destination-based flow control allows at most one packet per destination per buffer,

this means that there can be at most one packet per queue in the buffers of the last switching stage. This causes the 256-byte DAMQ buffers in the last switching stage of a network using destination-based flow control to operate as though they were 128-byte SAMQ buffers. This phenomena occurs in SAMQ and SAFC buffers, as well. It does not impact the SAMQ-buffered network, however, as the applied load on the last switching stage is low enough that the reduction from 256-byte SAMQ buffers to 128-byte SAMQ buffers does not reduce the throughput. It *does* impact SAFC buffers, but to a lesser degree than DAMQ buffers. Since SAFC buffers allow multiple packets to be transmitted from a single buffer, 128-byte SAFC buffers support a higher packet throughput than do 128-byte SAMQ buffers.

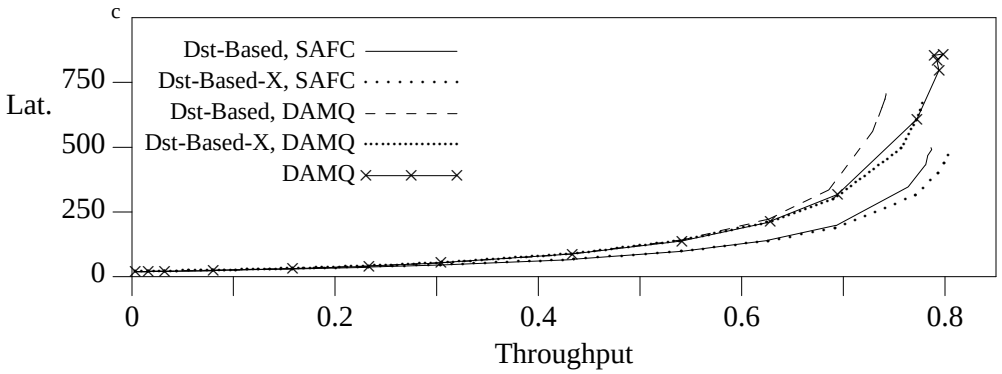


Figure 8.40: *Benchmark VI: Destination-Based Flow Control, Suppressed at Destination.* All senders transmitting 32-byte messages with uniformly distributed destinations. 256-byte buffers. Average latency vs. throughput for blocking flow control with DAMQ buffers, destination-based flow control, and destination-based flow control with discarding suppressed at the destination switches (‘‘Dst-Based-X’’ in the graph legend).

This hypothesis can be verified by simulating a multistage interconnection

network in which every switching stage except the final one implements destination-based flow control. Fig. 8.40 shows the average packet latency vs. network throughput for uniformly-distributed traffic in DAMQ- and SAFC-buffered networks (a) with destination-based flow control throughout the network and (b) with destination-based flow control operating in the first three switching stages, and blocking flow control in the final switching stage (labeled *Dst-Based-X* in Fig. 8.40).

As can be seen from Fig. 8.40, suppressing destination-based flow control in the last switching stage improves the SAFC-buffered network only slightly but has a significantly positive effect on the DAMQ-buffered network. Still, SAFC buffers with destination-based flow control achieve a higher throughput than do DAMQ buffers with destination-based flow control, even with discarding disabled for the final switching stage. This is simply a matter of there being more bandwidth spent transmitting packets which are subsequently discarded in the DAMQ-buffered network than in the SAFC-buffered network.

The reason that the DAMQ-buffered network discards more packets is that dynamic buffer allocation more efficiently utilizes the available buffer space than does static buffer allocation. This is true even with blocked-packet-requeueing. This can be seen from the fact that the product of the average latency at the maximum network throughput and the maximum network throughput is higher for the DAMQ-buffered network than the SAFC buffered network; there are more packets in a DAMQ buffered network than there are in an SAFC buffered network at a given throughput. Thus, in a DAMQ buffered network, when a packet arrives at a buffer, there are (on average) more packets in the buffer and a higher chance

that the incoming packet will encounter a packet with the same destination and be discarded.

8.6. Flow Control Mechanism Implementation

In the previous sections of this chapter, the performance of a number of hop-level flow control mechanisms have been evaluated. This section describes their implementation costs and discusses the implications this cost has on their feasibility and effectiveness.

8.6.1. Blocking Flow Control

Blocking flow control is the simplest of the flow control mechanisms examined in this dissertation. The function of blocking flow control is to halt transmissions to a buffer when it cannot guarantee that there is enough free memory to store the packet. The hardware to implement blocking flow control consists of subunits to perform three separate functions: (1) track the amount of free space in the buffer, (2) transmit flow control information to the neighboring switch, and (3) use the flow control information received from a neighbor to allow / prevent transmissions to that neighbor.

Under blocking flow control, the buffer cannot accept a new transmission if there is not enough free memory to store a maximum-size packet (thirty-two bytes, in our implementation of the DAMQ buffer — Ch. 7 and[Fraz89]). A counter is used to track the amount of free space in the buffer memory (the DAMQ buffer tracks buffer blocks, FIFO, SAMQ and SAFC buffers track bytes). When the value in this counter indicates that there are fewer than thirty-two bytes available in the

buffer, the flow control mechanism signals that the buffer is full.

Our simulations assume that there is one wire for flow control which accompanies each communication link. This wire transmits the flow control state of the destination buffer to the neighboring switch. With unidirectional links such as those of the multistage interconnection networks simulated for this dissertation, this wire is necessary. In the general case, however, wires dedicated to flow control represent bandwidth which is not being used to transmit data. It is possible to avoid dedicating bandwidth to flow control in networks whose connections are bidirectional (i.e. neighboring switches each having a link transmitting to the other). In these networks, one can ‘‘piggyback’’ flow control information on packets traversing the link. Hence, the flow control information competes for network bandwidth with normal traffic. In this case, minimizing the bandwidth required for flow control is desirable. As discussed earlier (Sec. 8.2), schemes with hysteresis reduce the required bandwidth for flow control. Such schemes may thus be desirable if there are no dedicated lines for flow control. Another consideration when evaluating piggybacked flow control is the latency that this adds to the flow control feedback; if the latency is too high, the link idle time between packets may have to be increased in order to prevent buffer overflow.

Having received the flow control feedback from their neighbors, the switches' crossbars must have a mechanism to prevent connecting buffers to output ports whose destination buffer is full. How this is done depends directly upon the implementation of the crossbar arbiter and the buffer / crossbar interface. An example is shown in Fig. 8.41. For a 4×4 switch, each buffer has four request lines to the crossbar indicating to which output ports the buffer has packets destined.

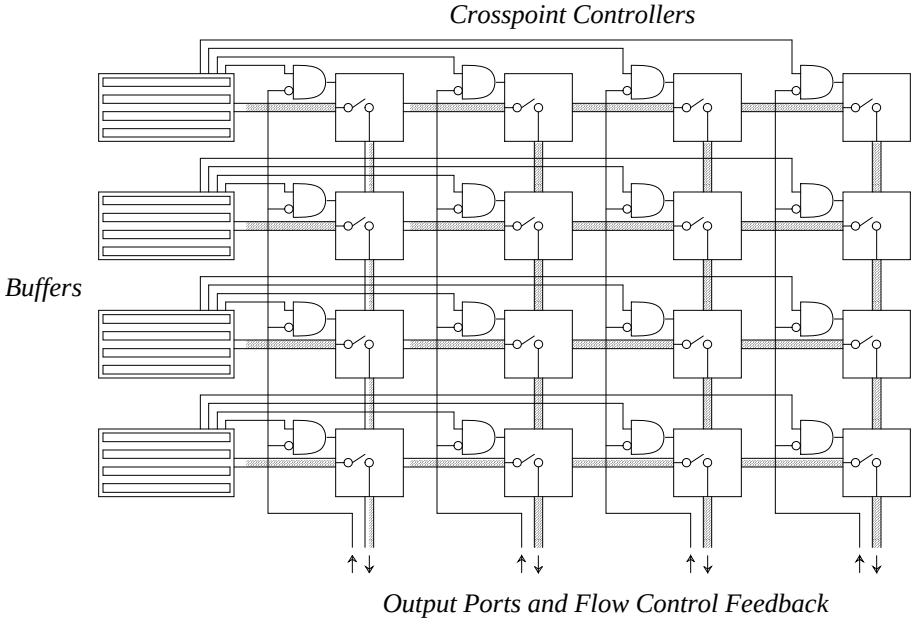


Figure 8.41: *Crossbar Hardware for Blocking Flow Control.* AND-gates are used to suppress connection requests from buffers to output ports whose destination input ports have blocked transmissions.

For FIFO buffers, at most one of the four lines can be asserted at any given point in time (assuming static routing). For non-FIFO buffers, the request lines correspond to non-empty packet queues within the buffer. In either case, the flow control signal associated with an output port can be used to mask requests for that output port (Fig. 8.41), preventing packets from being transmitted to that output port.

The total hardware overhead for blocking flow control is minimal. The most significant hardware addition is the wire dedicated to transmitting flow control information. Since one is only interested in changes in the flow control state, and since these changes do not happen on the majority of the clock cycles, dedicating the wire to flow control wastes some amount of bandwidth. The alternative (piggy-backing flow control information) (a) is topology dependent (b) is considerably more complex and (c) may lose bandwidth due to the added latency

of the feedback.

8.6.2. Queue-Based Flow Control

Our evaluation of hop-level flow control mechanisms included maximum usage flow control, statically allocated buffers, and two-counter flow control. These flow control mechanisms are similar to one another, in that they all block packets on a *per-queue* basis. Thus, the functional units which implement these mechanisms are very similar to one another. Each queue has its own counter to track the amount of data in the queue. Maximum usage and two-counter flow control also need a central counter to track free space in the DAMQ buffer. In the simulations presented in this chapter, it was assumed that each communication link was accompanied by an n-bit flow control channel, where n is the number of packet queues in the buffer. If $n = 4$ and the links are eight bits wide, twelve wires connect neighboring switches and flow control occupies 33% of the communication bandwidth. In other words, going from blocking flow control to queue-based flow control requires an increase in the link bandwidth of 33% (nine wires to twelve wires). This is an excessive amount of overhead; none of the queue-based flow control mechanisms increased the maximum network throughput by 33% in *any* of the benchmarks presented in this chapter.

An alternative to dedicating a separate wire to each queue in the buffer is to maintain a single wire for flow control feedback, and to sequentially transmit separate bits of flow control information over this wire. While this does increase the latency of the flow control feedback, it is an incremental increase, and would have negligible impact on network performance. It does increase the complexity of

the flow control logic, in that there must be logic on the transmitting end of the communication link that can parse incoming flow control information and decode it for the rest of the switch, but this is simple logic.

The queue-based flow control mechanisms all require that packets be *pre-routed,* and that the crossbar arbiter not connect buffers to output ports if the packet at the head of the queue associated with that output port is destined to a blocked queue. The complexity of pre-routing packets is dependent upon whether the results of pre-routing are transmitted with a packet. If the pre-routing result is not transmitted with the packet, two routing operations must be performed at each switch. The tradeoff between these two options is the relative overhead of dedicating $\log_2(n)$ bits of bandwidth per packet to transmit port information vs. an increase in the amount and complexity of the packet reception and routing hardware.

The complexity of the pre-routing mechanism is also dependent upon the routing protocol used by the network. It is at its worst for table-lookup routing. For it, each switch must maintain routing tables for each of its neighbors; after the output port and new header are determined for a packet, then the new header would be used to access the table associated with the neighbor connected to the destined output port to discover the destination queue. Even if the results of pre-routing are attached to packets, a new packet header must be determined before the pre-routing can take place. For topology-based routing algorithms, the hardware which determines the output port of a packet can be enhanced to return the destined output ports of both the current and the next switch without a major increase in complexity. Pre-routing becomes a particularly complex operation when used in

conjunction with dynamic routing. In this case, a switch must make intelligent routing decisions (or, at least, avoid *bad* routing decisions) one hop away from where the switching will actually take place.

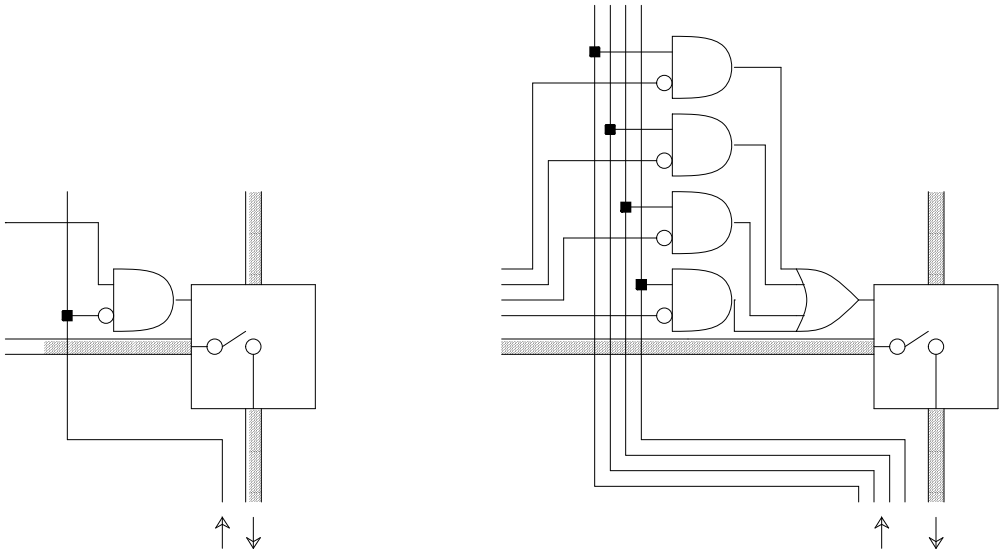


Figure 8.42: *Crosspoint Controller Hardware for Queue-Based Flow Control.* A single crosspoint controller from the crossbar of a switch using blocking flow control (Fig. 8.41) and queue-based flow control.

Figure 8.42 compares the complexity of the crosspoint controllers of the crossbar of a switch using queue-based flow control to that of a switch using blocking flow control. Pictured are single crosspoint controllers of a 4×4 switch for both queue-based and blocking flow control systems. When requesting an output port connection in a queue-based system, a buffer must choose one of four wires to assert — the asserted wire indicates which queue in the *next* buffer the packet is destined for. Combinatorial logic compares the request signal to the four flow control lines associated with the output port. If the flow control signal does

not mask the request, then the arbiter may make the connection, whereupon the buffer transmits the packet.

A difference between maximum usage and statically allocated buffer flow control is that, with maximum usage flow control, an incoming packet can be stored to *any* available buffer memory. This allows the buffer to determine where a packet will be stored before it actually arrives. With a statically allocated buffers, on the other hand, packets must arrive at an input port with the identifier of the queue to which they are to be appended in their header, or there must be a *staging buffer* at the input port to store the packet while it is being routed. The use of a staging buffer may add to the minimum cut through latency of the switch (the speed with which a packet can move through a switch). On the other hand, including the queue identifier in the header increases bandwidth overhead per packet.

8.6.3. Destination- and Path-Based Flow Control

Since the only difference between destination-based flow control and path-based flow control is the hardware to determine whether or not to accept a packet, this subsection focuses on destination-based flow control. Destination-based flow control requires considerably more enhancements to the basic switching hardware than do the other flow control mechanisms discussed in this chapter; their use alters the fundamental operation of a switch. When a packet is transmitted, the buffer space it occupies is not immediately deallocated under destination-based flow control. Rather, the buffer waits for an **ACK** signal from the next buffer, indicating that the packet is being accepted, before it deallocates its buffer space.

If it instead receives a **NACK**, the buffer must relink the packet at the tail of the queue and not deallocate the buffer space. We will first examine the the hardware required to determine whether a packet is to be ACK-ed or NACK-ed, then the communication overhead of sending ACK / NACK signals to neighboring switches, followed by the hardware required to handle requeueing NACK-ed packets, and finally we will examine the hardware implications of our enhancements to destination-based flow control (*abortive transmissions* and *restricted retry*).

Determining whether a packet is to be ACK-ed or NACK-ed requires comparing the *new* header of an arriving packet (i.e. the header returned by the router) to the headers of the packets currently stored in the buffer. This requires storing packet headers in a content-addressable memory (CAM). Every time a new header is read from the router interface, it is driven over the CAM — if a match is made, the reception is aborted and a NACK is generated. Altering the header register array to be content-addressable significantly increases the size of the array, but since it is a relatively small functional unit to begin with (in our example DAMQ buffer implementation — Ch. 7[Fraz89], — there are twelve header registers for a ninety-size-byte buffer), then this is not a major issue.

With destination-based flow control, the entire address portion of the packet headers is compared. For path-based flow control, however, only that part of the header which relates to the next n hops is compared. This is the only difference between the two, and for multistage interconnection networks, this simply means that a smaller CAM is needed for path-based flow control.

Since destination-based flow control must be accompanied by some form of blocking or queue-based flow control which will prevent buffer overflow, the

number of wires required to indicate ACK / NACK of a transmission is an open question. If a queue-based flow control mechanism is using a single wire, transmitting flow control information serially, then one option is to increase the number of codes to include ACK and NACK signals. Another option is to modify the accompanying flow control mechanism such that no pre-routing or blocking is performed; the ACK / NACK signal would indicate not only whether the packet collided with another destined to the same address, but also whether an attempt was made to append the packet to a full queue. This allows a single wire to easily transmit full / not-full information, as well as conveying the ACK (since the target application of this flow control mechanism is a tightly-coupled multicomputer, the NACK can be implemented as a fast timeout). This represents a different flow control mechanism than was described in Sec. 8.5.10, where the combined flow control mechanisms were evaluated. This would drop performance somewhat relative to the simulation results presented in that section, since the crossbar controller of a switch would not be able to distinguish between packets destined for full vs. not-full queues, but it would substantially reduce the amount of hardware required to implement the flow control, as well as eliminate the pre-routing (which, as was previously mentioned, is problematic under certain circumstances).

For the DAMQ buffer to have the option of either re-linking a packet to the tail of its queue or deallocating the memory blocks to the linked list, one would add an additional head and tail register to the buffer. These registers define an additional linked list of buffer blocks for the buffer — the *currently-transmitting list*. When a packet is transmitted, instead of moving its buffer blocks directly from the head of its queue to the tail of the free list, they are appended to the

currently-transmitting list. If an ACK is received, the block at the end of the currently-transmitting list is appended to the free list, the value in the tail register of the currently-transmitting list is copied to the tail register of the free list, the null register for the currently-transmitting list is set (this empties the list), and the memory is deallocated. If a NACK is received, the currently-transmitting list is appended to the queue's linked list by the same process. While this will add considerable complexity to the firmware which controls the buffer, the addition of a head / tail register pair is an insignificant increase in hardware.

We previously described two enhancements to the path-based flow control schemes: *abortive transmissions* and *restricted retry*. Since these enhancements measurably improve the performance of both destination-based and path-based flow control, the link-based flow control simulation results presented in this chapter were with both enhancements. As can be seen in Fig. 8.43, both *abortive transmissions* and *restricted retry* contribute to the performance of DAMQ buffers with destination-based flow control under uniformly distributed traffic conditions. Further, when evaluating SAFC buffers with destination-based flow control, the blocked-packet-requeueing mechanism was critical to the performance of the combination (blocked-packet-requeueing and abortive transmissions use the same hardware mechanisms).

The abortive transmissions enhances network performance to a significantly greater extent than does restricted retry under uniform traffic conditions. This is because restrictive retry only improves performance when a packet would, without restrictive retry, make multiple attempts to traverse a link and be repeatedly rejected due to collisions with the same packet. This is unlikely to occur in a

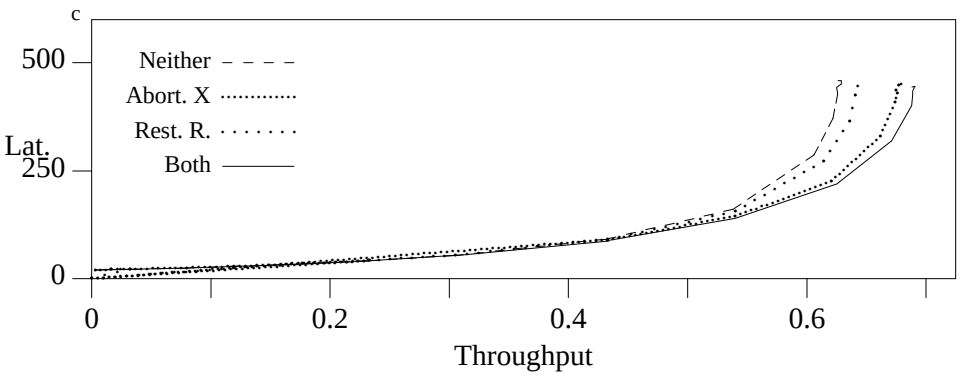


Figure 8.43: *Evaluating Link-Based Flow Control Enhancements: Benchmark VI.* All senders transmitting 32-byte messages with uniformly distributed destinations. Shown is the latency vs. throughput for Destination-based flow control with no enhancements, with abortive transmissions, with restricted retry, and with both enhancements.

uniformly distributed traffic pattern.

Figure 8.44 shows the performance of abortive transmissions and restricted retry for destination-based flow control under Benchmark II (a static hot spot created by half of the senders transmitting at saturation throughput, 3% of the packets to a hot spot, 97% of the packets uniformly distributed). Under this congestion pattern there is a point of congestion (the hot spot) which exists for an extended period of time. These are the circumstances in which individual packets may attempt to traverse a single link multiple times before being accepted by the next buffer. Restricted retry alone performs slightly better than does abortive retry alone, and the performance of the two enhancements together is no better.

The functionality of abortive transmissions significantly overlaps that of restricted retry. The sole purpose of restricted retry is to avoid ‘‘spending’’ link

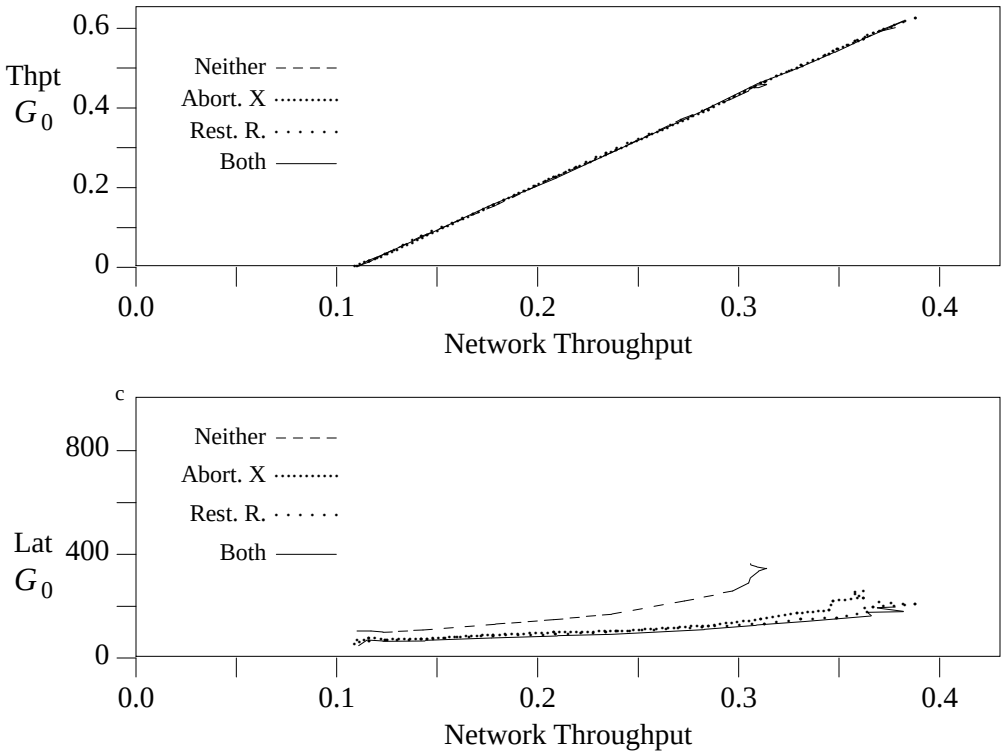


Figure 8.44: *Evaluating Link-Based Flow Control Enhancements: Benchmark II.* Hot spot traffic (G_{NUT}) operating at saturation, graduating applied load of uniform traffic (G_0). Shown is the G_0 throughput vs. network throughput and G_0 latency vs. network throughput for destination-based flow control with neither enhancement, abortive transmissions, restricted retry, and with both enhancements.

bandwidth on packets for which it is known that the packet will be rejected by the next buffer. Since abortive transmissions significantly reduce the amount of bandwidth spent on *all* rejected packets, restricted retry can at best provide an incremental improvement in performance.

Of the two enhancements we have proposed for link-based flow control, abortive transmissions requires the least amount of additional hardware. Abortive

transmissions involves simply ceasing to transmit a packet once a NACK has been received (the buffer must continue to requeue the packet — the simulations take this into account). The amount of hardware required to implement this is dependent upon the ‘‘strictness’’ desired of the mechanism — if one allows the transmission to continue for some cycles after the NACK’s reception, the hardware required to implement abortive transmissions is negligible. Given the significant improvement resulting from its implementation, it is a clear positive tradeoff to implement link-based flow control *with* abortive transmissions / blocked-packet-requeueing.

Restricted retry associates two bits of status with each packet queued in a buffer. One bit indicates whether or not a packet is a BLOCKER. BLOCKER-s are packets which have caused incoming packets to be discarded, i.e. they *blocked* a packet. The other bit indicates whether a packet is a BLOCKEE. BLOCKEE-s are packets which have attempted a transmission and been rejected. These bits should be both physically and logically located near the header registers, since their access pattern is similar to that of the header registers. When a packet is successfully received into a buffer, both its BLOCKER and BLOCKEE bits are cleared. When a CAM lookup matches one of the headers in the header register array, the BLOCKER bit of the matched packet is set. When a packet is rejected and requeued, the buffer firmware must explicitly set the BLOCKEE bit associated with the packet. This can be done when the head register of the currently-transmitting list is read to requeue the packet. When a packet whose BLOCKEE bit is set moves to the front of a queue, this must be detected and the packet moved to the rear of the queue without attempting to transmit it. Finally, when

transmitting a packet whose BLOCKER bit is set, a flow control signal must be sent to the neighboring switch to clear the BLOCKEE bits of all packets destined for this buffer.

While no single aspect of restricted retry is overly complex, taken as a whole this enhancement adds considerable hardware and complexity to the switch. One of its most difficult aspects is the amount of concurrency required. A large number of separate and sophisticated actions occur in parallel with the possibility of it being interrupted in the middle of one activity in order to perform another. Much of this complexity is eliminated if the queues are physically independent (i.e. SAMQ and SAFC buffers). In this case, each queue has its own transmission control logic and arbiter interface, which greatly simplifies the queue manipulations. However, since abortive transmissions provides the majority of the ‘‘enhancement’’ of enhanced path-based flow control, one would be disinclined to implement restricted retry.

8.7. Flow Control: Summary and Conclusions

This chapter presented an analysis of flow control mechanisms intended for use in scalable multicomputers. *Flow control* regulates the flow of data through the network by managing the allocation of resources (e.g., wires and buffers) to information units (e.g., packets) [Dall90a]. In many systems, the flow control mechanism exists solely to prevent the loss of data due to overflowing packet buffers. In this chapter, we explored the ability of flow control to optimize the performance of communication networks under uniform and non-uniform traffic patterns.

A network link is a *point of congestion* when the applied load of traffic attempting to traverse the link exceeds the bandwidth of the link. This usually occurs due to non-uniform traffic patterns, where there is a group of senders that are all transmitting packets that need to traverse the congested link. If packets are forced to wait to use the critical resource (the congested link) for an extended period of time, they will cause other packets to queue up waiting to use the network resource that they are occupying (the packet buffer). It is in this way that congestion spreads through a network. We termed this *congestion propagation.*

The bandwidth of the points of congestion directly limits the throughput of the non-uniform traffic group (G_{NUT} — the senders causing the point(s) of congestion). A flow control mechanism cannot, on its own, improve this. However, if congestion is allowed to propagate, other senders in the system which are not members of G_{NUT} may have their communication performance reduced by the congestion caused by G_{NUT}. A flow control mechanism can improve network performance by preventing congestion propagation, protecting well-behaved senders from those that cause congestion.

To support our analysis, a taxonomy for flow control mechanisms was developed. It uses three characteristics to classify flow control mechanisms. The first is the action taken to control traffic (blocking packets vs. discarding them). Second, where in the network the flow control decisions are made (global vs. end-to-end vs. hop-level flow control) was considered. The final characteristic used is the trigger which activates the flow control mechanism (predictive flow control, which is triggered by particular transmission patterns, vs. reactive flow control, which responds to congestion). In most cases, tightly-coupled interconnection

networks for scalable multicomputers utilize flow control mechanisms which are blocking, hop-level, and reactive.

In our evaluations of hop-level flow control mechanisms, we found the queue-based flow control mechanisms (maximum usage, static buffer allocation and hysteresis flow control) to be effective when the point of congestion is close to the senders. These flow control mechanisms are less effective when the point of congestion is more then two hops away from the senders because they do not provide a means for directing back-pressure, and thus do not differentiate between senders in G_{NUT} and others.

Queue-based flow control is also effective when the congestion is highly dynamic (i.e. bursty traffic, where the bursts are short-lived). Under these circumstances, the congestion has little time to propagate; if the packets which comprise the burst can be stored in the buffers of one or a few switches, the lack of directed back-pressure is not an issue. Since a queue-based flow control mechanism prevents the congestion-causing packets from entirely filling the buffers, they support a higher throughput (under these conditions) than does straight blocking flow control.

We found destination-based flow control [Dias89] to be highly effective against congestion which is associated with a particular destination. Even when the packets collide a large number of hops away from their destination, if they have the same destination, this flow control mechanism identifies the non-uniform traffic and prevents congestion propagation. This includes non-uniformities caused by long messages or any burst of traffic to a single destination (a situation which may be common in a scalable multicomputer with distributed secondary memory).

The situation in which destination-based flow control is ineffective is when the traffic from multiple nodes collides at some point internal to the network, where the packets do not share the same destination (an internal non-uniform traffic spot). Under this circumstance, destination-based flow control is ‘‘blind’’ to the congestion, and performs as blocking flow control would.

Perhaps the most interesting result was the effectiveness of SAFC buffers combined with destination-based flow control. In previous chapters, we showed how static buffer allocation reduces the degree to which buffers are utilized, limiting the maximum throughput of a network. Sec. 8.5.10 showed that, by implementing destination-based flow control with SAFC buffers, buffer utilization is dramatically improved. The result is a buffer architecture / flow control mechanism combination whose performance in every benchmark is superior to that of any other examined in this dissertation. It even outperforms DAMQ buffers with blocking flow control under uniform traffic conditions (Fig. 8.38).

This chapter makes a number of contributions to the field of communication network architectures for scalable multicomputers. Among them is the means by which we evaluated the flow control mechanisms. Segregating the senders into groups, having one group to cause congestion within the network, and then measuring the performance of the individual groups and the network as a whole reveals a wealth of information as to the performance characteristics of the flow control mechanism. When combined with our congestion benchmark suite, this methodology provides a thorough evaluation of a flow control mechanism’s ability to support high performance communication.

Chapter Nine
Summary and Conclusions

The potential for large multiprocessors and multicomputers to achieve high performance can only be realized if they are provided with a high-throughput low-latency communication network. Fast small $n \times n$ switches 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. The primary contribution of this dissertation is the architecture of a new packet buffer, the *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 ‘‘pre-route’’ 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. We have also presented the DAMQ Buffer Chip, a stand-alone implementation of the DAMQ buffer appropriate for use as the building block of a

multi-chip switch. We demonstrated that 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 buffer supports 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. For uniform traffic, our modeling and simulations show that these features result in large performance improvements over conventional FIFO buffers. The DAMQ buffer also provides a higher maximum throughput than do other practical non-FIFO buffer organizations with the same total buffer storage capacity.

In order to use large multiprocessors and multicomputers for real-time applications, it must be possible to guarantee, with a high degree of confidence, that high-priority traffic can be transmitted through the network with low specified latency. With conventional interconnection networks used in multiprocessors and multicomputers, the worst-case latency of traffic through the network can increase dramatically as the load on the network increases. This may prevent the use of these interconnection networks for critical real-time applications or force their use with very low utilization (and thus high cost/performance ratio) in order to guarantee low maximum latency.

For interconnection networks composed of small $n \times n$ switches, we have

shown that simply increasing the size of conventional buffers in the switches does not result in improved performance for high priority packets. There is thus a fundamental need for buffer organizations which support high-priority packets. We have developed a technique for efficiently supporting high-priority traffic, while maintaining good performance for normal traffic. The modifications to the DAMQ buffer in order to support high-priority traffic involve a few additional control registers and somewhat more complex arbitration of the crossbar switch. Overall these modifications are expected to require only a small percentage increase in total buffer area.

We have evaluated alternative approaches to providing support for high-priority packets in $n \times n$ switches. This evaluation was based on implementation complexity and simulation studies of a multistage interconnection network transmitting packets with two levels of priority. Our simulations have demonstrated five key points. (1) In a conventional network with a single priority level for all packets, worst-case latency for packets can be several times higher than average latency. Hence, there is a need to identify and provide preferential treatment to those packets for which fast service is particularly important. (2) Using the priority of packets as the determining factor in arbitrating contention within each switch does not provide sufficient support for high-priority traffic. Such arbitration does not reduce the 99^{th} percentile latency of the high-priority packets to the level of average normal packet latency, even under moderate network load. (3) As long as the proportion of high-priority traffic is low, dedicated queues for high priority traffic reduce the 99^{th} percentile latency of the high-priority packets to the level of the average latency for normal traffic under a

wide range of network throughputs. (4) When the proportion of high-priority traffic is large and the network is heavily loaded, multiple dedicated queues for high-priority traffic can reduce the 99^{th} percentile latency relative to the single dedicated queue approach. However, this performance advantage is marginal and the associated implementation cost is very high. (5) Dedicated buffers for the high-priority traffic can provide the same performance advantage as dedicated queues in shared DAMQ buffers. However, since DAMQ buffers are needed to maximize performance for normal traffic, the implementation cost for dedicated buffers is much higher than the cost of adding dedicated queues to DAMQ buffers. Hence, there is no reason to use dedicated buffers.

Our results indicate that the DAMQ buffer with a single dedicated queue for high-priority packets provides support for high-priority traffic which is superior to the support provided by alternate switch designs based on a dedicated high-priority queue. This resulting network performance, with respect to high-priority traffic, is very close to the performance of a network based on ‘‘ideal’’ switches for which a practical implementation is not feasible. Hence, given the low hardware overhead of the scheme based on a DAMQ buffer with a single high-priority queue, it is clearly the preferable option.

The DAMQ buffer architecture was designed to support variable-length packets and virtual cut-through — features which are not present under synchronous transmission protocols. In Ch. 6, we demonstrated the value of non-FIFO buffer and dynamic buffer allocation in asynchronous communication networks. We showed that, under uniform traffic conditions, for both constant and variable-length packets, the DAMQ buffer supports a higher saturation network

throughput over multistage interconnection networks than do FIFO buffers or statically allocated non-FIFO buffers. For constant-length packets, we showed that networks operating at saturation throughput *self-synchronize.* For a network of FIFO or DAMQ switches, this results in network behavior similar to that of a synchronous network — the number of connections that can be made during each crossbar arbitration is maximized and wasted network bandwidth is minimized. For SAMQ and SAFC switches (multi-queue buffers whose memory is statically allocated among their queues), the presence of virtual cut through reduces the impact of their inefficient buffer utilization, resulting in a slight improvement in their performance relative to the synchronous protocol. With variable-length packets, however, the statically allocated buffers suffered a significant reduction in their performance due to the amount of buffer memory lost to fragmentation. The DAMQ buffer, by virtue of its dynamic buffer allocation, experienced minimal fragmentation and maintained its high performance.

Chapter 6 also presented the use of the DAMQ buffer in two-dimensional torus networks. It is shown how the deadlock-free dimensional routing algorithm presented in [Dall87a] can be implemented using multi-queue buffers. The key factor of this implementation is that no single queue within a buffer be allowed to occupy the entire buffer space. Statically allocated buffers (and the Torus Routing Chip [Dall86]) accomplish this by physically separating the buffer memory associated with each queue. The DAMQ buffer prevents individual queues from occupying an entire buffer via maximum usage flow control. The simulation results for 11×11 and 21×21 torus networks indicate that the DAMQ buffer supports a higher saturation throughput than do statically allocated multi-queue

buffers, for small buffer sizes. Further, the throughput of buffered torus networks is shown to be ≈ 100% higher than that of networks with wormhole routing and minimal buffering [Dall90b]. We also show that buffered torus networks are highly reactive (in contrast to unbuffered torus networks[Dall90b]). This is due to the fact that congestion can propagate around the ring within a given dimension, creating a positive-feedback loop which causes the network throughput to drop dramatically when the applied load exceeds the point of maximum network throughput.

The DAMQ Chip, presented in Ch. 7, is a VLSI implementation of a stand-alone version of the DAMQ buffer. As an interface chip in the node of a multicomputer, it asynchronously receives packets from and transmits packets to other DAMQ Chips. As an example implementation of the DAMQ buffer, it demonstrates that a non-FIFO buffer can be efficiently implemented in VLSI. We presented details of the implementation, including the floorplan and critical path timing.

We have demonstrated that the DAMQ buffer can operate at high clock rates, despite its complex control. Hence, other factors, such as the buffer memory or the inter-chip links, will limit the raw bandwidth. For a DAMQ buffer with four queues and a packet size of 32 bytes, it was shown that the DAMQ buffer will have one less packet slot available to it than a FIFO buffer occupying approximately the same chip area. As was shown in Chs. 4 and 6, with as few as two packet slots (64 bytes), a network with DAMQ buffers can outperform a FIFO buffer network which has an additional packet slot. With three packet slots, the DAMQ buffer network will saturate at a throughput approximately 23% higher than a FIFO buffer

network with four packet slots. The DAMQ buffer is thus shown to be a highly effective building block for packet switches, demonstrating that for VLSI switches, increased control complexity may lead to higher performance.

Chapter 8 examines the performance of hop-level flow control mechanisms for scalable multicomputers. A significant contribution of this chapter is our methodology for measuring the relative performance of flow control mechanisms via simulation. A congestion benchmark suite was developed, where each benchmark in the suite creates a different pattern of congestion using stochastically-generated communication traffic. The benchmark suite was chosen to ‘‘stress’’ the flow control mechanisms in a number of different ways such that any flow control which preserves the network performance through all of the benchmarks is likely to perform well under the conditions which occur in a general-purpose multicomputer computer.

The chapter contains the description and analysis of a number of hop-level flow control mechanisms. Their performance is evaluated via the congestion benchmark suite described above. We also examined the complexity of their implementation. There was a single flow control / buffer architecture combination whose performance was superior to all other switching architectures examined in this dissertation for every benchmark in the congestion suite — destination-based flow control with statically allocated, fully-connected (SAFC) buffers. SAFC buffers are multi-queue buffers in which the buffer memory is statically partitioned among the queues, and each queue has its own read port. This allows multiple packets to be transmitted from a single buffer to different output ports simultaneously. While this superior connectivity gives SAFC buffers the potential

to support a higher throughput than DAMQ buffers, the results presented in Chs. 4 and 6 suggest that DAMQ buffers offer higher performance for moderately-sized buffers. However, when SAFC buffering is combined with destination-based flow control, a significant improvement in network performance results. Even under uniform traffic conditions, SAFC buffers with destination-based flow control supports a higher maximum throughput and a lower average latency for given throughputs than do DAMQ buffers with blocking, destination-based, or any other flow control mechanism that we examined.

Bibliography

[Agar90] A. Agarwal, B.-H. Lim, D. Kranz, and J. Kubiatowicz, ‘‘APRIL: A Processor Architecture for Multiprocessing,’’ *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pp. 104-114 (May 1990).

[Agar91] A. Agarwal, ‘‘Limits on Interconnection Network Performance,’’ *IEEE Transactions on Parallel and Distributed Systems* **2**(4) pp. 398-412 (October 1991).

[Ahma89] H. Ahmadi and W. E. Denzel, ‘‘Survey of modern high-performance switching techniques,’’ *IEEE Journal on Selected Areas in Communications* **7**(7) pp. 1090-1103 (Sept. 1989).

[Bert87] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall (1987).

[Cerf74] V. G. Cerf and R. E. Kahn, ‘‘A protocol for packet network interconnection,’’ *IEEE Transactions on Communications* **COM-22**(5) pp. 637-648 (May 1974).

[Cher89] D. R. Cheriton and C. L. Williamson, ‘‘VMTP as the transport layer for a high-performance distributed system,’’ *IEEE Comunications Magazine* **27**(6) pp. 44-47 (June 1989).

[Clar88] D. D. Clark, M. L. Lambert, and L. Zhang, ‘‘NETBLT: A high throughput transport protocol,’’ *Proceedings of ACM SIGCOM ’88*, pp. 353-359 (1988).

[Crow85] W. Crowther, J. Goodhue, R. Gurwitz, R. Rettberg, and R. Thomas,

‘‘The Butterfly Parallel Processor,’’ *IEEE Computer Architecture Newsletter*, pp. 18-45 (September/December 1985).

[Dall90a] W. Dally, ‘‘Network and Processor Architecture for Message-Driven Computers,’’ pp. 140-222 in *VLSI and Parallel Computation*, ed. Robert Suaya and Graham Birtwistle, Morgan Kaufmann Publishers, Inc. (1990).

[Dall86] W. J. Dally and C. L. Seitz, ‘‘The Torus Routing Chip,’’ *Distributed Computing* **1**(4) pp. 187-196 (October 1986).

[Dall87a] W. J. Dally and C. L. Seitz, ‘‘Deadlock-Free Message Routing in Multiprocessor Interconnection Networks,’’ *IEEE Transactions on Computers* **C-36**(5) pp. 547-553 (May 1987).

[Dall87b] W. J. Dally, L. Chao, A. Chien, S. Hassoun, W. Horwat, J. Kaplan, P. Song, B. Totty, and S. Wills, ‘‘Architecture of a Message-Driven Processor,’’ *14th Annual Symposium on Computer Architecture*, pp. 189-196 (June 1987).

[Dall88] W. J. Dally, *The J-Machine: System Support for Actors*, MIT (September 1988).

[Dall90b] W. J. Dally, ‘‘Performance Analysis of *k*-ary *n*-cube Interconnection Networks,’’ *IEEE Transactions on Computers* **C-39**(6) pp. 775-785 (June 1990).

[Dall91] W. J. Dally, ‘‘Express cubes: improving the performance of *k*-ary *n*-cube interconnection networks,’’ *IEEE Transactions on Computers* **40**(9) pp. 1016-1023 (Sept. 1991).

[Davi72] D. W. Davies, ‘‘The control of congestion in packet-switching networks,’’ *IEEE Transactions on Communications* **COM-20**(3) pp. 546-550 (June 1972).

[Davi92] A. Davis, ‘‘Mayfly: A Scaleable, Parallel Processing System,’’ *List and Symbolic Computation* **5**(1/2) pp. 7-47 (May 1992).

[Dias81] D. M. Dias and J. R. Jump, ‘‘Packet Switching Interconnection Networks for Modular Systems,’’ *Computer* **14**(12) pp. 43-53 (December 1981).

[Dias89] D. M. Dias and M. Kumar, ‘‘Preventing Congestion in Multistage Networks in the Presence of Hotspots,’’ *Proceedings of the 1989 International Conference on Parallel Processing*, pp. I.9-I.13 (August, 1989).

[Fraz89] G. L. Frazier and Y. Tamir, ‘‘The Design and Implementation of a Multi-Queue Buffer for VLSI Communication Switches,’’ *International Conference on Computer Design*, pp. 466-471 (October, 1989).

[Fuji83] R. M. Fujimoto, ‘‘VLSI Communication Components for Multicomputer Networks,’’ CS Division Report No. UCB/CSD 83/136, University of California, Berkeley, CA (1983).

[Gerl80] M. Gerla and L. Kleinrock, ‘‘Flow Control: A Comparative Survey,’’ *IEEE Transactions on Communications* **COM-28**(4) pp. 553-574 (April 1980).

[Gott83] A. Gottlieb, R. Grishman, C. Kruskal, K. McAuliffe, L. Rudolph,

and M. Snir, ‘‘The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer,’’ *IEEE Transactions on Computers* **C-32**(2) pp. 175-189 (February 1983).

[Inte89] Intel Corporation, *82596 User’s Manual*. 1989.

[Irla78] M. I. Irland, ‘‘Buffer Management in a Packet Switch,’’ *IEEE Transactions on Communications* **COM-26**(3) pp. 328-337 (March 1978).

[Kalm90] C. R. Kalmanek, H. Kanakia, and S. Keshav, ‘‘Rate controlled servers for very high speed networks,’’ *Proceedings of GLOBECOM ’90* **1** pp. 300.3.1-300.3.9 (1990).

[Karo86] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, ‘‘Input vs. Output Queueing on a Space-Division Packet Switch,’’ *IEEE Global Telecommunications Conference*, pp. 659-665 (December 1986).

[Karo87] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, ‘‘Input vs. Output Queueing on a Space-Division Packet Switch,’’ *IEEE Transactions on Communication* **COM-35** pp. 1347-1356 (December 1987).

[Kerm79] P. Kermani and L. Kleinrock, ‘‘Virtual Cut Through: A New Computer Communication Switching Technique,’’ *Computer Networks* **3**(4) pp. 267-286 (September 1979).

[Kerm80] P. Kermani and L. Kleinrock, ‘‘Dynamic Flow Control in Store-and-Forward Computer Networks,’’ *IEEE Transactions on Communications* **COM-28**(2) pp. 263-270 (February 1980).

[Klei80] L. Kleinrock and P. Kermani, ‘‘Static Flow Control in Store-and-

Forward Computer Networks,'' *IEEE Transactions on Communications* **COM-28**(2) pp. 271-278 (February 1980).

[Knig89] T. F. Knight, ''Technologies for Low Latency Interconnection Switches,'' *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pp. 351-358 (June, 1989).

[Kuma84] M. Kumar and J. R. Jump, ''Performance Enhancement in Buffered Delta Networks Using Crossbar Switches and Multiple Links,'' *Journal of Parallel and Distributed Computing* **1**(1) pp. 81-103 (August, 1984).

[Kuma86] M. Kumar and G. F.Pfister, ''The Onset of Hot Spot Contention,'' *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 28-34 (August, 1986).

[Kuri89] L. Kurisaki and T. Lang, ''Multistage Networks Including Traffic with Real-Time Constraints,'' *Proceedings of the 1989 International Conference on Parallel Processing*, pp. I.19-I.22 (August, 1989).

[Lang88] T. Lang and L. Kurisaki, ''Nonuniform Traffic Spots (NUTS) in Multistage Interconnection Networks,'' Computer Science Technical Report CSD-880001, University of California, Los Angeles, CA (January 1988).

[Lawr75] D. H. Lawrie, ''Access and Alignment of Data in an Array Processor,'' *IEEE Transactions on Computers* **C-24**(12) pp. 1145-1155 (December 1975).

[Lee86] G. Lee and C. P. Kruskal, and D. J. Kuck, ‘‘The Effectiveness of Combining in Shared Memory Parallel Computers in the Presence of ‘Hot Spots’,’’ *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 35-41 (August, 1986).

[Leis92] C. e. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. c. Kuszmaul, M. A. S. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak, ‘‘The Network Architecture of the Connection Machine CM-5,’’ TMC Technical Report, Thinking Machines Corporation, Cambridge, MA (July 1992).

[Leno92] D. Lenoski, J. Laudon, K. Gharachorloo, and W.-D. Weber et al., ‘‘The Stanford DASH Multiprocessor,’’ *Computer* **25**(3) pp. 63-79 (March 1992).

[Lill91] S. L. Lillevik, ‘‘The Touchstone 30 Gigaflop DELTA Prototype,’’ *The Sixth Distributed Memory Computing Conference*, pp. 671-677 (April, 1991).

[Lutz84] C. Lutz, S. Rabin, C. Seitz, and D. Speck, ‘‘Design of the Mosaic Element,’’ *Proceedings of the MIT Conference on Advanced Research in VLSI*, pp. 1-10 (January 1984).

[McMi86a] McMillen et al., ‘‘Packet Switched Multiport Memory NxM Switch Node and Processing Method,’’ *United States Patent 4,630,258*, (December 16, 1986).

[McMi80] R. J. McMillen and H. J. Siegel, ‘‘The Hybrid Cube Network,’’ *Distributed Data Acquisition, Computing, and Control Symposium*,

pp. 11-22 (December 1980).

[McMi86b] R. J. McMillen, ‘‘Packet Switched Multiple Queue NxM Switch Node and Processing Method,’’ *United States Patent 4,623,996*, (November 18, 1986).

[Mukh86] U. Mukherji, ‘‘A schedule-based approach for flow-control in data communication networks,’’ *Ph.D. thesis, MIT*, (February 1986).

[Ngai89] J. Y. Ngai, ‘‘A Framework for Adaptive Routing in Multicomputer Networks,’’ Caltech-CS-TR-89-09, Computer Science Department, California Institue of Technology (1989).

[Nikh92] R. S. Nikhil and G. M. Papadopoulos, and Arvind, ‘‘*T: A Multithreaded Massively Parallel Architecture,’’ *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 156-167 (May 1992).

[Oed] W. Oed, ‘‘The Cray Research Massively Parallel Processor System CRAY T3D,’’ Cray Research Technical Report ().

[Pfis85a] G. F. Pfister and V. A. Norton, ‘‘‘Hot Spot’ Contention and Combining in Multistage Interconnection Networks,’’ *IEEE Transactions on Computers* **C-34**(10) pp. 943-948 (October 1985).

[Pfis85b] 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,’’ *1985 International Conference on Parallel Processing*, pp. 764-771 (August 1985).

[Rama91] G. Ramamurthy and R. S. Dighe, ‘‘Distributed source control: a network access control for integrated broadband packet networks,’’ *IEEE Journal on Selected Areas on Communications* **9**(7) pp. 990-1002 (September 1991).

[Rath89] E. P. Rathgeb, ‘‘Comparison of policing mechanisms for ATM networks,’’ *Proceedings of the 3rd RACE Workshop*, (Oct. 1989).

[Reed87] D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, The MIT Press (1987).

[Reis83] M. Reiser, ‘‘Queueing and delay analysis of a buffer pool with resume level,’’ *9th International Symposium on Computer Performance Modelling, Measurement, and Evaluation*, (May 1983).

[Rett90] R. D. Rettberg, W. R. Crowther, P. P. Carvey, and R. S. Tomlinson, ‘‘The Monarch Parallel Processor Hardware Design,’’ *IEEE Computer* **23**(4) pp. 18-30 (April 1990).

[Rimo87] Y. Rimoni, I. Zisman, R. Ginosar, and U. Weiser, ‘‘Communication Element for the Versatile MultiComputer,’’ *15th IEEE Conference in Israel*, (April 1987).

[Scot90] S. L. Scott and G. S. Sohi, ‘‘The Use of Feedback in Multiprocessors and Its Application to Tree Saturation Control,’’ *IEEE Transactions on Parallel and Distributed Systems* **1**(4) pp. 385-398 (October 1990).

[Scot94] S. L. Scott and J. R. Goodman, ‘‘The impact of pipelined channels

on *k*-ary *n*-cube networks.,'' *IEEE Transactions on Parallel and Distributed Systems* **5**(1) pp. 2-16 (January 1994).

[Seit85] C. L. Seitz, ''The Cosmic Cube,'' *Communications of the ACM* **28**(1) pp. 22-33 (January 1985).

[Stev86] K. S. Stevens, S. V. Robinson, and A. L. Davis, ''The Post Office - Communication Support for Distributed Ensemble Architectures,'' *The 6th International Conference on Distributed Computing Systems*, pp. 160-166 (May 1986).

[Ston87] H. S. Stone, ''High-Performance Computer Architecture,'' *Addison-Wesley Publishing Company*, (1987).

[Stun94] C. B. Stunkel, D. G. Shea, B. Abali, M. M. Denneau, P. H. Hochschild, D. J. Joseph, B. J. Nathanson, M. Tsao, and P. R. Varker, ''Architecture and Implementation of Vulcan,'' *Proceedings of the 8th International Parallel Processing Symposium*, pp. 268-274 (April 1994).

[Swop86] S. M. Swope and R. M. Fujimoto, ''Simon II Kernel Reference Manual,'' Technical Report UUCS 86-001, University of Utah, Salt Lake City, UT (March 1986).

[Tami88a] Y. Tamir and J. C. Cho, ''Design and Implementation of High-Speed Asynchronous Communication Ports for VLSI Multicomputer Nodes,'' *International Symposium on Circuits and Systems*, pp. 805-809 (June 1988).

[Tami88b] Y. Tamir and G. L. Frazier, ''High-Performance Multi-Queue

Buffers for VLSI Communication Switches,'' *15th Annual International Symposium on Computer Architecture*, pp. 343-354 (May 1988).

[Tami88c] Y. Tamir and G. L. Frazier, ''Support for High-Priority Traffic in VLSI Communication Switches,'' *9th Real-Time Systems Symposium*, pp. 191-200 (December 1988).

[Tami92a] Y. Tamir and G. L. Frazier, ''Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches,'' *IEEE Transactions on Computers* **41**(6) pp. 725-737 (June 1992).

[Tami92b] Y. Tamir and G. L. Frazier, ''Hardware Support for High-Priority Traffic in VLSI Communication Switches,'' *Journal of Parallel and Distributed Computing* **14**(4) pp. 402-416 (April 1992).

[Tami93] Y. Tamir and H.-C. Chi, ''Symmetric Crossbar Arbiters for VLSI Communication Switches,'' *IEEE Transactions on Parallel and Distributed Systems* **4**(1) pp. 13-27 (January 1993).

[Turn86] J. Turner, ''New directions in communications,'' *IEEE Communications Magazine* **24**(10) pp. 8-15 (Oct. 1986).

[Whit85] C. Whitby-Strevens, ''The Transputer,'' *12th Annual Symposium on Computer Architecture*, pp. 292-300 (June 1985).

[Yew86] P.-C. Yew, N.-F. Tzeng, and D. H. Lawrie, ''Distributing Hot-Spot Addressing in Large-Scale Multiprocessors,'' *Proceedings of the 1986 International Conference on Parallel Processing*, pp. 28-34 (August, 1986).

[Yoon90] H. Yoon, K. Y. Lee, and M. T. Liu, ‘‘Performance Analysis of Multibuffered Packet-Switching Networks in Multiprocessor Systems,’’ *IEEE Transactions on Computers* **39**(3) pp. 319-327 (March 1990).

[Yum83] T. P. Yum and H.-M. Yen, ‘‘Design algorithm for a hysteresis buffer congestion control strategy,’’ *IEEE International Conference on Communications*, pp. 499-503 (June 1983).

[Zhan91] L. Zhang, ‘‘VirtualClock: a new traffic control algorithm for packet-switched networks,’’ *ACM Transactions on Computer Systems* **9**(2) pp. 101-24 (May 1991).