The Buffer Management in Wormhole-Routed Torus Multicomputer Networks

by

Kamala Kotapati
M.C.A.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario
November 1997

Abstract

The performance of a multicomputer network depends on the switching technique and the routing algorithm used. The wormhole switching technique has been widely used in the recent multicomputers. The most costly resources in a multicomputer network using wormhole routing are channel bandwidth and buffer memory. This thesis focuses on the *buffer management* issues in wormhole routed multicomputers.

The commonly used buffer organizations for recent multicomputers are centralized and dedicated buffer organizations. The results presented in this thesis indicate that the dedicated buffer organization has a better performance than the centralized in the uniform traffic. In the hot-spot traffic the centralized organization outperforms the dedicated. The *hybrid* buffer organization proposed in this thesis inherits the merits of the two. Hybrid buffer organization performs similar to the dedicated buffer organization under uniform traffic and its performance is intermediate to that of the two organizations for hot-spot traffic. The simulation results presented suggest that the hybrid buffer organization can be designed to configure itself dynamically from the dedicated to centralized depending on the traffic conditions.

Acknowledgments

I take this opportunity to express my sincere thanks to Dr. Sivarama Dandamudi for being an excellent supervisor and mentor. All the meetings with him on this work were very effective and focused, without which this thesis could not have been completed. I would also like to thank the School of Computer Science and Carleton University for giving me the opportunity to work and providing me with financial support.

I express my gratitude to my parents for teaching me to work hard to meet any goal in life. I thank my husband for being very supportive and optimistic. I also thank several of my friends and well wishers whose presence made my life at school and home very pleasant and interesting.

And, I dedicate this thesis as a tribute to the memories of my beloved uncle **Kotapati Koteswara Rao garu**. He always had a very positive and rational approach towards life and had left unforgettable memories to rest of the family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The problem domain to which computers could be applied, is expanding rapidly. There is almost no field of study which is not taking advantage of computers. As computers are used to solve many complex problems, the demand for larger and faster computer systems is also increasing steadily.

In order to meet this requirement and to manufacture computers with increased speed and capacity for lower cost, computer architects have followed two general approaches [4]. The first uses conventional serial computer architecture. The second approach exploits the parallelism inherent in many problems.

Data Parallelism found in data takes advantage of large number of independent data elements. For each data element a processor is assigned and all the operations are performed on the data in parallel.

Control Parallelism is used to program most multiprocessor computers. In this approach threads of control that could operate independently are identified and then different processors are used to run these threads in parallel. The primary problems of this approach are the difficulty in identifying and synchronizing these independent

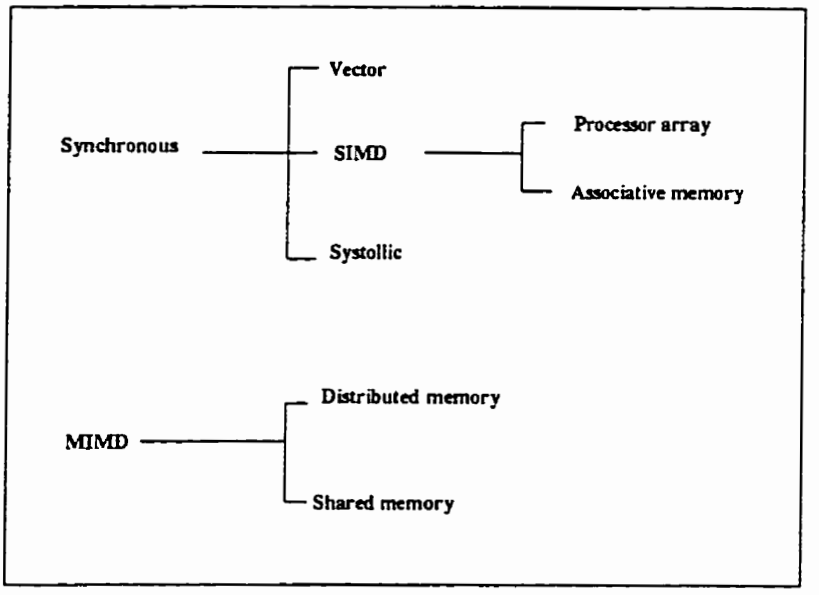


Figure 1.1: High-level taxonomy of parallel computer architectures [5]

threads of control.

1.1 Parallel Architectures

A parallel architecture provides an explicit, high-level framework for the development of parallel programming solutions by providing multiple processors, whether simple or complex, that cooperate to solve problems through concurrent execution [5]. A simple taxonomy of parallel computer architectures based on this definition is shown in Figure 1.1.

1.1.1 Synchronous architectures

Synchronous parallel architectures support data parallelism. These architectures coordinate concurrent operations in lockstep through global clocks, central control units, or vector unit controllers.

Pipelined vector processors. Pipelined vector processors directly support massive vector and matrix calculations [6]. These machines are characterized by multiple, pipelined functional units, which implement arithmetic and boolean operations for both vectors and scalars and which can operate concurrently. Cray-1, Fujitsu VP-200, Data Cyber 205 and Texas Instruments Advanced Scientific Computer are example machines of this architecture.

SIMD architectures. SIMD architectures typically employ a central control unit, multiple processors, and an interconnection network for either processor-to-processor or processor-to-memory communications. The control unit broadcasts a single instruction to all processors, which execute the instruction in lockstep fashion on local data. The interconnection network allows instruction results calculated at one processor to be communicated to another processor for use as operands in a subsequent instruction.

Processor array architectures are structured for numerical SIMD execution and have often been employed for large-scale scientific calculations, such as image processing and nuclear energy modeling [7]. Loral's Massively Parallel Processor, ICL's Distributed Array Processor and Thinking Machines' Connection Machine (CM1 and CM2) exemplify this kind of architecture.

Associate memory processor architectures are built around an associative memory and constitute a distinctive type of SIMD architecture that uses special comparison logic to access stored data in parallel according to its content [8]. Bell Laboratories' Parallel Element Processing Ensemble and Loral's Associative Processor are the machines developed using this architecture and are used for database-oriented applications.

Systolic architectures. These architectures are used to build machines to solve the problems of special-purpose systems [9]. These computers are pipelined multi-

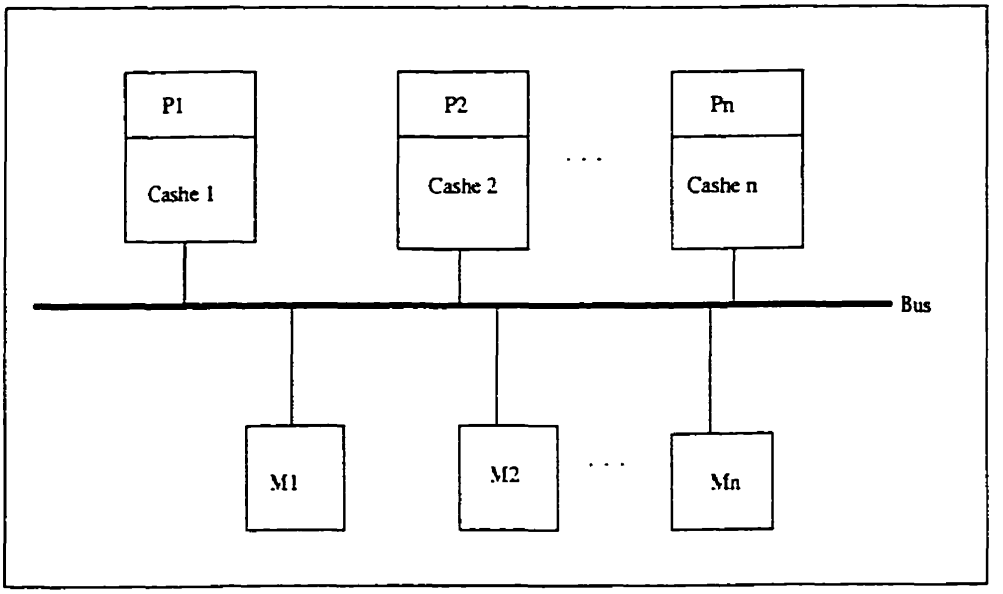


Figure 1.2: MIMD-shared memory architecture: bus interconnection [5]

processors in which data is pulsed in rhythmic fashion from memory and though a network of processors before returning to memory. A global clock and explicit timing delays synchronize this pipelined data flow, which consists of operands obtained from memory and partial results to be used by each processor. Carnegie Mellon's Warp and Saxpy's Matrix-1 fall into this category.

1.1.2 MIMD architectures

MIMD architectures employ control parallelism i.e., they use multiple processors that execute independent instruction streams using local data. MIMD computers support parallel solutions that require processors to operate in largely autonomous manner. The software processes executing on MIMD architectures are synchronized by accessing data in shared memory units or by passing messages though an interconnection network. MIMD architectures are asynchronous computers and are characterized by a decentralized hardware control. MIMD architectures can be divided into shared-memory and distributed-memory systems.

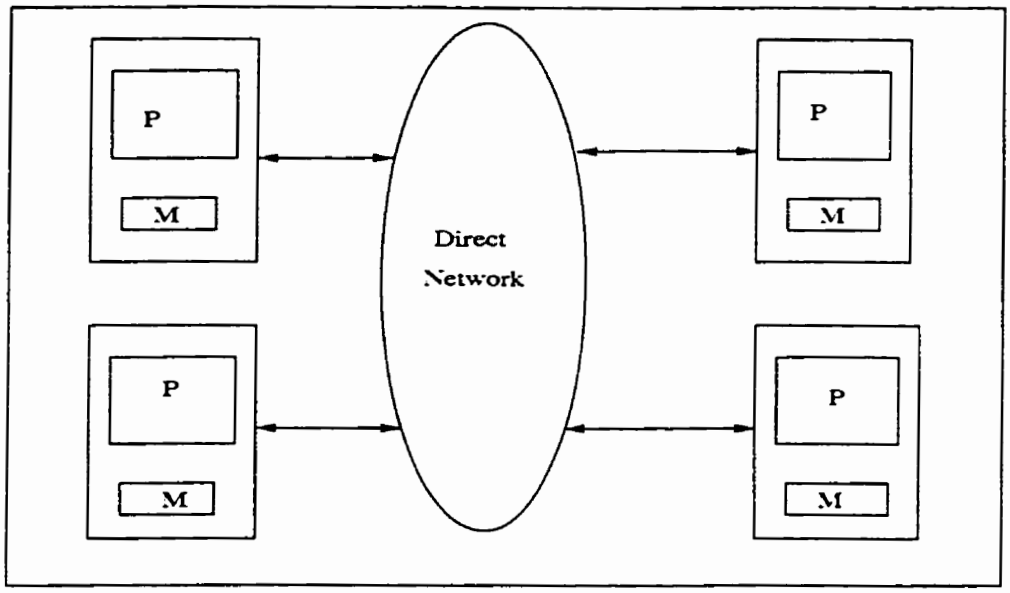


Figure 1.3: Structure of MIMD-distributed memory architecture [5]

Shared-memory architectures

Shared memory architectures (Figure 1.2) accomplish interprocessor coordination by providing a global, shared memory that each processor can address. Commercial shared-memory architectures, such as Flexible Corporation's Flex/32 and Encore Computer's Multimax are examples for this category. These machines have no massage sending latency problem but suffer from data access synchronization and cache coherency problems. These computers are also referred to as *multiprocessor systems.*

Distributed-memory architectures

Distributed memory architectures (Figure 1.3) connect processing nodes, consisting of an autonomous processor and its local memory, with a processors-to-processor interconnection network. Nodes share data by explicitly passing messages though the interconnection network, since there is no shared memory. These systems are also referred as *multicomputer systems.* As the scope of this thesis is multicomputer networks, the remainder of the thesis focuses on this type of networks.

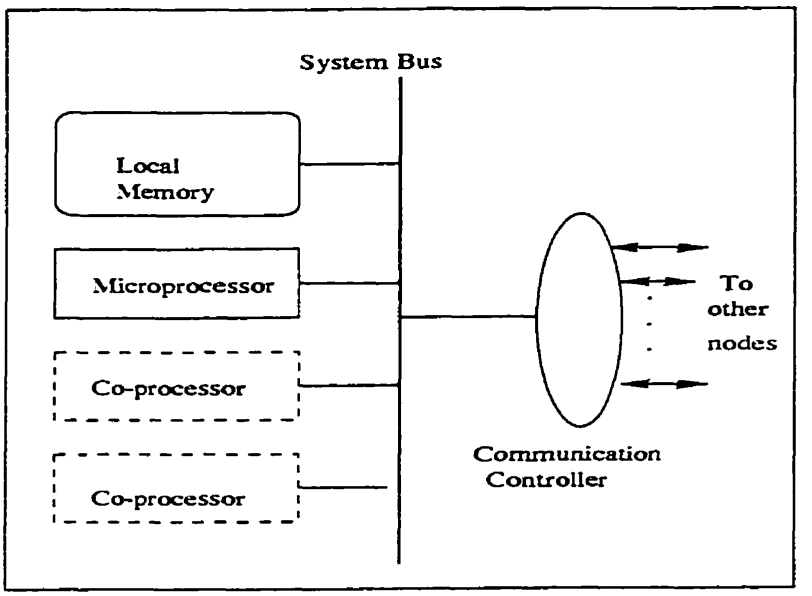


Figure 1.4: A typical multicomputer node [1]

1.2 Multicomputer Networks

Multicomputer architecture has become a popular architecture for constructing massively parallel computers because they scale well; i.e., as the number of nodes in the system increases, the total communication bandwidth, memory bandwidth, and processing capability of the system also increase. Several examples for these machines are given in the subsequent sections.

A *multicomputer network* consists of hundreds or thousands of *nodes* connected in some fixed *topology* [1]. As shown in Figure 1.4, a multicomputer node minimally contains a microprocessor, local memory, and hardware support for inter-node communication. Specific applications may dictate inclusion of specialized co-processors for floating point, graphics, or secondary storage operations. Ideally, each node would be directly connected to all other nodes (fully connected or complete network). The packaging constrains and hardware costs limit the number of connections to a small number, typically ten or less. Because the node degree is limited, messages must often be routed though a sequence of intermediate nodes (multiple-hop networks) to reach their final destinations.

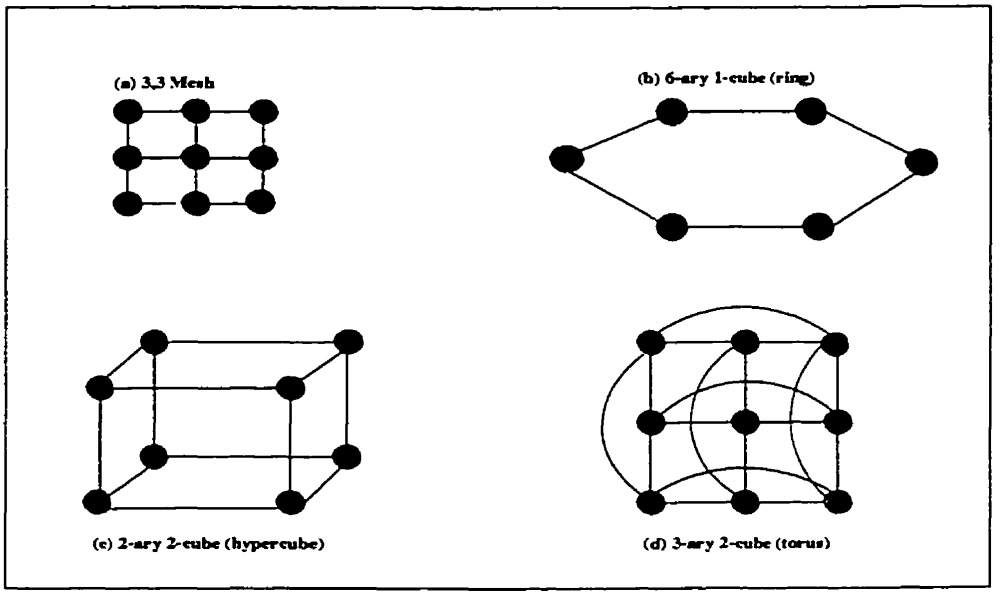


Figure 1.5: Multicomputer network topologies

In summary, multicomputers have large numbers of nodes executing instructions asynchronously. The communication between processors is message based with moderate overhead. The user implemented programs are decomposed into medium grained computation tasks. A multicomputer network is characterized by four factors: topology, routing, flow control and switching [5]. There are also other factors that can be considered.

1.2.1 Topology

The topology of a network, usually modeled as a graph, defines how the nodes are interconnected by channels. Two requirements of a network are that it must accommodate a large number of nodes and exhibit a low network latency [3]. There are many ways to interconnect nodes in multicomputer networks. Most of the popular direct network topologies fall in the general category of either n-dimensional meshes or k-ary n-cubes because their regular topologies and simple routing. Some sample topologies are shown in Figure 1.5.

Mesh

A n-dimensional mesh has $k_0 \times ... \times k_{n-2} \times k_{n-1}$ nodes, k_i nodes along each dimension i, where $k_i \geq 2$. Each node X is identified by n coordinates, $\sigma_{n-1}(X), \sigma_{n-2}(X), ..., \sigma_1(X)$, $\sigma_0(X)$, where $0 \leq \sigma_i(X) \leq k_i - 1$ for $0 \leq i \leq n-1$. Two nodes X and Y are neighbors if and only if $\sigma_i(X) = \sigma_i(Y)$ for all i, $0 \leq i \leq n-1$, except one, j, where $\sigma_j(Y) = \sigma_j(X) \pm 1$. Thus, nodes have from n to $2n$ neighbors, depending on their location in the mesh (Figure 1.5(a)).

k-ary n-cubes

In a k-ary n-cube, all nodes have the same number of neighbors. The definition of a k-ary n-cube differs from that of an n-dimensional mesh in that all the k_i's are equal to k and two nodes X and Y are neighbors if and only if $\sigma_i(X) = \sigma_i(Y)$ for all i, $0 \leq i \leq n-1$, except one, j, where $\sigma_j(Y) = (\sigma_j(X) \pm 1) \bmod k$. The use of modular arithmetic in the definition results in *wraparound* channels in the k-ary n-cube, which are not present in the n-dimensional mesh. A k-ary n-cube contains k^n nodes.

Several special cases of the n-dimensional meshes and k-ary n-cubes have been proposed or implemented as multicomputer network topologies. When n=1, the k-ary n-cube collapses to a *ring* with k nodes (Figure 1.5(b)). When n=2, the topology is a 2D *torus* with k^2 nodes (Figure 1.5(d)). A *hypercube* is an n-dimensional mesh in which $k_i = 2$ for all i, $0 \leq i \leq n-1$, that is, a 2-ary n-cube (Figure 1.5(c)).

Hypercubes, low-dimensional meshes, and tori can be compared in terms of their bisection width η, that is, the minimum number of channels that must be removed to partition the network into two equal subnetworks. If $N = 2^{2n}$ nodes are present in each topology network then $\eta_{hypercube} = 2^{2n-1}$, $\eta_{2Dmesh} = 2^n$, and $\eta_{2Dtorus} = 2^{n+1}$, respectively. The bisection density is the product of η and the channel width W and can be used as a measure of the network cost. If all the networks have the same

bisection density, then
$W_{2Dmesh}/W_{hypercube} = 2^{n-1} = \sqrt{N/2}$
and
$W_{2Dtorus}/W_{hypercube} = 2^{n-2} = \sqrt{N/4}$.

In other words, for the same cost, the 2D mesh and 2D torus can support wider channels and there by offer higher channel bandwidth. However, the low-dimensional networks have larger diameters, which is the maximum distance between two nodes. The diameters of the topologies are $2n$, $2^{n+1} - 2$, and $2^n - 1$, respectively.

Both the hypercube and the torus are symmetric networks in that there exists a homomorphism that maps any node of the graph representing the network graph onto any other node. Mesh networks, on the other hand, are asymmetric because the wraparound channels are absent.

The Intel Touchstone Delta, Cray T3E, and Symult 2010 use a 2D mesh; the MIT J-machine and Caltech's Mosaic use a 3D mesh; and the Ncube-2/3 uses a hypercube.

1.2.2 Routing

In the absence of the complete network, *routing* determines the path selected by a message to reach its destination. Efficient routing plays a very important role for the performance of the multicomputer networks. For maximum system performance, a routing algorithm should have high throughput and exhibit following important features: low-latency message delivery, avoidance of deadlocks, live-locks and starvation and ability to work well under various traffic patterns.

Routing can be classified into *deterministic* or *adaptive*. In *deterministic routing*, the path is completely determined by the source and destination addresses. This

method is also referred to as oblivious routing. The well-known e-cube routing algorithm (discussed in Chapter 2) is an example of deterministic routing.

In *adaptive routing* for a given source and destination, the path taken by a particular packet depends on dynamic network conditions, such as the presence of faulty or congested channels. A routing algorithm is said to be *minimal*, if the path selected is one of the shortest paths between the source and destination pair. Using a minimal routing algorithm, every channel visited will bring the message closer to the destination. A *non-minimal* routing algorithm allows messages to follow a longer path, usually in response to current network conditions. If *non-minimal* routing is used, care must be taken to avoid a situation in which the message will continue to be routed though the network but never reach the destination (live lock).

A minimal fully-adaptive algorithms do not impose any restrictions on the choice of shortest paths to be used in routing messages; in contrast, partially adaptive minimal algorithms allow only a subset of available minimal paths in routing messages. An adaptive routing algorithm can be fully- or partially-adaptive. *-channel and Nhop routing algorithms are examples for fully-adaptive routing algorithms (discussed in Chapter 2).

1.2.3 Flow control

A network consists of many channels and buffers. Flow control deals with the allocation of channels and buffers to a message as it travels along a path though the network. A resource collision occurs when a message cannot proceed because some resource that it requires is held by another message. Whether the message is dropped, blocked in place, buffered, or rerouted though another channels depends on the flow control policy. A good flow control policy should avoid channel congestion while reducing the network latency.

The allocation of channels and their associated buffers to messages can be viewed from two perspectives. The routing algorithm determines which output channel is selected for a message arriving on a given input channel. Therefore, routing can be referred to as the *output selection policy*. Since an outgoing channel can be requested by messages arriving on many different input channels, an *input selection policy* is needed to determine which packet may use the output channel. Possible input selection policies include round robin, fixed channel priority, and first come, first served. The input selection policy affects the fairness of routing algorithms.

1.2.4 Switching

Switching is the actual mechanism that removes data from an input channel and places it on an output channel. Network latency is highly dependent on the switching technique used. Four switching techniques have been adopted in multicomputer networks: store-and-forward, circuit switching, virtual cut-through, and wormhole routing.

Store-and-forward

In store-and-forward switching, when a message reaches an intermediate node, the entire message is stored in a message buffer. The message is then forwarded to a selected neighboring node when the next output channel is available and the neighboring node has an available buffer. This switching strategy was adopted in the research prototype Cosmic Cube and several first-generation commercial multicomputers, including the iPSC-1, Ncube 1, Ametek 14, and FPS T-series.

Store-and-forward technique is simple, but it has two major drawbacks. First, each node must buffer every incoming message, consuming memory space. Second, the network latency is proportional to the distance between the source and destination nodes. The network latency is $(L/B)D$, where L is the message length. B

is the channel bandwidth, and D is the length of the path between the source and destination.

Virtual cut-through

In virtual cut-through, a message is stored at an intermediate node only if the next required channel is busy. The network latency is $(L_h/B)D + L/B$ where L_h is the length of the header field. This is true only if there is no contention for channels along the paths. When $L >> L_h$, the second term, L/B, will dominate, and the distance D will produce a negligible effect on the network latency. The research prototype Harts, developed at University of Michigan, is a hexagonal mesh multicomputer that adopts virtual cut-through switching mechanisms.

Circuit switching

In circuit switching, a physical circuit is constructed between the source and destination nodes during the circuit establishment phase. In the message transmission phase, the message is transmitted along the circuit to the destination. During this phase, the channels constituting the circuit are reserved exclusively for the circuit; hence, there is no need for buffers at intermediate nodes. In the circuit termination phase, the circuit is torn down as the tail of the message is transmitted.

The network latency for circuit switching is $(L_c/B)D + L/B$, where L_c is the length of the control packet transmitted to establish the circuit. If $L_c << L$, the distance D has a negligible effect on the network. If a circuit cannot be established because a desired channel is being used by other messages, the circuit is said to be blocked. Depending on the way blocked circuits are handled, the partial circuit may be torn down, with establishment to be attempted later. This policy is called *loss mode*. Some second-generation multicomputers, such as Intel's iPSC-2 and iPSC/860, used circuit switching.

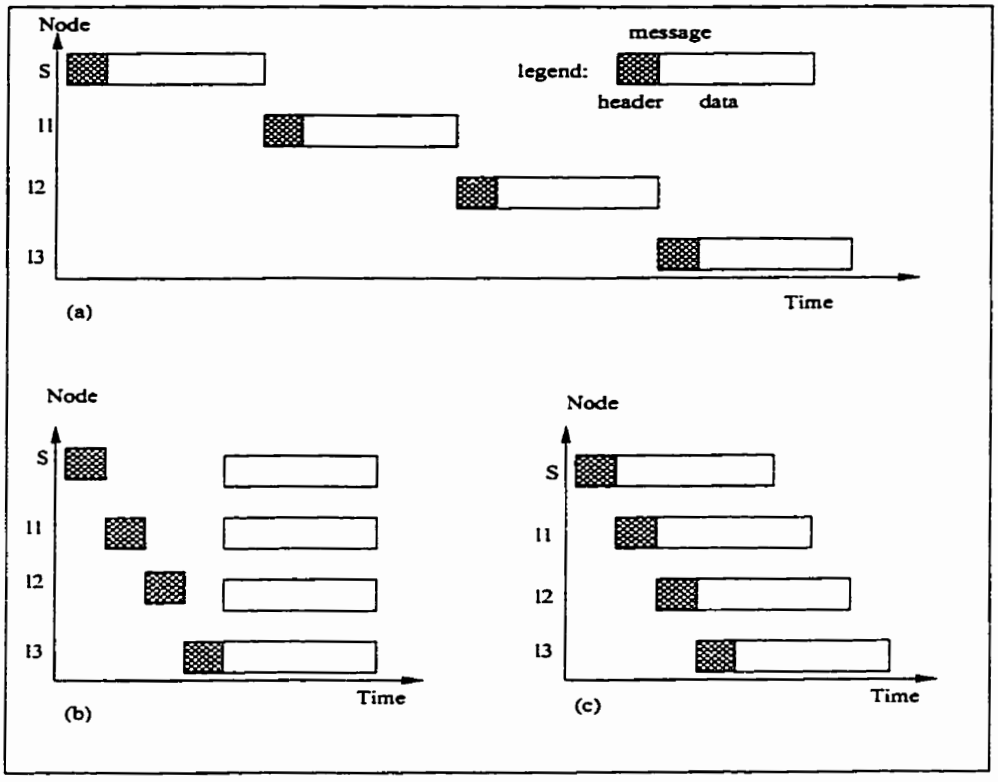


Figure 1.6: Comparison of different switching techniques: (a) store-and-forward (b) circuit (c) wormhole [5]

Unlike store-and-forward switching, both virtual cut-through and circuit switching offer low network latencies that are relatively independent of path length. The virtual cut-through requires that blocked messages be buffered, and circuit switching makes it difficult to support sharing of channels among messages.

Wormhole routing

In wormhole routing a message is divided into a number of *flits* (flow control digits) for transmission. The header flit governs the route. As header advances along the specified route, the remaining flits follow in a pipeline fashion. If the header flit encounters a channel already in use, it is blocked until the channel becomes available. Rather than buffering the remaining flits by removing them from the network

channels, as in virtual cut-though, the flow control within the network blocks the trailing flits and they remain in flit buffers along the established route. The network latency for wormhole routing is $(L_f/B)D + L/B$, where L_f is the length of each flit, B is the channel bandwidth, D is the path length, and L is the length of the message. If $L_f << L$, the path length D will not significantly affect the network latency unless it is very large.

Figure 1.6 compares the compares the communication latency of wormhole routing with that of store-and-forward switching and circuit switching in a contention-free network. In this case, the behavior of virtual cut-though is the same as that of wormhole routing, so virtual cut-though is not shown explicitly. The channel propagation delay is typically small relative to L/B and is ignore here. The figure shows the activities of each node over time when a packet is transmitted from a source node S to the destination node D through three intermediate nodes, I1, I2, I3. The time required to transfer the packet between the source processor and its router, and between the last router and the destination processor, is ignored. Unlike store-and-forward switching, both circuit switching and wormhole routing have communication latencies that are nearly independent of distance between the source and destination nodes.

1.3 Goals and Contributions of the Thesis

The goal of this research work is to study various buffer management strategies in Wormhole-routed multicomputer networks. To this end, the torus network is used to study the performance of buffer organizations. The buffer organizations considered are 1) centralized, 2) dedicated and 3) hybrid (described in detail in Chapter 3).

During this research work it has been observed that in hot-spot traffic with constant number of buffers, the centralized buffer organization results in a better performance (higher utilization and lower latency) than the dedicated organization.

Under uniform traffic dedicated buffer organization performs better.

In practice, the traffic is bounded between hot-spot and uniform traffic. Therefore, it is necessary to handle these two cases in an efficient way. A new buffer organization, called hybrid buffer organization, is proposed in this thesis. This new organization can act as the centralized for hot-spot traffic and it can work like the dedicated organization for uniform traffic. In hybrid organization, the total number of buffers n is partitioned into p $(0 \leq p \leq n)$ dedicated buffers and $n-p$ centralized buffers. And, by changing p appropriately depending on the traffic pattern it is possible to achieve a better performance by switching dynamically between centralized and dedicated buffer organizations.

Chapter 2 provides necessary background. In this chapter the basic concepts of wormhole routing such as deadlocks, virtual channels, deterministic and adaptive routing are discussed. This chapter also described nCUBE system as an example massively parallel computer.

Chapter 3 discusses different issues of buffer management. The centralized, dedicated, and hybrid buffer organizations are described in detail.

Chapter 4 describes the system model used in the simulation experiments to evaluate the relative performance of the buffer organizations.

Chapter 5 gives the results of the simulations. In this chapter an analysis of the results obtained is also given.

Finally, the conclusions , contributions and future work of this research are given in Chapter 6.

Chapter 2

Background

2.1 Wormhole Routing

In wormhole routing [5] a message (packet) is divided into a number of *flits* for transmission. The size of a flit depends on system parameters, in particular the channel width. Normally, the bits constituting a flit are transmitted in parallel between two routers. The header flit (or flits) of the message governs the route. As the header advances along the specified route, the remaining flits follow in a pipeline fashion, as shown is Figure 2.1. If header flit encounters a channel already in use, it is blocked until the channel becomes available. The flow control within the network blocks the trailing flits and they remain in flit buffers along the established route. Once a channel has been acquired by a packet, it is reserved for the packet. The channel is released when the last, or tail, flit has been transmitted on the channel.

The pipelined nature of wormhole routing produces two positive effects. First, in the absence of network contention, network latency is relatively insensitive to path length. Secondly, large packet buffers at each intermediate node are obviated; only small FIFO flit buffer is required. When the flit buffers are as large as the messages themselves, the behavior wormhole routing resembles that of virtual cut-through.

Flits passing between two adjacent nodes use a handshaking protocol. In the

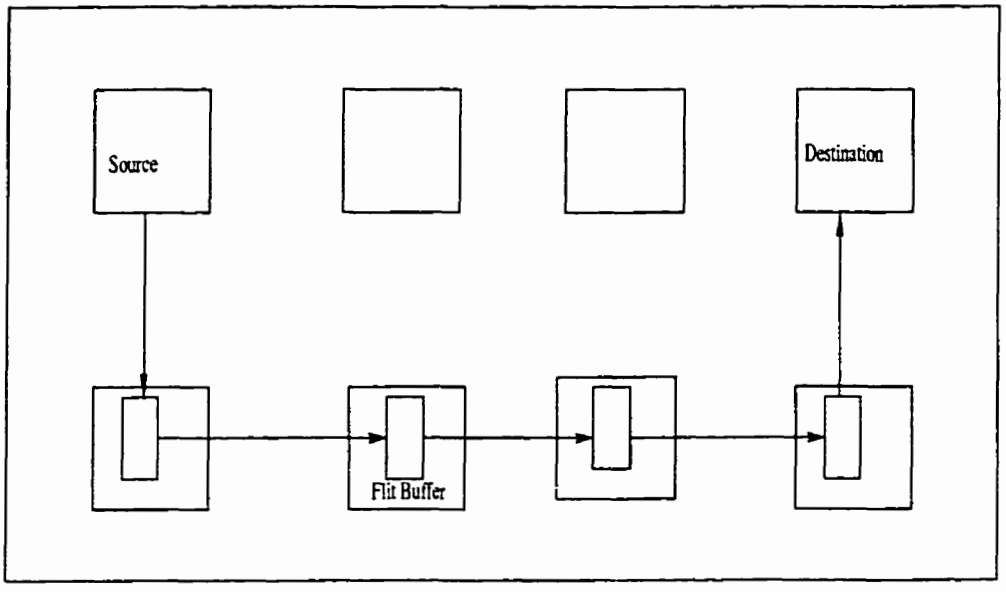


Figure 2.1: Wormhole Routing

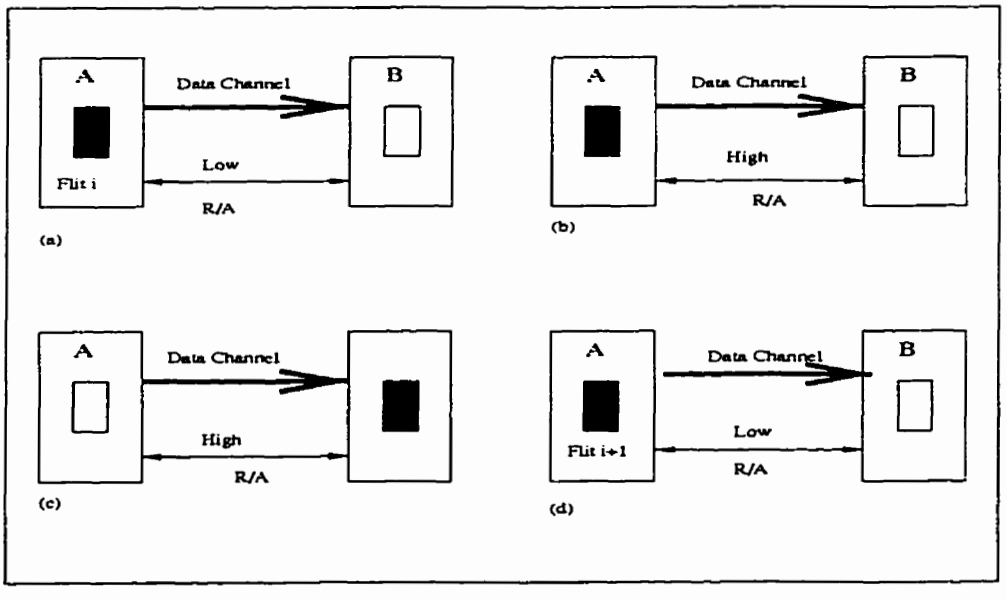


Figure 2.2: Handshaking between two routers through a request/acknowledge line:(a) B is ready to accept a flit by setting R/A to low; (b) A is ready ready to send flit i by raising R/A to high; (c) flit is latched in B's flit buffer; (d) B sets R/A to low when flit i is removed (also A has received flit i+1) [5]

example in Figure 2.2, a unidirectional channel from router A connects to router B. A single-wire request/acknowledge (R/A) line is associated with the channel. The R/A line can be raised only by router A, the requesting side, and lowered only router B, the acknowledging side. When A is ready to send a flit to B, A must wait unit the R/A line is low. A then places the data on the data channel and raises the R/A line to high. Router B will lower the R/A line when it has removed the flit from the flit buffer.

In circuit switching, once a channel is assigned to a packet, it cannot be used by other packets until the channel is released. In contrast, wormhole routing allows a channel to be shared by many packets. Wormhole routing also allows packet replication, in which copies of flit can be sent on multiple output channels. Packet replication is useful in supporting broadcast and multi-cast communication.

Wormhole switching technique has been a popular switching technique in new-generation multicomputer networks. The first commercial multicomputer to adopt wormhole routing was Ametek 2010, which used a 2D mesh topology. The Ncube-2, which uses a hypercube topology, has also adopted wormhole routing. The Intel Touchstone Delta and Inter Paragon use wormhole routing in a 2D mesh. Finally, MIT's research prototype J-machine uses wormhole routing in a 3D mesh.

2.2 Deadlocks

A multicomputer network is said to be in a *deadlock* condition when no message can advance towards its destination. This situation can postpone message delivery indefinitely [17]. Deadlock can occur if messages are allowed to hold some resource while requesting others. Consider the example shown in Figure 2.3 where channel deadlock occurred between four routers and four messages. Each message is holding a flit buffer while requesting the flit buffer being held by another message. In this locked state, no communication can occur over the deadlocked channels until

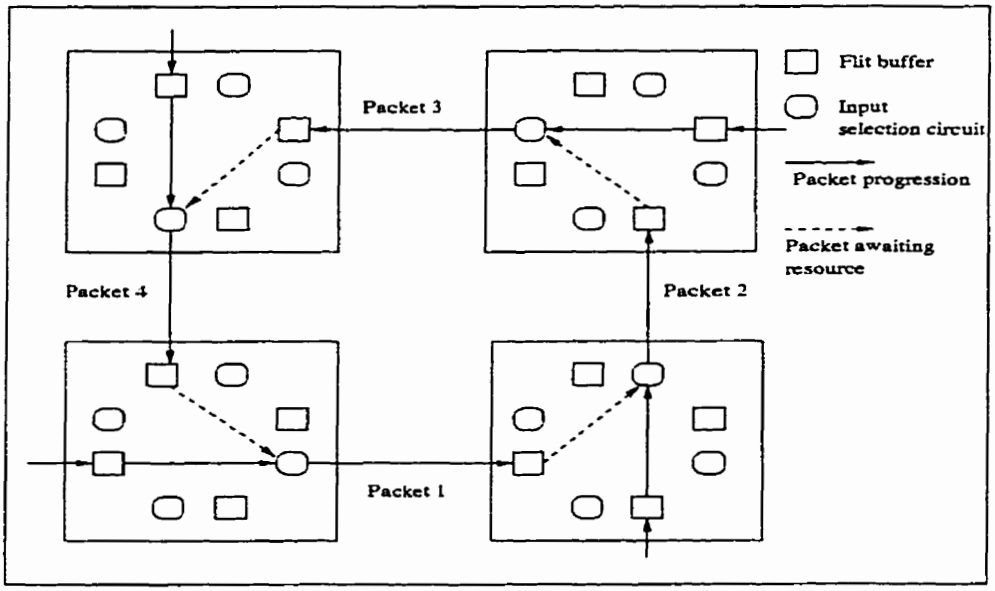


Figure 2.3: An example of channel deadlock involving four packets [5]

exceptional action is taken to break the deadlock.

In store-and-forward and virtual cut-through switching, the resources are buffers. In circuit switching and wormhole routing, the resources are channels. Because blocked packets holding channels (and their corresponding flit buffers) remain in the network, wormhole routing is particularly susceptible to deadlock.

Livelock is a different situation in which some messages are not able to reach their destination, even if they never block permanently [2]. A message may be traveling around its destination node, never reaching it because the channels required to do so are occupied by other messages. This situation can occur when messages are allowed to follow non-minimal paths.

2.2.1 Channel dependency graphs

A *channel dependency graph* can be used to develop deadlock free routing algorithms [17]. Channel dependence graph for a multicomputer network and a routing algo-

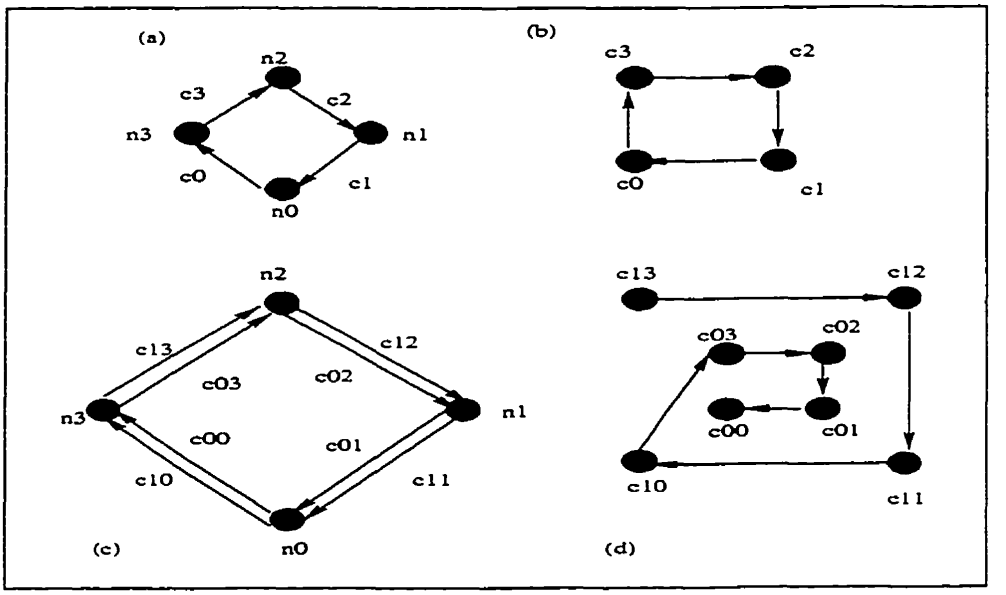


Figure 2.4: Breaking deadlock by adding virtual edges

rithm is a directed graph D=G(C,E), where the vertex set C(D) consists of all the unidirectional channels in the network, and the edge set E(D) includes all the pairs of connected channels, as defined by the routing algorithm. In other words, if (a,b) belongs to E(D), then a and b are, respectively, an input channel and an output channel of a node, and the routing algorithm may route messages from a to b. A routing algorithm for a multicomputer network is deadlock free if and only if there is no cycle in the channel dependence graph. Consider, for example, the case of a uni-directional four-cycle shown in Figure 2.4(a), the corresponding channel dependence graph is shown in 2.4(b). Since there is a cycle in the channel dependence graph, deadlock is possible. One way to break the deadlock is to split the channels into two classes of *virtual channels* for each channel. If C_0 is chosen as the dividing channel of the cycle and split each channel into high virtual channels, $C_{10}...C_{12}$, and low virtual channels, $C_{00},..,C_{03}$, as shown in Figure 2.4(c).

Messages at a node numbered less than their destination are routed on the high channels, and messages at a node greater than their destination node are routed

on the low channels. Channel C_{00} is not used. Now a total ordering of the virtual channels can be obtained according to their subscripts: $C_{13} > C_{12} > C_{11} > C10 > C_{03} > C_{02} > C_{01}$. Thus, there is no cycle in D and the routing function is dead-lock free Figure 2.4(d).

2.3 Virtual Channels

A virtual channel consists of a buffer that can hold one or more flits of a message and associated state information [16]. Several virtual channels may share the bandwidth of a single physical channel. Virtual channels separate the allocation of buffers from allocation of channels by providing multiple buffers for each channel in the network. If a blocked packet A holds a buffer b_{i0} associated with channel c_i, another buffer b_{i1} is available allowing other messages to pass A. Figure 2.5(b) illustrates the addition of virtual channels to the network of Figure 2.5(a). In this figure each buffer is represented as $KD.i$, where K is the number of the node, D is the direction, and i is the buffer number. Message A remains blocked holding buffers 3E.1 and 4S.1. In Figure 2.5(b), however message B is able to make progress because buffer 3E.2 is available allowing it access to channel (3E to 4W).

Virtual channels were introduced for purpose of deadlock avoidance. In addition, they increase throughput and additional degree of freedom in allocating resources to messages in the network. The most costly resource in a multicomputer network is the physical channel bandwidth. The second most costly resource is the buffer memory. Adding virtual channel flow control to a network makes more effective use of both of these resources by decoupling their allocation. The only expense is a small amount of additional control logic.

2.3.1 Buffer organization

Each node of a multicomputer network (or interconnection network) contains a set of buffers and a switch. If buffers are partitioned into sets associated with each

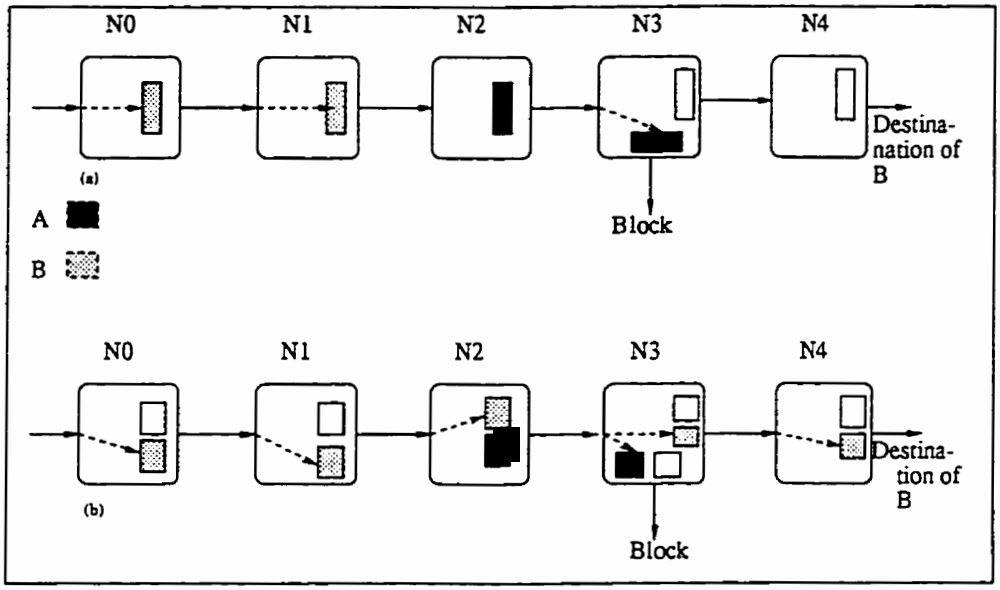


Figure 2.5: (a) Packet B is blocked behind packet A while all physical channels remain idle. (b) Virtual channels provide additional buffers allowing packet B to pass blocked packet A [16]

input channel, an input-buffered node looks as shown in Figure 2.6.

A conventional network organizes the flit buffers associated with each channel into a first-in, first-out (FIFO) queue as shown in Figure 2.7(a). This organization restricts allocation so that each flit buffer can contain only flits from a single message. If this message gets blocked, the physical channel is idle as no other message is able to acquire the buffer resources needed to access the channel.

A network using virtual channel flow control organizes the flit buffers associated with each channel into several lanes as shown in Figure 2.7(b). The buffers in each lane can be allocated independently of the buffers in any other lane. This added allocation flexibility increases channel utilization and thus throughput. A blocked message, even one that extends though several nodes, holds only a single lane idle and can be passed using any of the remaining lanes.

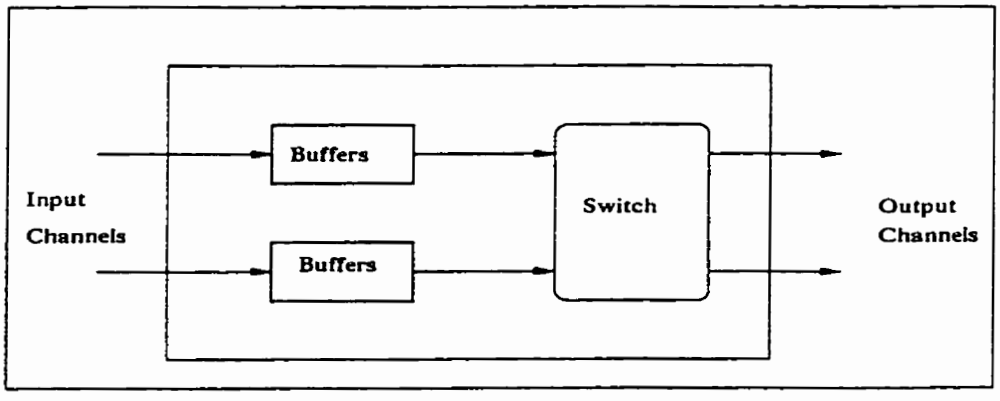


Figure 2.6: Node organization

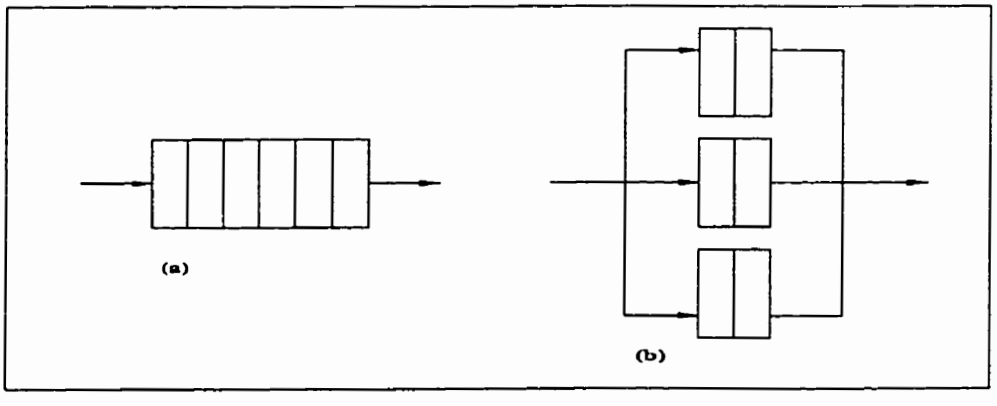


Figure 2.7: (a) Conventional organization. (b) Virtual-channel flow control organization [16]

2.3.2 Flow control

In a network using virtual channel flow control, flow control is performed at two levels. Virtual channel assignment is made at the message level while physical channel bandwidth is allocated at the flit level. When a message arrives at a node, it is assigned (according to the routing algorithm) to an output virtual channel. This assignment remains fixed for the duration of the message. The virtual channels associated with a physical channel arbitrate for physical channel bandwidth on a flit-by-flit basis.

2.3.3 Operation

Figure 2.8 illustrates the hardware required to support virtual channel flow control on one physical channel. The transmitting node (node A) contains a status register for each virtual channel that contains the state of the lane buffer on the receiving node (node B). This state information includes: bit to indicate if the lane is free, a count of the number of free flit buffers in the lane, and optionally the priority of message occupying the lane. Node B contains a lane buffer and a status register for each virtual channel. The status maintained on node B includes input and output pointers for each lane buffers and the state of channel: free, waiting (to be assigned an output), and active.

Lane assignment for physical channel P is performed by node A. When a message arrives in an input buffer on node A (not shown), it is assigned a particular output channel based on its destination, the status of the output channels, and the routing algorithm in use. The flow-control logic then assigns this message to any free lane of the selected channel. If all lanes are in use, the message is blocked in the waiting state until a lane is available. Maintaining lane state information on the transmitting end of the channel allows lane assignment to be performed on a single node. No additional inter node communication is required to maintain this information as it is already required for flit-level flow control.

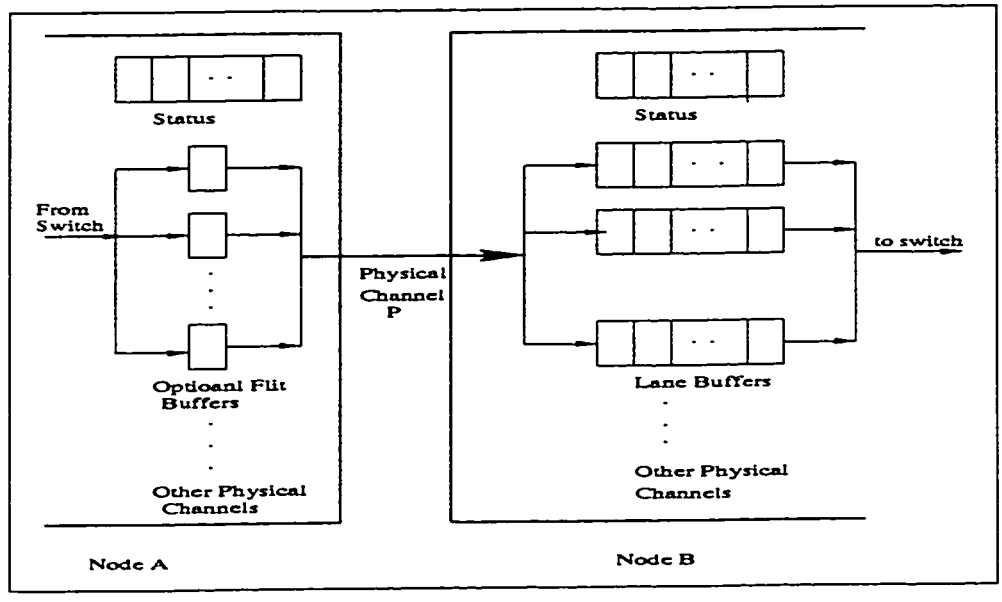


Figure 2.8: Logic associated with one physical channel P to support virtual channel flow control [16]

Once a lane is assigned to a packet, flit-level flow control is used to advance the message across the switch and physical channel. To advance from an input buffer on the node A to an input buffer on node B, a flit must gain access to 1) a path through the switch to reach the output of the node A, and 2) the physical channel to reach the input of node B. Typically entire switch is nonblocking, and thus always available, or a few optional flit buffers are provided at the output of node A so that switch and channel resources do not have to allocated simultaneously.

When the last flit of the message (the tail flit) leaves a node the lane assigned to that message is deallocated and may be reassigned to another message.

2.3.4 Physical channel allocation

Flit-level flow control across the physical channel involves allocating channel bandwidth among lanes that 1) have a flit ready to transmit and 2) have space for this

flit at the receiving end. Any arbitration algorithm can be used to allocate this resource including random, round-robin, or priority. For each physical channel, the arbitration algorithm is implemented as combinatorial logic that operates on the contents of the status registers and picks the highest priority lane that has space available at the receiving end. For random and round-robin arbitration schemes, priority information is generated by logic based on the lane's position and the previous state. For priority based schemes, priority information is stored in the status register for each lane.

2.4 Deterministic Routing

Another way to design a deadlock-free routing algorithm for a wormhole-routed network is to ensure that cycles are avoided in the channel dependence graph. This can be achieved by assigning each channel a unique number and allocating channels to messages in strictly ascending (or descending) order [17]. If the behavior of the algorithm is independent of current network conditions, it is deterministic. The e-cube routing algorithm is an example for this class of routing algorithms.

2.4.1 e-cube routing

In e-cube routing algorithm, messages are routed in order of dimension, most significant dimension first. In each dimension, i, a message is routed in that dimension until it reaches a node whose subscript matches the destination address in ith position. The physical channel is divided into an upper and lower virtual channel. The message is routed on the high channel if the ith digit of the destination address is greater than the ith digit of the present node's address. Otherwise, the message is routed on the low channel. As this routing algorithm routes messages in order of descending subscripts, it is deadlock free.

Formally, the definition of routing function R_{KNC} for k-ary n-cubes with C channels is given below:

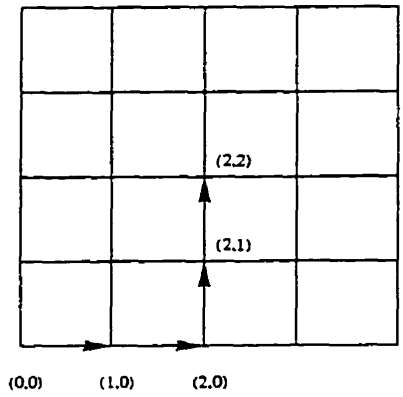


Figure 2.9: Example for the e-cube routing on a 4-mesh

$R_{KNC}(c_{dvx}, n_j)$
$= cd1(x - k^d) \quad if(dig(x,d) < dig(j,d))and(dig(x,d) \neq 0),$
$= cd0(x - k^d) \quad if(dig(x,d) > dig(j,d))or(dig(x,d) = 0),$
$= ci1(x - k^d) \quad if(\forall k > i, dig(x,k) = dig(j,k))and(dig(n,i) \neq dig(j,i)).$

Where dig (x,d) extracts the dth digit of x, and k is the radix of the cube. The subtraction, $x - k^d$, decrements the dth digit of x modulo k.

The e-cube routing is illustrated in Figure 2.9. For a message to go from (0,0) to (2,2), the message moves in X-axis direction until it reaches (2,0) and then moves on Y-axis.

2.5 Adaptive Routing

The main disadvantage of the deterministic routing is that it cannot respond to dynamic network conditions such as congestion. An adaptive routing algorithm for a wormhole-routed network, however, must ensure that it is deadlock free. To do so often requires the use of additional channels; in particular, virtual channels. The virtual channels may share one or more physical channels. *-Channels algorithm is

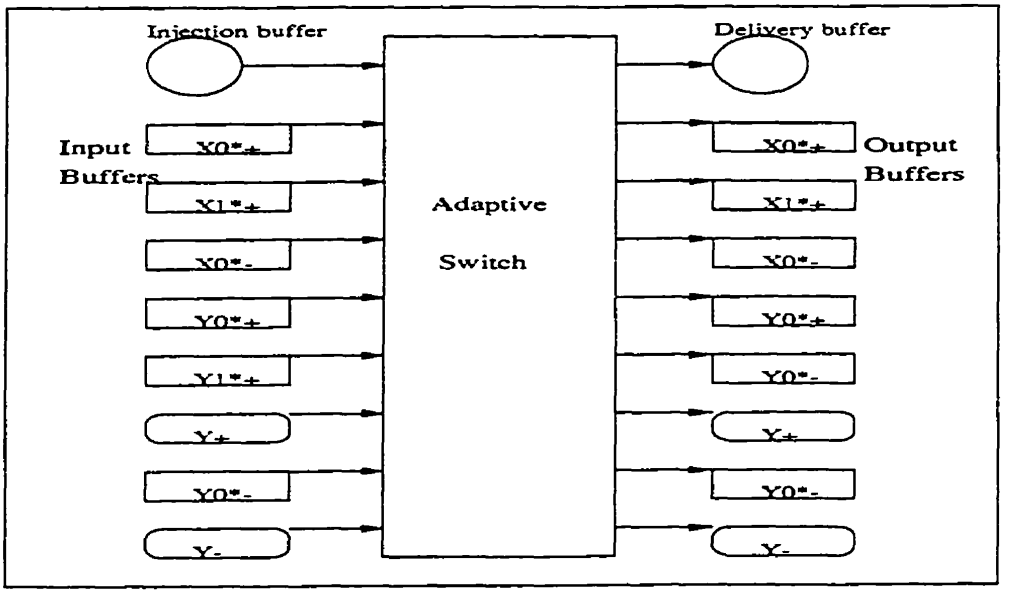


Figure 2.10: Node Model for *-Channels [11]

an example of this class.

2.5.1 *-channels

*-channels is a fully adaptive, deadlock and livelock free algorithm. It requires a moderate amount of resources in the routing nodes [11]. This algorithm is applicable to n-dimensional tori. It can work for messages of unknown size.

*-channels requires 5 virtual channels per bidirectional link of an n-dimensional tori, a number that does not depend on the size or dimension of the network. It allows each message to choose *adaptively*, step by step, among *all* the minimal paths that take it to its destination. No minimal path is discarded in order to obtain freedom from deadlock.

A simplified routing node model for a 2-dimensional tori is depicted in Figure 2.10. The central idea in the *-channels is simple. As shown in the figure there are

basically two types of virtual channels, as will be explained below: *star* channels, and *non-star* channels. Messages will move through the star channels when following dimension-order oblivious routing (or e-cube routing). The non-star channels will be used when taking any of the transitions that would not be allowed by the e-cube routing algorithm, thus obtaining full adaptivity while preserving freedom from deadlock. A more detailed description of the algorithm follows.

Consider the direct link:$((x_{n-1},...,x_i,...x_0),(x_{n-1},...,(x_i+1) mod\ k,....x_0))$. This link will have three virtual channels associated with it:

1)$c^*_{i,+,0,(x_{n-1},...,(x_i+1) mod\ k,...,x_0)}$,

2)$c^*_{i,+,1,(x_{n-1},...,(x_i+1) mod\ k,...,x_0)}$,

3)$c_{i,+,(x_{n-1},...,(x_i+1) mod\ k,...,x_0)}$,

Analogously, link: $((x_{n-1},...,x_i,...x_0),(x_{n-1},...,(x_i 11) mod\ k,...,x_0))$ have three virtual channels associated:

4)$c^*_{i,-,0,(x_{n-1},...,(x_i-1) mod\ k,...,x_0)}$,

5)$c^*_{i,-,1,(x_{n-1},...,(x_i-1) mod\ k,...,x_0)}$,

6)$c_{i,-,(x_{n-1},...,(x_i-1) mod\ k,...,x_0)}$,

The channels 1,2,4,5 are called *star* channels.

In both cases, the star channels will be used for dimension-order routing: messages will move though star channels with prefix $i,+,0$ while correcting dimension X_i following orientation X_i^+ before taking a wrap-around link along dimension X_i. After taking a wrap-around link along dimension X_i following orientation X_i^+, messages will move though star channels with prefix i,+,1 when correcting dimension X_i.

Therefore, a message will be allowed to correct *any* of the dimensions that need correction through non-star channels, A message will be allowed to enter a star channel corresponding to dimension X_i only if X_i is the most significant dimension the message needs to correct, and only if the star channel corresponds to the message's

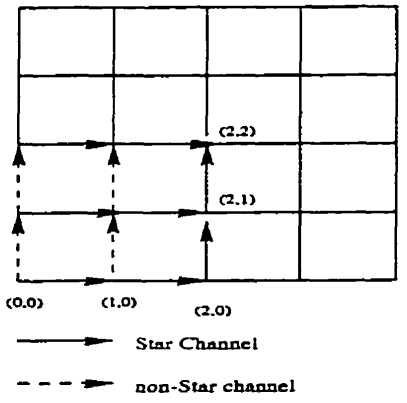


Figure 2.11: Example of the *-Channel routing on a 4-Mesh

having taken a wrap-around along that dimension or not. *-Channel is illustrated in Figure 2.11 for a message moving from (0,0) to (2,2).

2.5.2 Interaction between virtual flow control and adaptive routing

Virtual channel flow control (VCFC) and adaptive routing are two concepts proposed to improve the performance of multicomputer networks [19]. Employing adaptive routing with a minimal channel topology (no extra virtual channels) can result in a poor performance for uniform and some non-uniform traffic patters. Also, using virtual channel flow control alone (using virtual channels to provide extra lanes, but not allow adaptivity in routing) can severely degrade performance for certain traffic patterns. It is also observed that adaptivity, partial or full, might not always result in a better performance [12]. But, when virtual channel flow control is combined with adaptivity in routing, these effects are mitigated and a good performance is obtained. Thus, there is lot of interaction between virtual channel flow control and adaptivity. Adaptive algorithms perform well once VCFC is employed and VCFC gives good benefits with adaptive algorithms. Hence, both adaptive routing and VCFC are indeed beneficial and their real benefit can be felt when they are employed together.

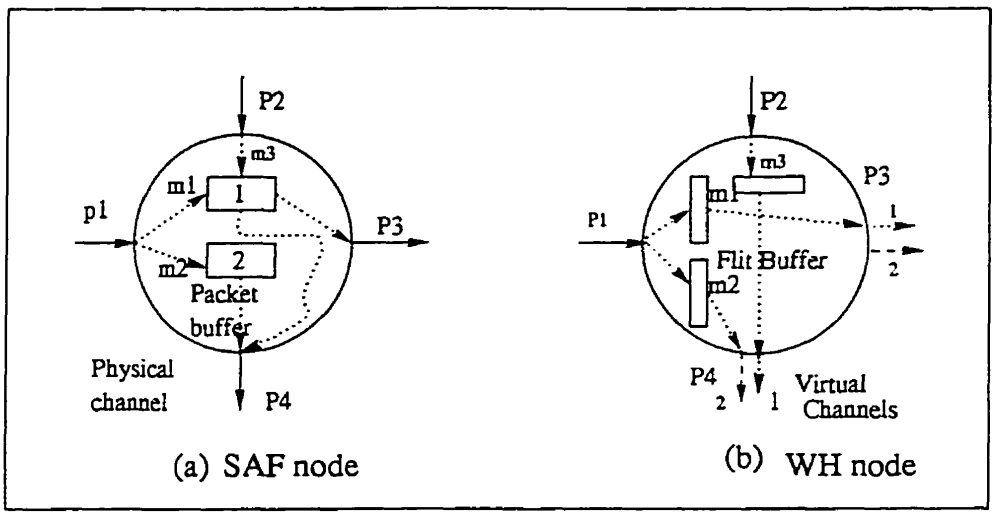


Figure 2.12: Example of a WH router construction from a SAF router [14]

2.6 Construction of New WH Algorithms

The work on designing wormhole (WH) routing algorithms is done largely independent of the results developed for Store-and-forward (SAF) switching computer networks. Recently, it has been shown [14] by Boppana and Chalasani that a class of SAF routing algorithms can also be used, with appropriate modifications, for WH routing. They gave a sufficient condition for deadlock free routing by these WH algorithms and also provided a sufficient condition for sharing flit buffers among multiple channels without creating deadlock. NHop is one of the algorithms developed using this technique.

Figure 2.12 illustrates the construction of WH node from SAF node. In the SAF algorithms based on buffer reservations, each message is given a class, and a message of class i occupies a buffer of class i. A message takes hops from one buffer to another until it occupies a buffer of its destination node, at which point it awaits consumption. Then, the routing relation, S, for a SAF algorithm is from $b \times N$ to b, where b is number of buffers per node and N is number of nodes in the network. Hops allowed are given by the elements of S. The element (b_1, y, b_2) of S represents

a hop allowed from buffer b_1 to b_2 by a message destined to y.

The process of designing a WH algorithm, W, from a SAF algorithm, S, consists of two steps: specification of c, the set of virtual channels, and W, the routing relation from $c \times N$ to c.

1. Let b_1, ..., b_m be the classes of buffers occupied by messages before reaching their destinations in the SAF algorithm. Then, for the WH algorithm, on each physical channel in the network, virtual channels of classes c_1,...,c_m are provided. Figure 2.12 shows this for $m = 2$.

2. Let (b_1, y, b_2) belongs to S, a hop from buffers b_1 to b_2 by a message destined to y in the SAF routing. Then, $(c^1, y\, c_1)$, (c_1, y, c^{11}) belong to W, where class(b_1) = class(c_1) , channel(c_1)=channel(b_1, b_2), c^1 is any virtual channel simulated for any buffer and physical channel combination used by the message to reach b_1, and c^{11} is any virtual channel simulated for any buffer and physical channel combination used by the message after reaching b_2 (see Figure 2.13). If (b_1, y, b_2) is the first hop of the message in the SAF routing, then c^1 is *inj*, the injection channel of the node of b_1. If (b_1, y, b_2) is the last hop of the message in the SAF routing, then c^{11} is *cons*, the consumption channel of the node of b_2.

Informally, if the SAF algorithm specifies that a message should occupy a buffer of class b_i at node x and use a channel from a set of physical channels, l, to complete the next hop, the corresponding WH algorithm specifies that the message at x should take the next hop using a virtual channel of class c_i on any of the physical channels in l. Negative hop algorithms is an example of this category.

2.7 The Negative-Hop Algorithm

In hop schemes, the class of the message at any time is a function of the hops it has taken up to that point. Depending on the function used, various hop schemes can

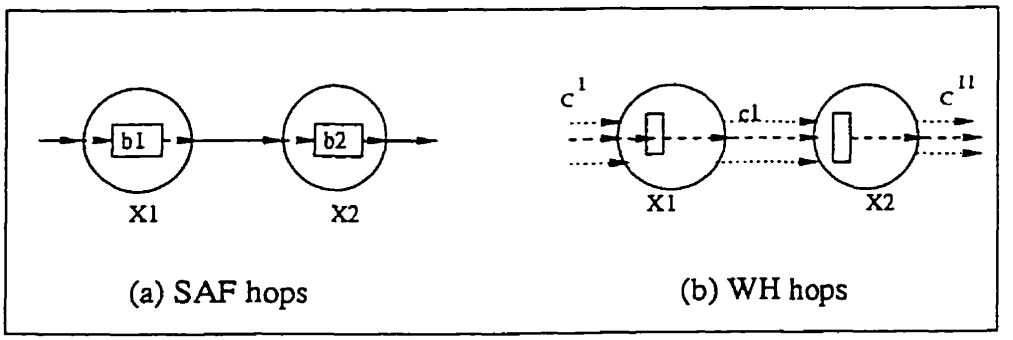


Figure 2.13: Illustration of hops in a WH algorithm constructed from a SAF algorithm

be designed. For many networks, the NHop may require too many virtual channels [12]. The channel requirements can be reduced using improved negative hop schemes (INHop), which are based on the negative hop scheme. The basic technique of INHop algorithm is discussed next.

2.7.1 The SAF version

The network is partitioned such that there are no cycles in any partition, and each partition is given a unique number. Now a negative hop is a hop that takes a message from a node in a higher numbered partition to a node in a lower numbered partition. It has been proved that if H_N is the maximum number of negative hops taken by any message under the improved negative-hop scheme, then $H_N + 2$ buffers are enough for deadlock-free routing [14]. One of these $H_N + 2$ buffers is required to handle direct deadlocks that exist when messages between neighbors in the partition are exchanged.

2.7.2 The WH version

A message can use any hop that takes it closer to its destination. A message that has taken i negative hops uses a c_i virtual channel for its next hop. Direct deadlocks cannot occur with wormhole switching, since messages exchanged between neighbors

use distinct physical channels. Direct deadlocks occur with SAF switching because of the centralized buffer pool. Therefore, the NHop WH algorithm require at most $1 + (\frac{H_I(C-1)}{C})$ virtual channels, where H_I is the maximum number of inter-partition hops a message can take and C is the number of distinct partitions.

2.7.3 Application to tori

Coloring Scheme

In order to implement the NHop it is necessary to give a proper coloring scheme. If the node set of a (k,n)-torus, where n is the dimension of the torus, and k is the number nodes in each dimension, is partitioned into two subsets: P_0, P_1. The subset to which a node $x = (x_{n-1}, ..., x_0)$ belongs is determined using the following rule:

$x \epsilon P_0$ if $(\sum_{i=0}^{n-1} x)\ mod\ 2 = 0$ or $x \epsilon P_1$ otherwise.

For even k the underlying graph of the (k,n)-torus is bipartite, and the partitioning colors the graph. And the maximum number of negative hops in a (k,n)-torus with even k is $n(k/2)$. For odd k, the (k,n)-torus is not a bipartite graph and the partitioning does not color the graph. To solve this problem, assume that for every pair of the nodes a and b connected by a wraparound link, there is an imaginary node c between a and b on the wraparound link; further, assume that this imaginary node belongs to the subset other than that of a and b. Thus a hop on the wraparound link from a to b passes from a to the imaginary node c and then from c to b. One of these hops is a negative hop. The net effect is to increase the maximum number of hops in a dimension by 1, to $\lceil k/2 \rceil$, for odd k.

Algorithm

(Initially, current-class =0 and current-host =source of the message.)

if (current-host $\neq$ destination) **then**

{

1. If color of the current-host is 0 or colors of the previous-host and current-host match, then increment current-class by one.
2. Select any neighbor node that is on a shortest path to destination as the next-host.
3. Reserve a virtual channel of current class.
4. If the virtual channel is available, set previous-host $\leftarrow$ current-host, current-host $\leftarrow$ next-host, and route the message; otherwise, go to step 2.

}

else Consume the message.

When a message is generated, the total number of negative hops taken is set to zero, and the current host is set to the source node. The pseudo-code given above describes how a message is routed as per the negative-hop scheme. A message, when it moves from a node of color 0 to a node of color 1, reserves a virtual channel of the same class it reserved in the previous hop; otherwise, it reserves a virtual channel on a class higher than that it reserved in the previous hop. The class of a message is also incremented if it takes a hop between nodes of the same color. For the partition that is described, this can happen only for hops on the wraparound links in odd radix (k,n)-tori.

The NHop is illustrated in Figure 2.14 for a message from (2,2) to (0,0) in a 4×4 uni-directional link tori. The second and fourth hops are negative hops, but the message class is incremented before making those hops.

2.7.4 NHop with class ranges

The NHop algorithm can be improved by giving more choice of virtual channels for messages in higher classes (INHop). For example, a message with virtual channel

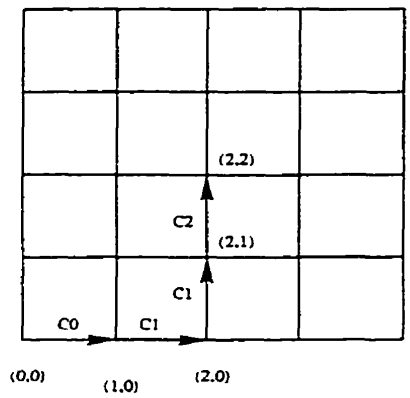


Figure 2.14: Example of the Nhop routing in a 4-mesh

class $i \geq 0$ may use any virtual channel of classes 0,...,i. The actual implementation is as follows. If a message of class 2 does not find a virtual channel of class 2 in the path to its next host, the message selects any free virtual channel in classes 0 and 1 that is in its path, relabels it as 2 and uses it. A virtual channel relabeled by a message of higher class number returns to its original class after the message has relinquished it. A blocked message, however, can only wait for a virtual channel of its class.

Deadlocks cannot occur, since each blocked message waits for virtual channels as per the original algorithm. Starvation may be avoided by ensuring that a virtual channel is relabeled to a higher class only when there are no messages of its class waiting for it. For example if a message is eligible for class i virtual channel and there is no virtual channel of that class available then the message can take any virtual channel of class 0 to it i. The precondition to acquire such lower class (say p) is possible only if there is no other message waiting for the virtual channel of class p. Using *ranges* of classes to select virtual channels gives priority to messages that have already used many virtual channels. In the remaining part of the thesis where ever NHop is referred, it should be interpreted as the improved NHop, INHop.

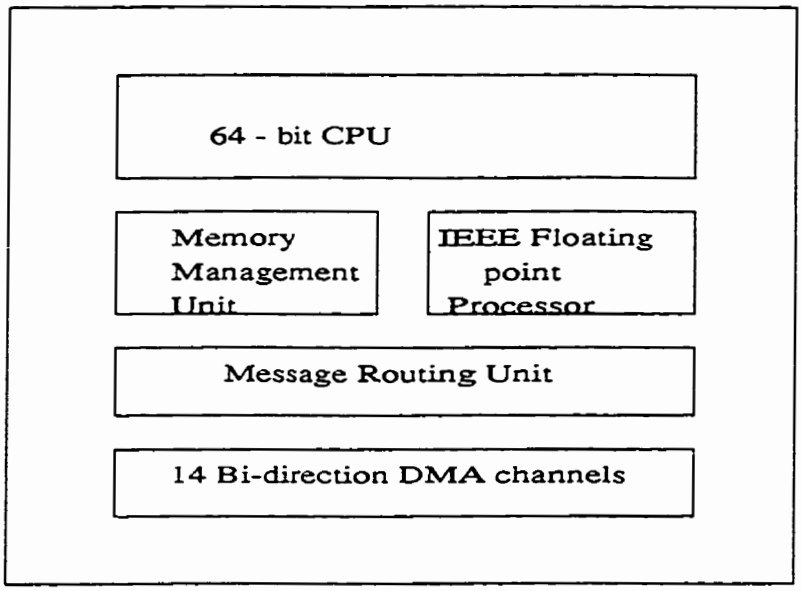


Figure 2.15: The architecture of the nCUBE 2S processor [20]

2.8 nCUBE 2S: An example system

The nCUBE a processor, designed specifically for parallel processing, balances computation with communication. The nCUBE's custom VLSI processor, the nCUBE 2S processor, realizes integrates communication channels with its processing facilities [20].

The processor design is shown in Figure 2.15. Each processor includes 14 bidirectional communication channels: 13 for the interprocessor network - and one for I/O. When multiple nCUBE 2S processors are configured in a hypercube network, the processor architecture provides unmatched communication bandwidth. And because each processor includes its own communication facilities, adding processors to a system increases computational speed, communication bandwidth, and I/O bandwidth.

2.8.1 Hardware implementation

The nCUBE 2S processor's Network Communication Unit (NCU) has been designed to provide full communication support for hypercube parallel processing systems. Communication between processors is accomplished via messages. The messages are switched between processors using wormhole routing. The transmission of a message consists of three stages: Path Creation, Data Transmission, and Path Removal. Each stage is performed in turn by each processor in the message path.

Messages travel though message buffers in each of the three states of transmission. First, the message is copied to the message buffer, which is defined as part of the memory when the program is started. Second, the message is transferred though the network with a speed of 2.75 MB/s. While the message is transferred the sending processor continues its work: sending of the message is asynchronous. Third, the message arrives in the destination's message buffer; from here it can be copied to the processor's memory with the *nread* command. The *nread* is blocking. After calling *nread*, the destination processor waits until the message arrives. By allowing processors to send or receive a message and then immediately return to computation, the nCUBE processor design allows communication and computation to occur simultaneously. In effect, an algorithm can hide its communication. Interprocessor communication is illustration in Figure 2.16.

A message can be visualized as a chain of packets (or flits) being relayed from processor to processor. The first packet (or header) contains an address that the Routing Layer uses to determine which port or ports to use in creating the next link or links in the path. The Routing Layer also uses the address to determine whether to notify the CPU of a message arrival. Complex tree-structured and multi-node paths can be created using broadcasting and forwarding techniques.

The channels the address packet passes through as it moves from processor to processor are automatically reserved for the data packets that follow the address

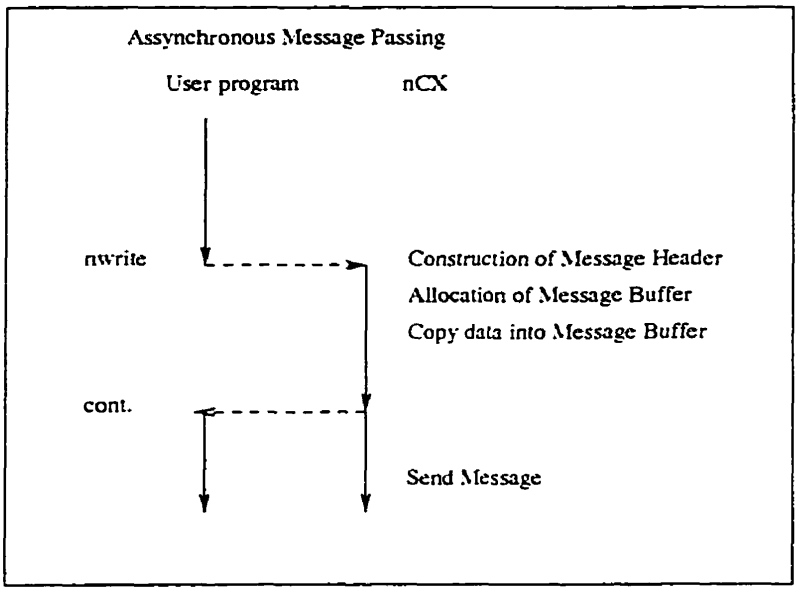


Figure 2.16: The nCUBE communication is characterized by an asynchronous write and blocking read [20]

packet. The NCU is able to buffer up to two packets (buffer depth) on each incoming channel. The NCU will request another packet as soon as it has space for it, until an End of Transmission (EOT) packet (or tail) passes through the channel. The EOT packet causes the processor to release the channel for use by other messages. A short message of a few packets could be stored entirely within channel buffers of intermediate processors in the path, prior to the address packet reaching the destination processor.

Any transmission errors detected by the intermediate processors are encoded in the packets as they pass through. The transmission errors are then detected by the destination processor.

The architecture of the NCU has three layers: the Interconnect Layer, the Routing Layer, and the Message layer. The Interconnect Layer provides the hardware required for establishing physical communication links. The Routing Layer provides

the arbitration and switching logic for creating, maintaining, and removing communication paths between processors in the network. The Message Layer provides the services for reliable and efficient point-to-point data transfer between processors.

2.8.2 Software implementation

The application algorithms that run on each nCUBE processing node are identical to those that run on sequential machines. To achieve efficiency, however, some parts of the programs that implement these algorithms need adaptation to the parallel environment. Typically, it takes care of two steps:
-partitioning the data and/or code among the nCUBE processors;
-communication between nCUBE processors and with the host.

Before processing begins, code and data need to be down-loaded to the local memory in the processing nodes. During processing, partial results obtained at the boundaries of the partitioned data or code in each node may need to be communicated to neighboring nodes.

In summary, the nCUBE massively parallel computers can include up to 8192 nCUBE 2S processors. The nCUBE's communication software takes advantage of the integrated design, by supporting asynchrouns reads and writes and allowing communication and computation to overlap. nCUBE software also includes powerful tools to profile communication and detect any load imbalances. The ability of this hardware and software system to achieve high performance has been demonstrated at various nCUBE installations [21].

2.8.3 Applications

nCUBE's parallelization tools save programmers the trouble of studying network topologies, calculating data distribution schemes, and working with long procedures

of low-level routines. A program can simply call nCUBE parallelization or decomposition routine, and routine automates grid decomposition and data distribution and ensures that the program will be executable on hypercubes of varying sizes. The port of important applications are Discover, Fire, LS-DYNA3D and Oracle database - already in use at major banks [22] - to the nCUBE platform reflects the success of nCUBE's communication model, and demonstrates the adaptability of nCUBE hardware and software to a wide range of applications, from computational chemistry codes to commercial databases.

Chapter 3

Buffer Management

In multicomputer networks, each node in the network communicates with other node in the network using messages. Due to conflicts that arise when several messages simultaneously require the use of the same link, buffering is required in each node. The strategy for managing usage of these buffers can have a significant effect on performance [1].

A scheme is necessary to allocate a node's buffers among the virtual circuits using the node. One simple solution gives each channel on each link a separate buffer. This is inefficient, however, because much of the buffer space will be unused most of the time. By allowing several channels to share buffers, fluctuations in the need for buffer space can be averaged over a large number of communication paths.

There are three common solutions to this situation [1]: 1) direct (dedicated), 2) set-associative and 3) fully associative (centralized). These three schemes offer an increased degree of buffer sharing and increased channel utilization but at the cost of increased complexity in the control circuitry. They are distinguished by restrictions on the placement of each channel's messages. In direct (or dedicated) scheme there is minimal sharing of buffers and each channel has a set of dedicated buffers, i.e., its own FIFO queue. The set-associative scheme has moderate sharing of buffers, and allows each channel to use a larger set of buffers, but the channel is no longer given

the sole access to them. This scheme could be implemented by letting all channels of a single port share a pool of buffers dedicated to this port. In fully-associative (or centralized) scheme there is maximal sharing of buffers and each node has a centralized pool of buffers that all channels share.

The full associative or centralized scheme offers the most sharing at the cost of additional control circuitry. However, this scheme could also suffer from **buffer hogging**. Buffer hogging occurs when one output port becomes congested and uses a disproportionately large portion of the buffer pool, impeding traffic on other ports. Under these circumstances, better performance is obtained if the degree of buffer sharing is limited, and each port is limited to some maximum number buffers.

3.1 Dynamic Assignment of Buffers

Buffer assignment problem does not exists in dedicated buffers scheme, it is only in the case of centralized organizations that it comes into picture. Unlike in the dedicated buffers organization, in the centralized organization buffers are dynamically assigned to virtual channels on demand. So, a mechanism is required to keep track of the buffer assignment. In this section a solution for this management is discussed. This strategy assumes that number of messages waiting to use a given output channel can be larger than one.

The buffers waiting to be forwarded on an output channel are "chained" into a linked list for that channel. Buffer chaining may also be used to implement variable buffers in datagram networks. When a message (or flit) arrives, it is placed at the end of the linked list corresponding to the output channel on which the message has to be forwarded. It is removed from the list after it has been successfully transmitted to the next node. The linked lists are managed as FIFO queue to ensure that messages are forwarded in the same order in which they arrived.

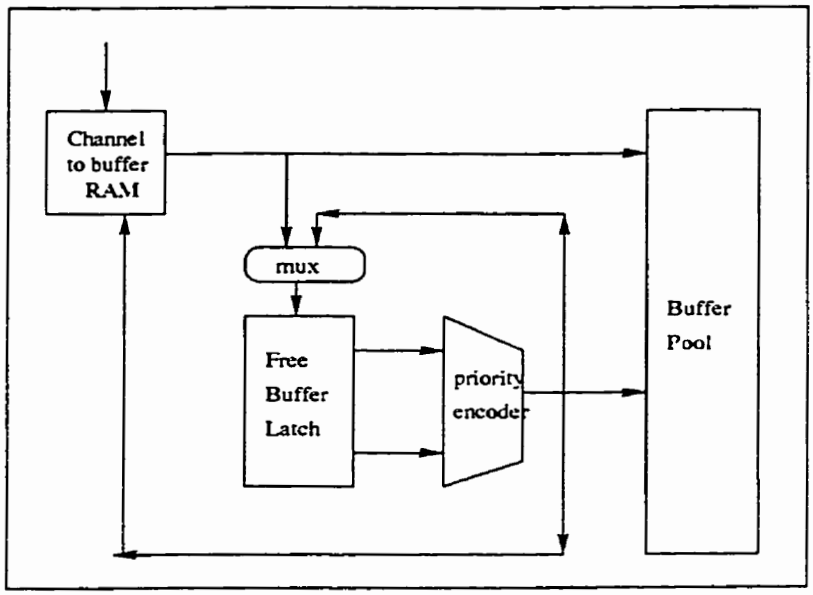


Figure 3.1: Buffer management circuitry [1]

A simple design where an output channel can use at most *one* buffer at a time is shown in Figure 3.1. The $c \times p$-word RAM maps output channels to buffer addresses. Word i holds the address of the buffer containing a message for channel i. The list of free buffers is maintained by a b-bit latch, called the "free buffer latch". The free buffer latch is implemented as a bit-addressable latch, i.e., a memory device that is written as a RAM (one bit at a time), but read as latch (all bits in parallel). Each bit indicates the status of a buffer: free (1) or in use (0).

When a new message arrives, the buffer management circuitry must perform two operations, assuming the flow control circuitry has first established that the message can be accepted:

1. find and allocate a free buffer, and
2. record the location of the buffer so that the message can be found when it is time to forward it.

The address of a free buffer is determined by a priority encoder attached to the free buffer latch. The resulting address is sent to the memory module, buffer pool. This address is also used to clear the corresponding bit in the buffer latch, effectively allocating the buffer and completing the first operation. The second operation is accomplished by writing the address of the selected buffer into the channel-to-buffer RAM at the memory location corresponding to the output channel responsible for forwarding the message. The latter is obtained from translation table.

Sending a message on output channel i also requires two operation:
1. locate the buffer holding the message for channel i, and
2. release the buffer.

The first task is accomplished by reading address i of channel-to-buffer RAM. The resulting address is used to set the corresponding bit in the free buffer latch, marking the buffer free to be used by other message, thus accomplishing the second task.

3.2 Amount of Buffer Space

Buffers increase the total bandwidth provided by the network through pipelining and by "softening" the impact of statistical fluctuations in the traffic distribution. However, a successively smaller improvement in performance is obtained with each additional buffer that is added. Intuitively, the node need not provide buffer space beyond that which is necessary to keep output links busy when there is traffic requiring use of the link.

In extreme cases, buffer deadlock will result. Buffer deadlock occurs when message traffic halts because two or more nodes have exhausted all available buffer space. A cycle is formed where each node in the cycle cannot forward a message because no buffers are available to receive the message, and no buffers can be freed because

message cannot be forwarded. Thus, sufficient buffer space must be provided to: reduce the probability of buffer deadlock. and ensure good performance.

3.2.1 Deadlock considerations

Buffer deadlock can be prevented if enough buffer space is provided in each node. A brute force solution is to provide each virtual channel with its own buffer (dedicated buffers). Because each circuit is allocated a buffer in each node it passes through, traffic on a circuit cannot be blocked by traffic on other circuits, and buffer deadlock cannot occur. Providing a separate buffer on each channel is wasteful, however, because each component must provide as many buffers as there are channels.

Similar performance can be achieved if many channels share a much smaller pool of buffers. Several approaches have been proposed that avoid deadlocks in store-and-forward networks. For example, assuming each node has at least LV_{max} buffers, where LV_{max} is the maximum number of hops traversed by any virtual circuit. The node's buffer pool is partitioned into LV_{max} disjoint pools or levels. A register is associated with each circuit passing through the node indicating the number with i hops remaining before the final destination is reached. A message arriving on a circuit with i hops remaining can only be placed in a level i buffer.

Deadlock is avoided because all messages in the network can be delivered to their respective destinations, and all buffers holding messages can therefore be released. All level 1 buffers can be released because the messages they hold can immediately be forwarded to their final destinations. Because level 1 buffers are released, level 2 buffers can also be released by first emptying the level 1 buffers and forwarding the level 2 messages to these level 1 buffers, when the message may leave the network. Applying this argument recursively, it is easy to see that all messages in the network can be forwarded to their respective destinations.

The central disadvantage of this scheme is that large networks require more buffers than smaller ones, so the switch must provide enough buffers to accommodate the largest possible network. This may require an excessively large amount of buffer space. In addition, if traffic is highly localized, many buffers are wasted because those reserved for higher hop counts are never used.

3.2.2 Performance considerations

Each switching node must provide sufficient buffer space to maintain a steady flow of traffic. Otherwise, communication bandwidth will be wasted. Studies of multi stage switching network indicate that little performance improvement arises beyond three buffers per node [23]. However , as the studies of the previous section indicate, three buffers are not sufficient to avoid many deadlock scenarios in the single stage networks discussed here. Now, the main questions are: how many buffers should each component provide to achieve good performance, and how many buffers should each virtual circuit be allowed to use at one time.

The appropriate amount of buffer space depends on characteristics of the traffic distribution. Buffering and flow control are of little consequence when the network is lightly loaded. In most heavily loaded network, a few heavily loaded links become saturated. These links will limit the overall performance of the system. Under these circumstances, traffic will back up on circuits "upstream," i.e., leading up to the bottleneck area, which circuits "downstream" will be starved waiting for messages to get past the bottleneck.

Studies have indicated that [1] network with a certain buffers per node (for example 16) yield same performance as networks with an infinite amount of buffer space. Restricting virtual circuits to using at most one buffer at a time results in no significant degradation in performance.

3.3 Buffer Requirements of WH algorithms for Tori

3.3.1 e-cube and *-channel

The e-Cube routing algorithm requires two classes of virtual channels one for upper class and another for lower class. With dedicated buffer organization this algorithm requires 4n buffers for nD torus (8 for (16,2) torus i.e., 16 nodes in each of the 2 dimensions). The *-channel algorithm requires three classes of virtual channels and is based on the e-cube algorithm: two virtual channels are used to avoid deadlocks and an extra class is used to provide adaptive routing. With dedicated buffers, this algorithm requires a minimum of 6n buffers for nD torus (12 for (16,2)-torus). It is possible to reduce the requirement, by providing dedicated flit buffers for the e-cube channels and centralized flit buffers for adaptive channels. With as few as 4n+1 buffers (9 for (16,2)-torus), fully-adaptive routing can be provided by this algorithm.

3.3.2 NHop

A (k,n)-torus, n dimensional torus with k nodes in each dimension, has a maximum of $n\lceil k/2 \rceil$ hops. Since the graph of (k,n)-mesh is bipartite, for both odd and even k, the total hops is n(k-1). Using C=2 and depending on the type of the network, we obtain that the number of virtual channels needed is at most $1 + \lfloor n\lceil k/2 \rceil /2 \rfloor$, for a (k,n)-torus (7 for (8,3)-torus), and $1 + \lfloor n(k-1)/2 \rfloor$ for a (k,n)-mesh (12 for (8,3)-mesh).

For k-ary n-cubes, NHhop scheme requires more virtual channels than the e-cube and *-channel. But hop schemes provide a deadlock free routing even when flit buffers are shared among multiple channels. This ability of hop schemes make them competitive for many practical network sizes. It has been shown that NHop performs better than e-cube and *-channel schemes for tori.

3.4 Relative Performance of WH algorithms for Tori

With centralized buffer organization, NHop provides, for many configurations of k-ary n-cube networks, fully-adaptive routing while requiring fewer buffers than the e-cube. For example, for (8,3)-torus used in a 512 node Cray T3D, the Nhop requires seven flit buffers for fully adaptive routing, while the e-cube requires 12 flit buffers. For $8 \times 16 \times 8$ torus (the maximum configuration for Cray T3D), the NHop requires nine buffers, while e-cube requires 12 buffers. Because of longer diameters and simpler routing with e-cube meshes, the NHop requires more buffers than the e-cube in meshes, the NHop requires more buffers than the e-cube, unless $k \leq 5$.

The performance evaluation given in paper [14] for three algorithms, NHop, e-cube, and *-channel indicates that the NHop performs better than the e-cube and *-channel schemes for tori. Based on the buffer cost and throughput evaluations, the NHop has advantage over previously proposed wormhole routing algorithms for torus networks. This is the main reason why the NHop was selected for doing further research in the buffer management in this thesis.

3.5 Buffer Organizations

The main focus of this thesis is to study the performance of the network using different buffer organization. Three buffer organizations are chosen for conducting this study. These three buffer organizations are described in detail in this section.

3.5.1 Dedicated buffer organization

In this organization (shown in Figure 3.2) if m is the number of flit buffers used, p the number of incoming physical channels to a router, and v the number of virtual channels per physical channel then $m = pv$. In other words, each virtual channel

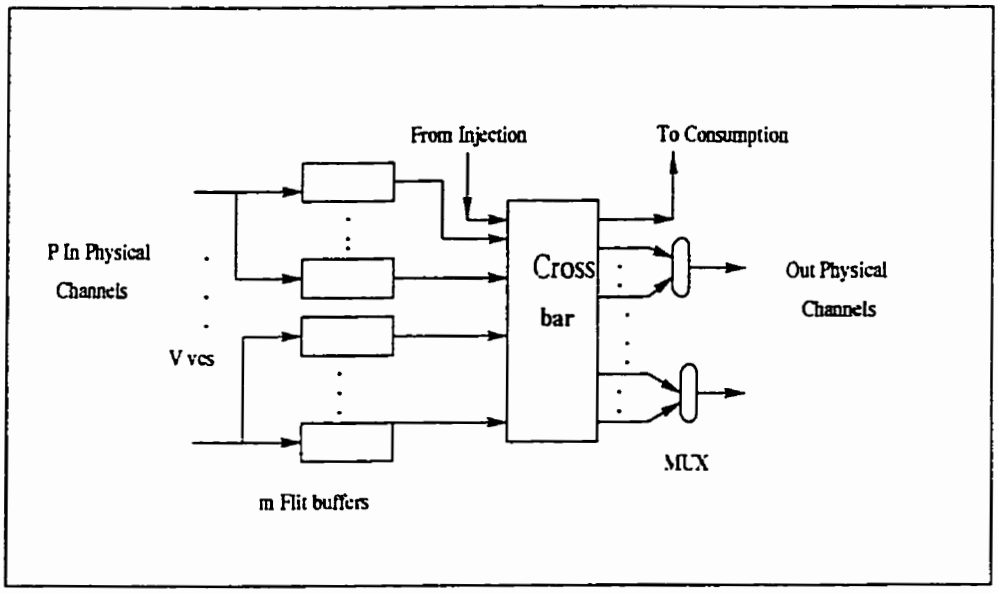


Figure 3.2: Dedicated buffers organization

leading into a router has its own flit buffer. No delay is suffered by the message in order to acquire the buffer.

Router Delay

For dedicated buffers organization, the major components of delay are flow control from incoming physical channels to flit buffers, crossbar delay from flit buffers to the outputs of central crossbar, and the virtual channel controller delay from the outputs of the crossbar to outgoing physical channels. For header flit, header decode and update and channel selection are the additional costs.

3.5.2 Centralized buffer organization

In this organization (shown in Fig.3.3)all the buffers are shared by all the virtual channels. Each buffer is assigned a class. The virtual channel goes through a crossbar before accessing its exclusive buffer. Once a flit buffer is allocated to a virtual channel, it remains associated with that virtual channel until it is released.

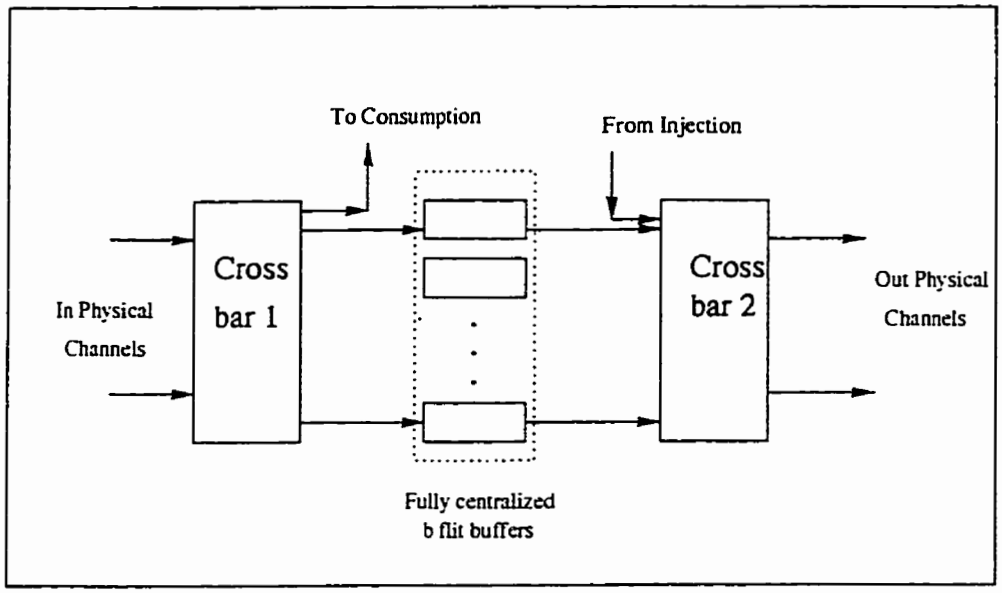


Figure 3.3: Centralized buffers organization

The assignment of buffers to a particular class can be done in two different ways:

1. Uniform: all the classes are assigned equal number of buffers i.e., all the buffers of a node are divided equally among the classes used.

2. Non-Uniform: Buffers are assigned to a particular class based on some criteria, such as locality of the messages.

Router Delay

The header decode and update and channel selection are similar for both dedicated and centralized organizations. The flow control in the centralized organization is done in Crossbar 1. So, when a header flit arrives, say from Node A to Node B, it is allocated a central buffer by establishing a connection through Crossbar 1 of Node B or is refused connection. The header is retained by Node A for a few cycles, by which time rejection of the header, if occurred, will be known.

Once the connection is established, the allocated central flit buffer acts as a dedicated flit buffer to that virtual channel, and the transit of data flits is similar to that of dedicated flit buffer implementation. Therefore, a multiplexer between the inputs and buffers in the logical organization is set once at the time of setting up the path. An input channel may be allocated multiple flit buffers, one for each active virtual channel on the input channel. Since a cross bar naturally provides the multi-cast communication, this can be accomplished easily by request accepted and removing one such connection for each request completed.

The amount of switching done by Crossbar 2 is same as the amount of switching done by the multiplexers at the output physical channels shown dedicated buffers organization Figure 3.2. This crossbar changes its settings on flit-by-flit basis, much the same way the multiplexer in Figure 3.2 change their settings.

Dedicated verses centralized. A connection from an input virtual channel to an output virtual channel takes more time, and data flits go through two smaller crossbars instead of one large crossbar with centralized organization. But the centralized organization with buffers between the crossbars lends itself easily to pipelining, thereby avoiding increase in clock cycle time. Since centralized organization has longer data path, the router delay for a message increases compared to dedicated organization, when the number of buffers is kept the same.

3.5.3 Hybrid buffer organization

The dedicated and centralized buffer organizations have their advantages and disadvantages. The hybrid buffer organization proposed here inherits the merits of the two organizations and it is described next.

In this organization (shown in Figure 3.4) each channel is assigned a particular number of dedicated buffers. So, in order to use these buffers the virtual channels need not go through the crossbar. The remaining buffers are organized centrally . If p is the number of physical channels per node, l is the number of dedicated buffers

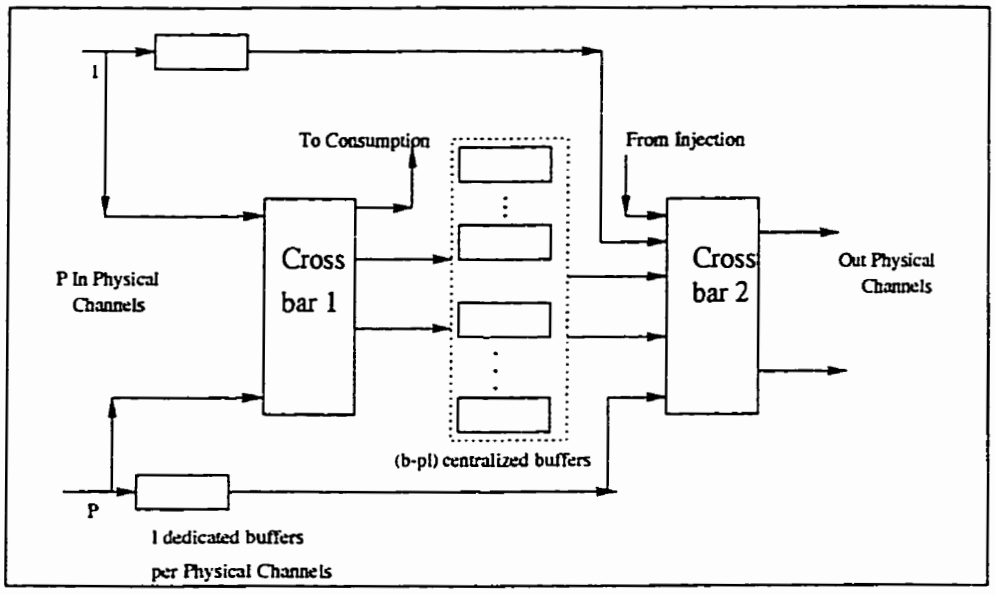


Figure 3.4: Hybrid buffers organization

assigned to each physical channel then the remaining $(b - pl)$ buffers are maintained as a central pool of buffers.

Router Delay

The header decode and update and channel section are same as other two organizations. The flow control is done partly as in the dedicated buffer organization and partly as in the centralized buffer organization, depending on the nature of the virtual channel (dedicated or centralized). Messages which could get a dedicated buffer virtual channel suffer no extra delay at the router. If message is taking centralized buffer virtual channel, it suffers more delay as explained in the centralized buffer organization.

3.6 Summary

In summary, dedicated, hybrid, and centralized schemes offer an increased degree of buffer sharing and increased channel utilization but at the cost of increased complexity in the control circuitry. The results presented in chapter 5 indicate that dedicated buffer organization has a better performance over centralized in uniform traffic pattern. On the other hand, centralized out performs dedicated in hot-spot traffic. In reality the traffic in a network dynamically changes from uniform to occasional hot-spots. The intuition is that the hybrid buffer organization can handle this kind of traffic more efficiently by shifting itself to centralized in hot-spot traffic and towards dedicated in uniform traffic. The following chapters give the experimental results of the study conducted to analyze the performance of hybrid buffer organization.

Chapter 4

System Model

To compare the performance of three different buffer organizations: Centralized, Dedicated and Hybrid for NHop routing algorithm an experimental system is simulated with following features:

4.1 Simulator

A continuous time and discrete event simulator has been developed to perform the desired study on the interconnection network. In this system the time parameter is continuous and events occur at discrete points in time t_1, t_2, ... where $(t_{i-1} - t_i)$ is not a constant. The simulator has the flexibility to support various topologies, routing algorithms and traffic patterns.

Some of the *system variables* that describe the state of the system: network size, network topology, number of virtual channels per physical channel, number classes of virtual channels, total number of buffers per router, type of the buffer organization, average path length, average latency, and average utilization.

Some of the *events* that change the state of the system are: initializing the system, injection and consumption of messages, scheduling messages for the transmission on flit by flit basis, transmitting the flits, avoiding starvation of the worms.

incrementing the clock and system variables, gathering statistics, and at the end of the simulation period analyzing the collected data and producing the report.

A singly linked list of events is maintained. The events are kept in this list in the increasing order of their time. The clock cycle time is same for all the routers. The clock is incremented from one event time to the next event time. If the header flit is accessing a centralized buffer for the transmission then it takes one clock cycle to setup the connection and one clock cycle for the transmission, all other flits (body) take only one clock cycle for transmission.

4.2 Network Topology

The simulations are run on a (16,2)-torus. Torus is a special case k-ary n-cube family with $n = 2$. This is a symmetric network in that there exists a homomorphism that maps any node of the graph representing the network graph into any other node. It has k^n nodes with a diameter of $2^n - 1$. Torus is commonly used in multicomputer systems (ex: Cray T3D, T3E). Because it is symmetric network as hypercube, and it supports wider channels as mesh. And, most multicomputer network topologies used in wormhole-routed systems are low-dimensional meshes and hypercubes.

Each channel is represented by a pair of (uni-directional) physical channels. The number of virtual channels per physical channel for Nhop centralized and hybrid versions is 18. In the case of Nhop dedicated version 9 virtual channels are mapped onto a physical link. The virtual channels of Nhop algorithm are divided uniformly among 9 different classes. Multiple virtual channels mapped to a physical channel share its bandwidth in time-multiplexed manner.

4.3 Workloads Model

In this section the message generation and traffic patters used in the system are described.

4.3.1 Message generation

A processor generates messages with a mean message generation rate of λ. For most experiments, the message inter-arrival times are exponentially distributed. However, we have also ued a hyper-exponential distributed inter-message generation time in order to study its impact on the performance. The messages have fixed length of 20 flits. For example, 20 flit messages could be used for transmitting four 64-bit words together with header, checksum and other information on 16-bits wide physical channels such as the ones used in Cray T3D. However, performance results for workload that consists of both long and short messages are also given in the next chapter.

Hyper-exponential message generation and message length. For a given mean message generation rate or mean message length i.e., λ or L, the nodes can generate messages in hyper-exponential distribution. In hyper-exponential distribution nodes can chose to generate messages in any one of the two different stages (S_1 or S_2). Where S_1 is the mean λ or L of stage 1, α is the selection probability of this stage, and S_2 is the mean λ or L of state 2. The values of S_1 and S_2 can be computed as follows:

$$S_1 = S - S(1-\alpha)\ (\sqrt{\tfrac{C^2-1}{2\alpha(1-\alpha)}})$$

$$S_2 = S + S\alpha\ (\sqrt{\tfrac{C^2-1}{2\alpha(1-\alpha)}})$$

where S is the mean λor L and C is the desired coefficient of variation. The value of α should satisfy the following relationship: $\frac{1-\alpha}{1-\alpha} > C^2$

In the simulations experiments conducted for this study $\alpha = 0.95$ and C=4.

4.3.2 Traffic patterns

Uniform

Uniform traffic is widely used in simulation studies and serves as a benchmark traffic pattern. In this traffic pattern a node sends messages to any other node in the network with an equal probability. Uniform traffic could be representative of the traffic generated in massively parallel computations in which array data are distributed among the nodes using hashing techniques.

Hot-spot

More realistically, the traffic pattern tends to be random coupled with some hot-spot type traffic. In hot-spot traffic a single node (hot node) receives a specified fraction of the total messages generated in the network.

More specifically, given that N is the total number of nodes in the network, m is the number of messages generated per node per a clock cycle ($0 \leq m \leq 1$), h is the fraction of messages directed at the hot-spot, each node generates messages directed to hot-spot at the total rate of mh. The effective number of messages to hot-spot are $m(1-h)+mhn$.

4.4 Traffic Sampling

For better randomness separate sequence of random numbers per node have been maintained for the distribution of message interval time, selection of destination, etc.

For each simulation, sufficient warm-up time is provided to allow the network to reach steady state. After the warm-up time, the network traffic is sampled at periodic intervals. The counters used for statistics gathering are reset at the beginning of each sampling period. Statistics are gathered during sampling time and analyzed for convergence. Thirty one samples of 2000 worms have been taken during

a simulation. First sample has been discarded to allow the network to reach steady state. It was observed that the network reached steady state after the first sample.

4.5 Some Parameters of Interest

4.5.1 Buffer size

In wormhole routing, bubbles could be introduced, especially at low traffic, in transmission of consecutive flits of message because of asynchronous pipelining. To reduce these bubbles, the depth of buffers is fixed to 4, i.e., each buffer can hold four flits of the same message. Whenever, a buffer has space for one or more flits, next data flit is sent from the previous router in the path.

4.5.2 Average number of hops

For uniform traffic, the average number of hops is the average path length of the networks. For a k-ary n-cube, it is approximately $\frac{nk}{4}$; for (16,2) - tori the average path length is 8. As a part of validatiion of the simulation, we have collected information on the average path length for all the simulations for uniform traffic and it was found to be around 8 hops.

4.5.3 Performance metrics

Average response time, l, and average channel utilization, ρ have been considered to study the performance of the network generating messages at a mean message generation rate of λ.

Average response time

The average response time of a message is $\omega + (m_l + \bar{d} - 1) \times f_t$, [12] where ω, m_l, $\bar{d}$, f_t are the average wait time, average length of the message in flits, average number of hops taken by a message, and the time to transfer a flit between neighbors,

respectively. In some of the experiments conducted in this study, $m_l = 20$, $\bar{d} = 8$ for uniform traffic and $f_t = 1$ for the transmission of body flits and it could be two for the header flit depending on the type of buffer it is accessing.

Average utilization

The average channel utilization refers to the fraction of the physical channel bandwidth utilized in any time interval when the network is in steady state. It is also called the network utilization factor or normalized throughput of the network. The average channel utilization, denoted ρ. is computed as the ratio of network bandwidth utilized to the raw bandwidth available.

$\rho = \lambda m_l \bar{d} \times \frac{Number of nodes}{Number of channels}$,

where $\frac{1}{\lambda}$ is the average message interval time. For k-ary n-cube, this can be simplified to $\rho = \frac{\lambda m_l \bar{d}}{2n}$ [12] ,the numerator computes the average traffic generated by a node. and the denominator gives the available bandwidth due to the physical channels originating from a node. In our simulation with $m_l = 20$ $\bar{d} = 8$ for uniform traffic, $n = 2$ it becomes $\rho = 40\lambda$.

Chapter 5

Simulation Results

This chapter presents the results of the simulation experiments carried out to evaluate the performance of the centralized, dedicated and hybrid buffer organizations using NHop routing algorithm. For each experiment two sets of results are presented: one for the uniform traffic and the other for the hot-spot traffic.

The first experiment was to test the correctness of the results produced by the simulator. The goal of the second experiment was to establish a base results set for the research. The third experiment studies the impact of the clusters. In the fourth experiment the message length was varied to study the effect long and short messages on the performance of the three buffer organizations. The fifth experiment was aimed at studying the effect of the percentage of hot-spot messages. The final experiment was run to find the best combination of dedicated and central buffers for the hybrid buffer organization for a given number of total buffers per node (router).

In all the experiments the response of the system for the given mean message generation rate (λ) was used as parameter for performance comparison. NHop algorithm is chosen to perform this study because of its better performance over e-cube and *-channel (described in Chapter 3).The default parameters were as given in the system model and some of them are listed below:

Variable Name	Default Value
Network topology	Torus
No. of nodes in the network	256
Message generation	Exponential
Message length	20
Routing algorithm	NHop
Hot-spot percentage (h%)	10
No. of classes	9
No. of virtual channels per physical channel:	
Dedicated	9
Centralized	18
Hybrid	18
Total number of buffers per node	36
No. dedicated buffers in hybrid org.	16
Buffer depth	4
Body flit transmission time	1
Header flit transmission time:	
using Dedicated buffer	1
using Centralized buffer	2

Table 5.1: Default parameter values

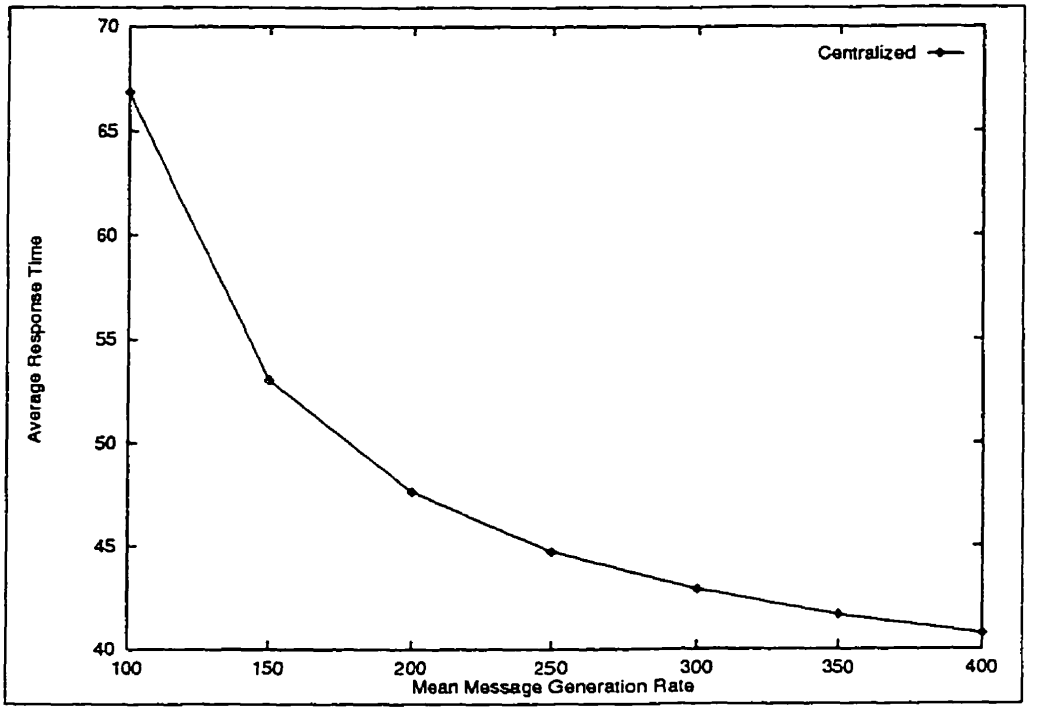


Figure 5.1: Validation: Performance of NHop under uniform traffic on (16,2)-torus

5.1 Validation

The first experiment on the simulator was to validate it. Results are captured and compared with the results given in [14].

The results given in the Figure 5.1 are very close to results presented in [14]. The average message latency as given in Chapter 4, is $\omega + (m_l + \bar{d} - 1) \times f_t$, [12] where ω, m_l, $\bar{d}$, f_t are the average wait time, average length of the message in flits, average number of hops taken by a message, and the time to transfer a flit between neighbors, respectively. In this experiment $m_l = 20$, $\bar{d} = 8$, traffic pattern is uniform. The buffer organization is centralized with $f_t = 1$ for the transmission of body flits and 2 for header flit. The total number of buffers is 16. In a contention free network when λ is very high $\omega = 0$, then it was observed that NHop took 45 clock cycles to transmit a single message. It is easy to see that the result obtained satisfy the given equation for average message latency.

The results given in Figure 5.2 are also very close to those given in [14]. Note

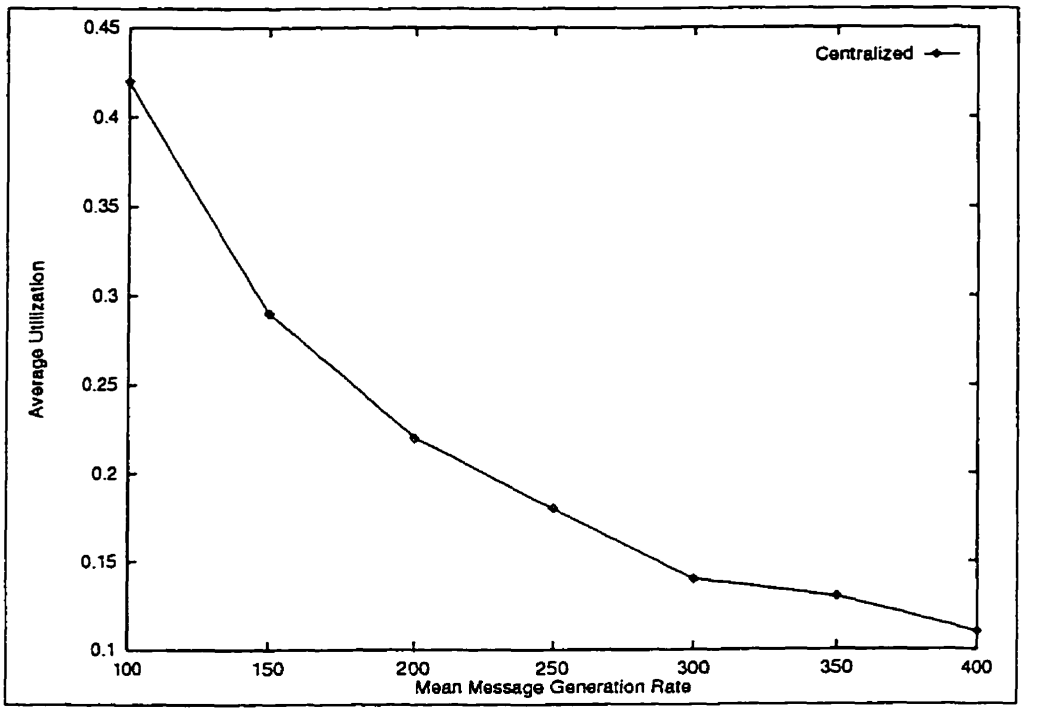


Figure 5.2: Validation: Performance of NHop under uniform traffic on (16,2)-torus.

that the average channel utilization for a k-ary n-cube is given as $\rho = \frac{\lambda m_l \bar{d}}{2n}$, where $m_l = 20$ $\bar{d} = 8$ for uniform traffic, $n = 2$. Now, $\rho = 40\lambda$. We have observed that the results obtained from the simulation experiment satisfy this equation too.

5.2 Base Class Results

This experiment is aimed at establishing a base set of results for this research work. Each node in the network generates messages in the exponential distribution for a given mean message generation rate (λ). The message length was fixed to 20 flits.

5.2.1 Uniform traffic

From Figure 5.3, it is clear that the dedicated buffer organization has a much better performance than the centralized buffer organization and the hybrid organization

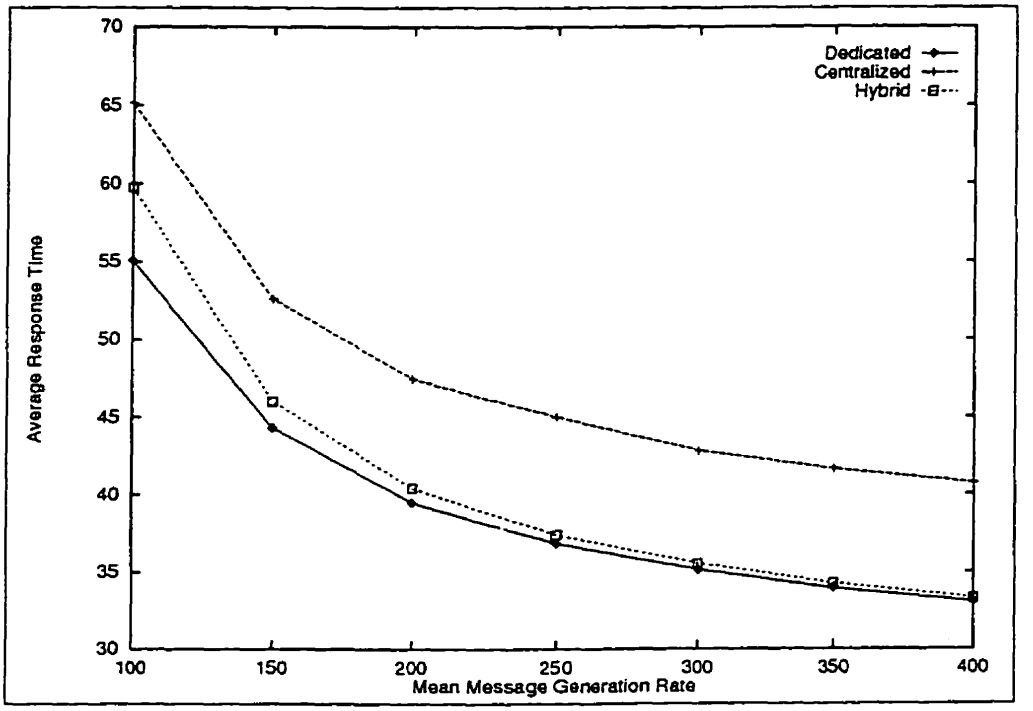


Figure 5.3: Uniform traffic: exponential message generation

closely follows the dedicated organization. The better performance of dedicated organization could be because of not having any delay in acquiring a buffer for transmission.

In uniform traffic pattern all the links are utilized in an uniform way and the necessity to increase the number of active virtual channels in one direction, by shifting the buffers of under utilized links does not arise. If centralized buffer organization is used in this scenario, there will be a overhead in acquiring buffers from the central pool. It is also clear from the graph that message delay in centralized buffer organization is approximately equal to the message delay in dedicated organization plus the average path length. The extra time can be attributed to the extra one clock cycle spent by the header flit to acquire a buffer at each node of its path.

The hybrid buffer organization is almost like dedicated buffer organization for

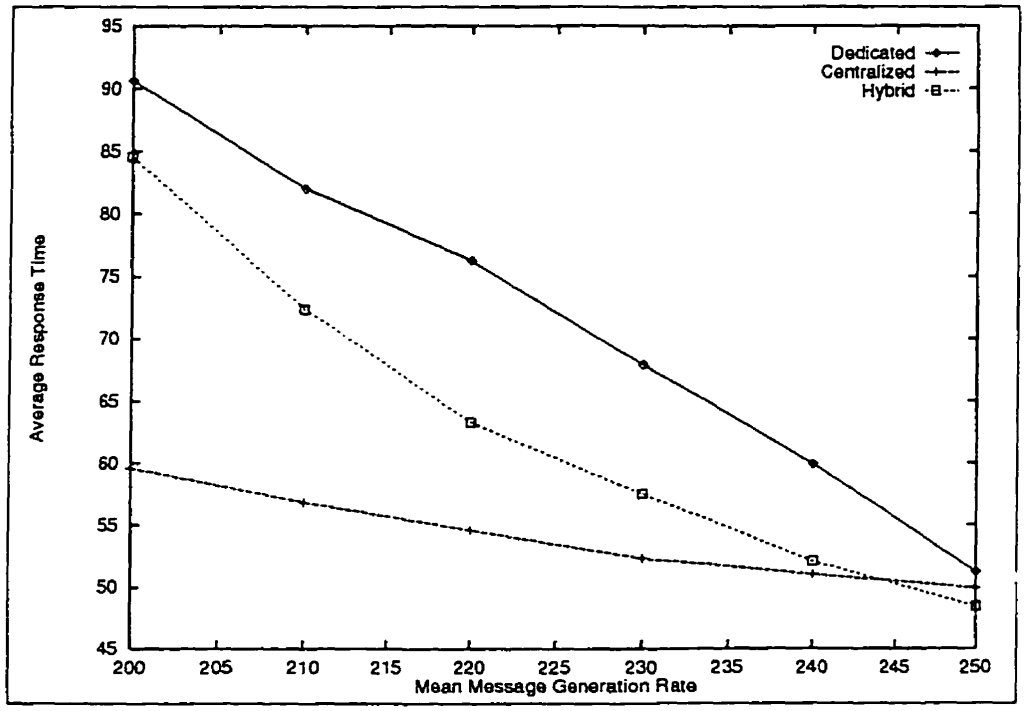


Figure 5.4: Hot-spot traffic: Total delay, Exponential message generation

$\lambda > 200$ and it is at the mid point of centralized and dedicated at lower λ. This behavior is expected because of the combination of buffers used for hybrid was almost half dedicated and half centralized (to be specific 16 dedicated and 20 centralized).

5.2.2 Hot-spot traffic

To have a better picture of the behavior of the response time for the messages generated in the network, the results for hot-spot traffic are divided into three different categories. In the first category the average response time of the system for all messages (both regular and hot-spot) is given. In the second category the results are given only for the regular messages i.e., only non-hot-spot messages. The third category gives the results for the hot-spot messages only.

Total delay

From the Figure 5.4 for total delay shows that the centralized buffer organization is very much suitable for hot-spot traffic and as the λ decreases the performance of the dedicated organization deteriorates rapidly. The hybrid organization is again in the middle of the other two.

This behavior of dedicated buffer organization could be because of the fixed number of virtual channels per link and its in-capability to dynamically increase the capacity of some links which have more demand. In contrast the asset of centralized buffer organization is the capability to increase number of active virtual channels for links with high demand. This is possible, as explained in previous section, because of its capability to shift the buffers of under utilized links to the links in high demand.

At $\lambda > 250$, the hybrid organization is even better than the centralized and the dedicated. This is because it uses 16 dedicated buffers and there is no additional clock cycle delay in allocating these buffers. In contrast, in the centralized organization all buffers are in the central buffer pool.

Regular delay

This set of results consider only the regular messages (messages whose destination is not the hot-spot node). Figure 5.5 shows very interesting results for this category. For higher λ the centralized organization is taking more time than the dedicated buffer organization and the hybrid is slightly better than the dedicated. But, as the λ decreases the behavior of dedicated is not stable and its performance deteriorates rapidly. On the other hand, the response time for the centralized organization is fairly robust.

Initially, when the message generation rate is low the network is almost utilized uniformly. As the λ decreases more messages acquire the resources and hold them

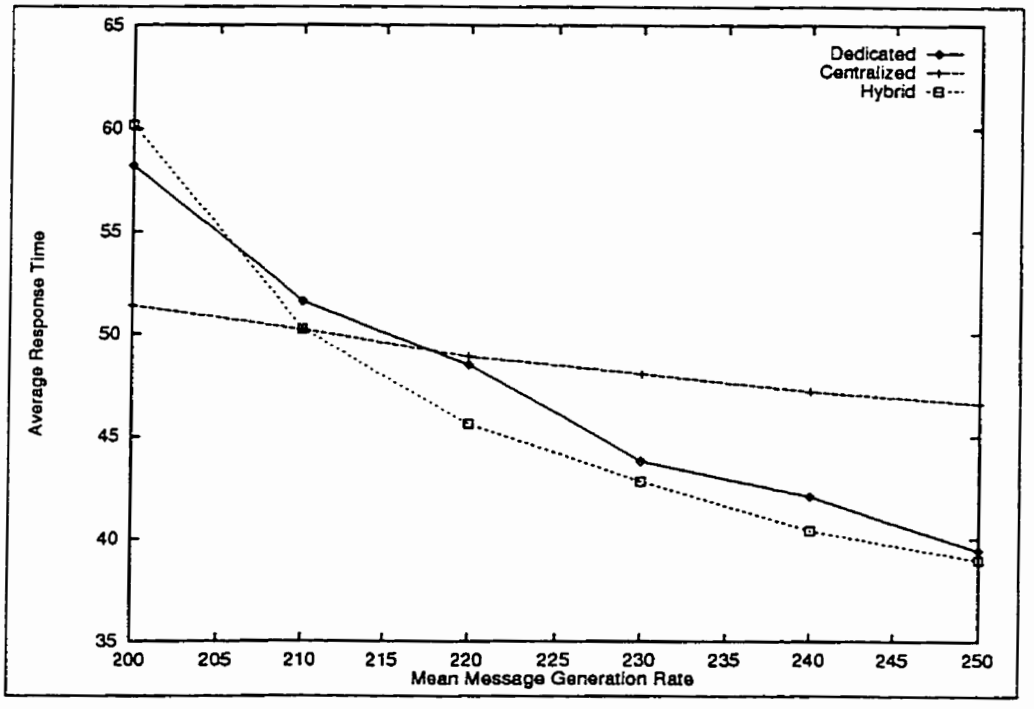


Figure 5.5: Hot-spot traffic: regular messages delay, exponential message generation

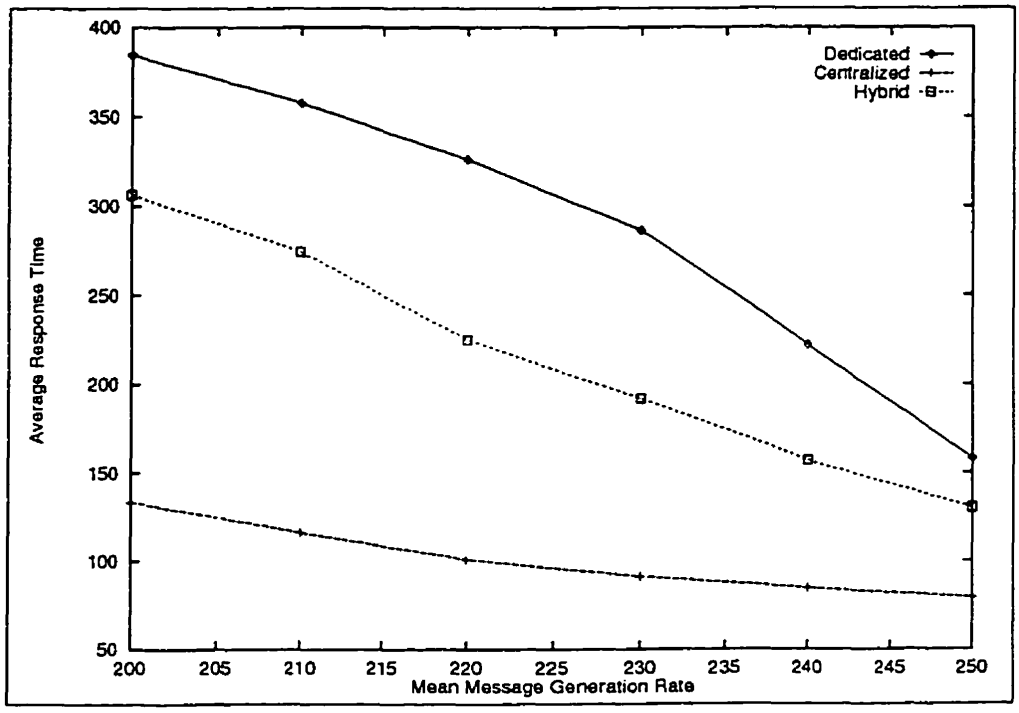


Figure 5.6: Hot-spot traffic: hot messages delay, exponential message generation

until they reach the destination. Because the links towards the hot-spot node become very busy and that in turn slows down the delivery of even regular messages in dedicated organization. Again, the centralized paying its initial overhead, responds to this condition in the network by increasing the buffer capacity of the highly demanded links and keeps latency robust.

The behavior of hybrid could be again because of its partial centralized and partial dedicated behavior. At higher λ the messages reach faster than dedicated because of the dynamic behavior of its router. But as it is limited it behaves worse than dedicated at higher λ.

Hot delay

In this set of results the delay suffered by the hot-messages (messages whose destination is hot-spot node) in the network are given. It is very clear from Figure

5.6 that the advantage of centralized buffer organization is fully exploited in this category. The centralized buffer organization delivers the hot-messages much faster and is stable when compared to the dedicated organization. Again the performance of the hybrid organization is intermediate between the other two.

In summary, for uniform traffic with exponential message generation and fixed message length, dedicated buffer organization is more suitable than the centralized buffer organization. The hybrid closely follows the dedicated buffers.

The centralized buffer organization is better than dedicated for hot-spot traffic. The centralized organization takes more time to deliver the regular messages at higher λ when compared to the dedicated and hybrid organizations. But, its very good behavior towards hot-spot messages and total messages generated in the network puts it in the first place.

5.3 Hyper-Exponential Message Generation

In this experiment each node in the network generates messages in hyper-exponential distribution. The message length is fixed 20 flits. This scenario is similar to a node communicating with other nodes in the network in clusters of time. For example, when a processor is synchronizing with other processors in the network by sending messages to all the nodes and once it gets synchronized, it starts executing instructions locally. And, after the local processing is done (i.e., after a gap) it might again try to communicate with all other nodes.

5.3.1 Uniform traffic

It can be seen from Figure 5.7 that the hybrid buffer organization has a better performance than other two organizations for $\lambda \geq 220$. For $\lambda < 220$ it has slightly worse performance than the dedicated buffer organization. This could be attributed to its partial capability of responding to the dynamic conditions in the network. As

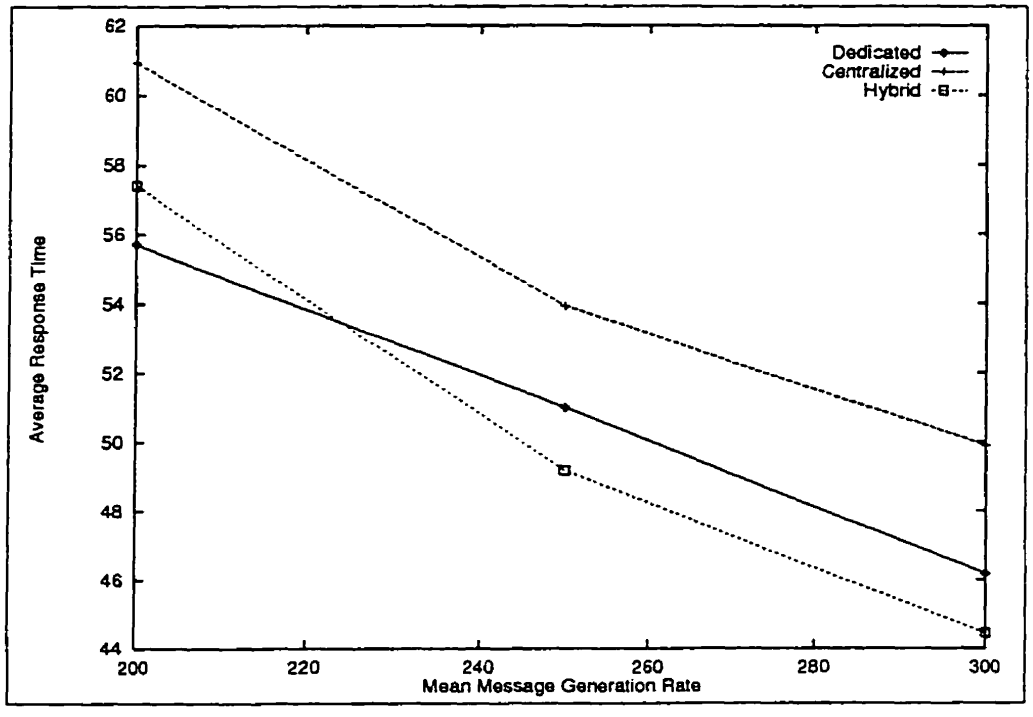


Figure 5.7: Uniform traffic: hyper-exponential message generation

in previous set of results centralized has the worst behavior. And, every organization has higher latency when compared to previous set of results.

5.3.2 Hot-spot traffic

Total delay

The Figure 5.8 indicates that the results for hyper-exponential message generation and exponential message generation. This set of results exhibit the same trend as in base class, with a slight increase in the latency. In this case too, the centralized has the best behavior, dedicated has the worst behavior, and the performance of the hybrid is intermediate between that of the other two.

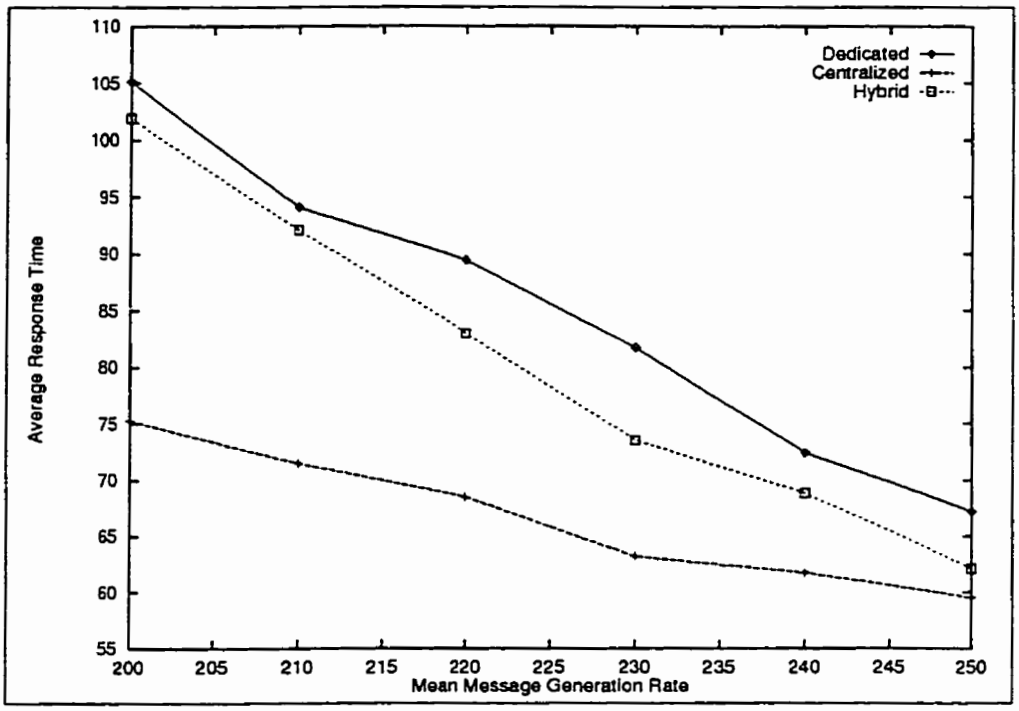


Figure 5.8: Hot-spot traffic: total delay, hyper-exponential message generation

Regular delay

The results given in Figure 5.9 for regular messages are similar to those for the regular messages in the previous set of results. But, the break point where the performance of dedicated and hybrid is worse than the centralized occurs at a much earlier stage (higher $\lambda = 230$).

Hot delay

The results given Figure 5.10 for hot-messages are almost similar to the results in Figure 5.6. But, the messages suffer slightly more delay when compared to the messages generated in exponential distribution.

In summary, for uniform traffic hybrid has much better performance than the other at low message generation rate and it is slightly worse than the dedicated at higher message generation rate. Centralized has the worst behavior. Therefore for

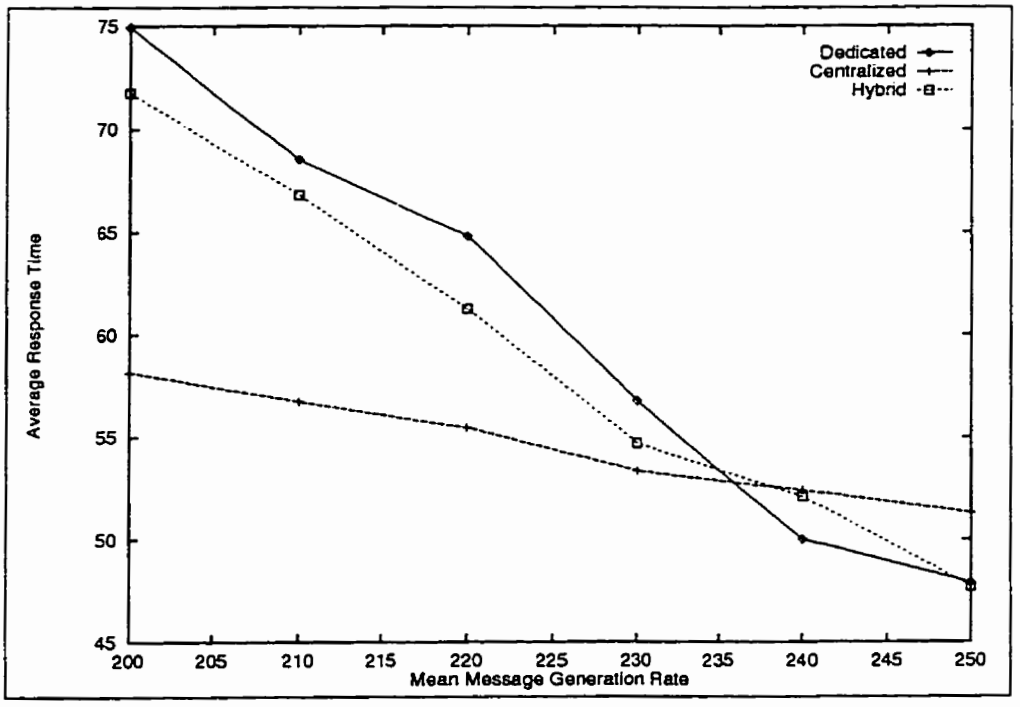


Figure 5.9: Hot-spot traffic: regular messages delay, hyper-exponential message generation

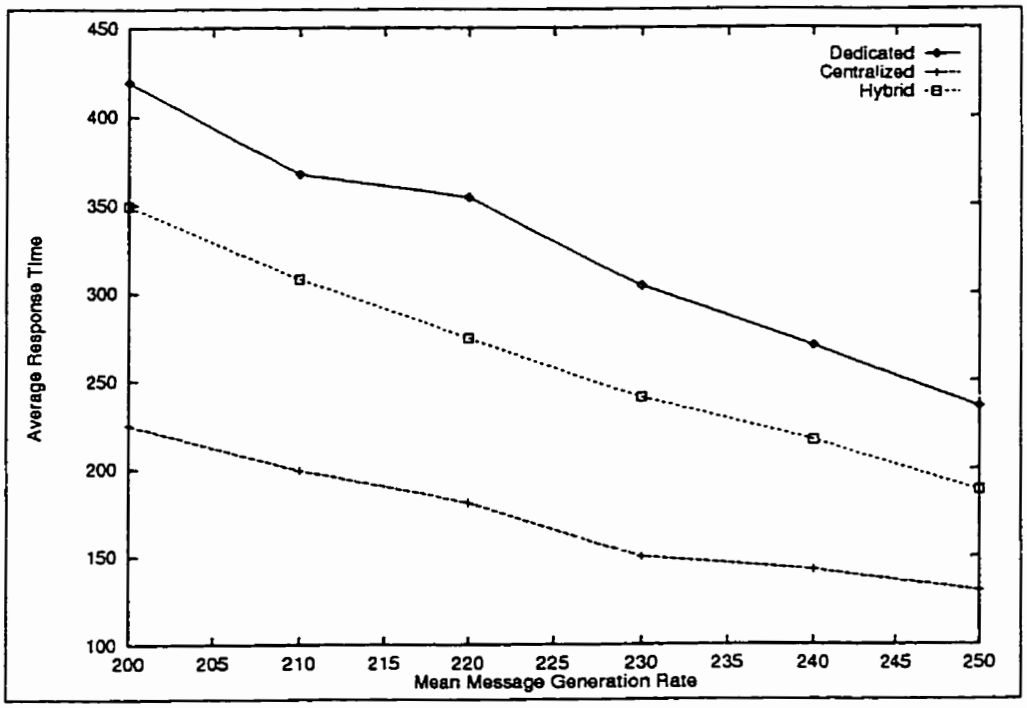


Figure 5.10: Hot-spot traffic: hot messages delay, hyper-exponential message generation

uniform traffic, with hyper-exponential message generation, hybrid organization is preferred .

For hot-spot traffic, as in the previous section, centralized buffer organization is more suitable for this kind of message generation too. Dedicated has the worst performance and hybrid falls in between the other two.

5.4 Variable Message Length

In this experiment the length of the messages generated by each node in the network is varied. To be specific, for a given mean message length (20 flits) the node generate messages in hyper-exponential distribution. Therefore, a node generates several short messages and few very long messages. This scenario is similar to video transmission applications, where several short messages are exchanged by the nodes before actual long video data is transmitted.

5.4.1 Uniform traffic

The results of the experiments for the uniform traffic are given in Figure 5.11. In contrast to the previous results for the uniform traffic, this set of results indicate that centralized has a much better performance than the dedicated organization and hybrid is slightly better than dedicated and worse than the centralized organization. The main reason for this behavior is that the long messages holding the resources for long time and leading to conditions in the network somewhat similar to hot-spot behavior. The centralized could respond to dynamic conditions in a better way as it has a central buffer pool.

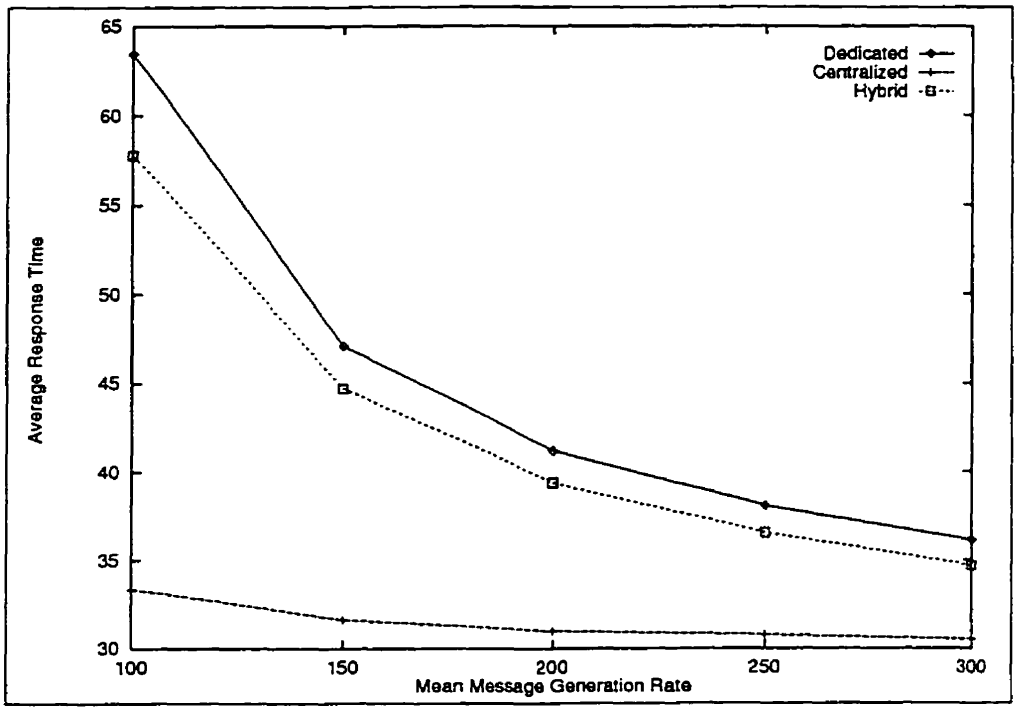


Figure 5.11: Uniform traffic: hyper-exponential message length

5.4.2 Hot-spot traffic

Total delay

The results given in Figure 5.12 for total message delay for hot-spot traffic. The results of this experiment are similar to the results obtained in the previous sections. The centralized organization is much better than the other two. The centralized organization takes less time to transmit messages in this scenario than the base class results. The other two have worse performance than the base class.

Regular delay

The results for regular messages for this scenario are given in Figure 5.13. The trend of the results obtained are similar to the base class results. But, the centralized is always performing better than dedicated, and hybrid is performing better than centralized until $\lambda > 230$ and after that the performance deteriorates rapidly.

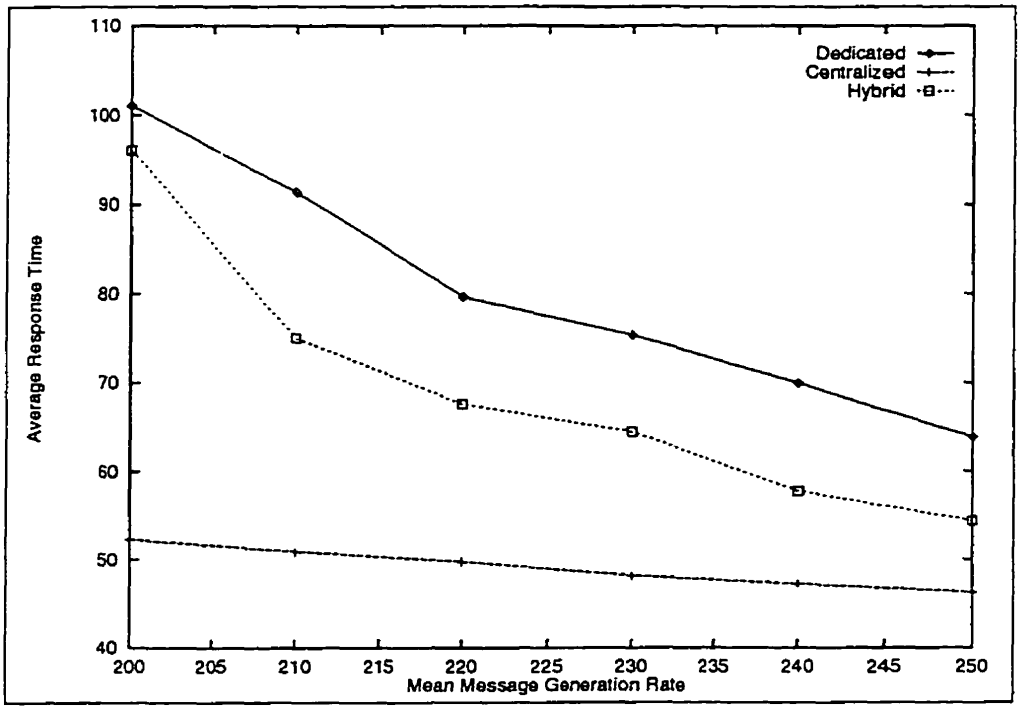


Figure 5.12: Hot-spot traffic: total delay, hyper-exponential message length

Hot delay

The results given in Figure 5.14 for this category indicate that the centralized buffer organization has a very good performance for hot-messages when compared to the dedicated and the hybrid is almost in the middle of the other two. The performance of centralized is better than base class results for this category. The other two have slightly worse than the base class results for this category.

In summary, for this scenario, the centralized outperforms the dedicated not only for the hot-spot traffic but also for the uniform traffic. The performance of the hybrid buffer organization is intermediate between that of the other two.

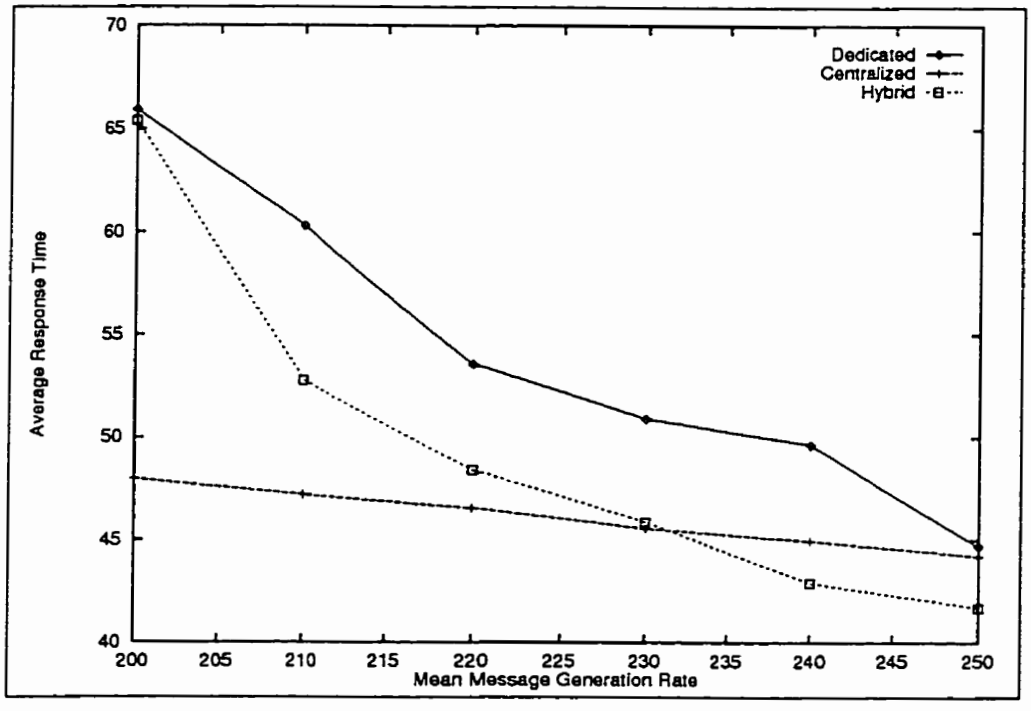


Figure 5.13: Hot-spot traffic: regular messages delay, hyper-exponential message length

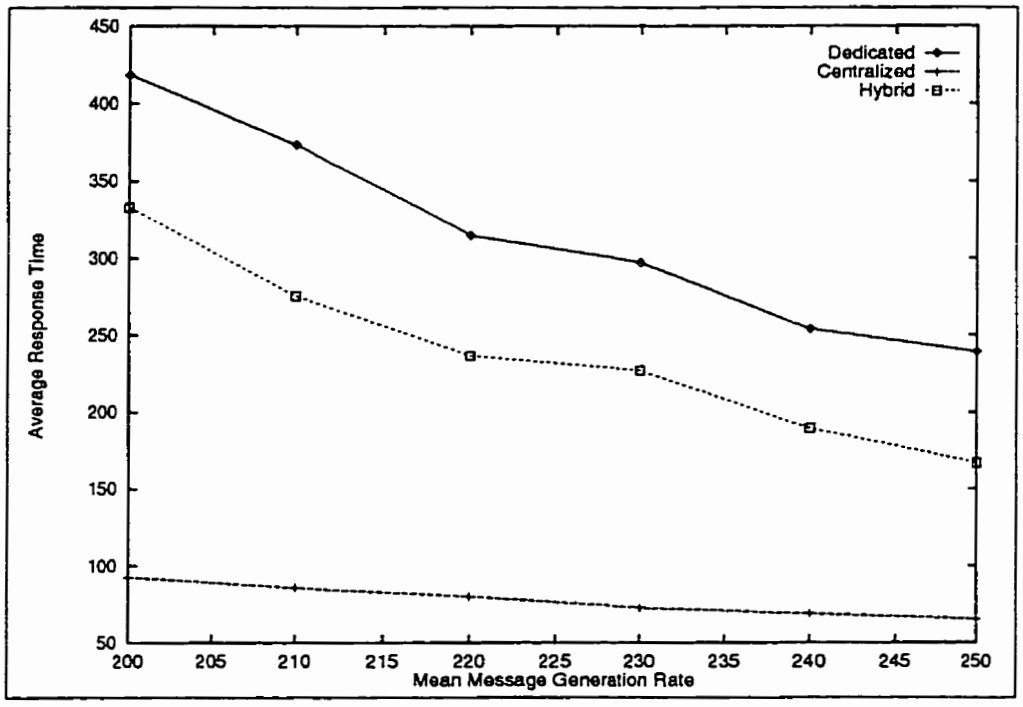


Figure 5.14: Hot-spot traffic: hot messages delay, hyper-exponential message length

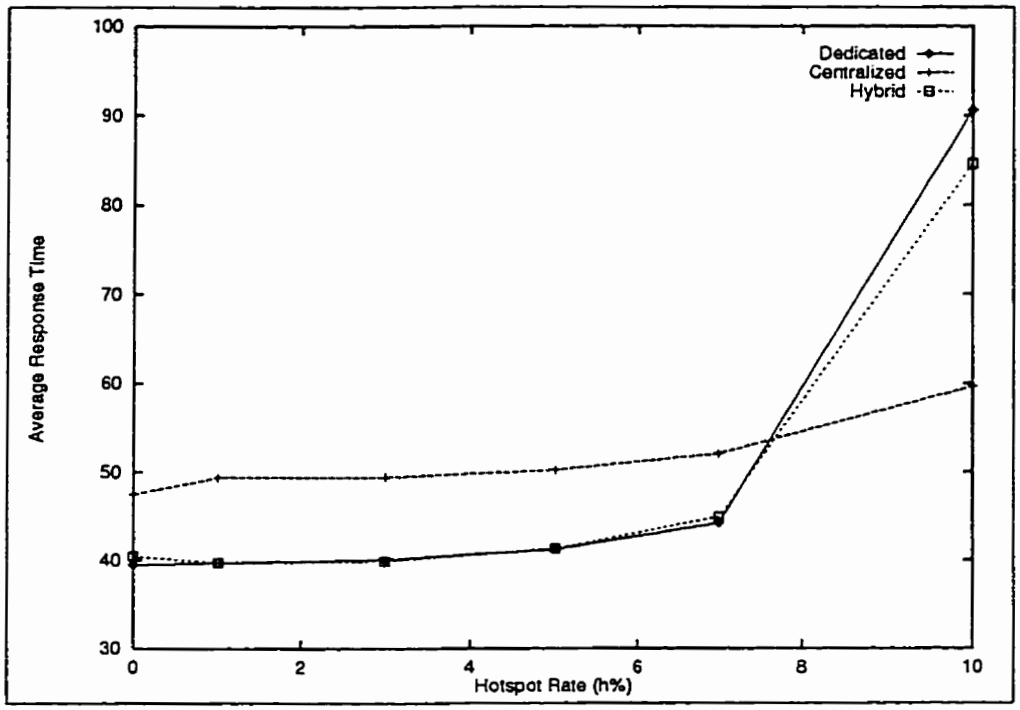


Figure 5.15: Hot-spot traffic: total delay, mean 200

5.5 Impact of Percentage of Hot-Spot Messages

In all the previous sections the results for hot-spot traffic are given for h=10%. The percentage of hot-spot messages indicate the fraction of messages going to the hot-node from a node. The h has been varied from 0 to 10. At $h = 0$, the traffic is uniform and as the h increases the number of hot-messages generated per unit time by each node also increases.

5.5.1 Total delay

The total delay suffered by the messages in three different organizations is given in Figure 5.15. The figure indicates that until $h < 7\%$ the dedicated has a better performance than the centralized and beyond that its performance deteriorates rapidly. The hybrid buffer organization closely follows dedicated at lower h ($h \leq 7$) and has a better performance at higher h than the dedicated. The behavior of centralized is

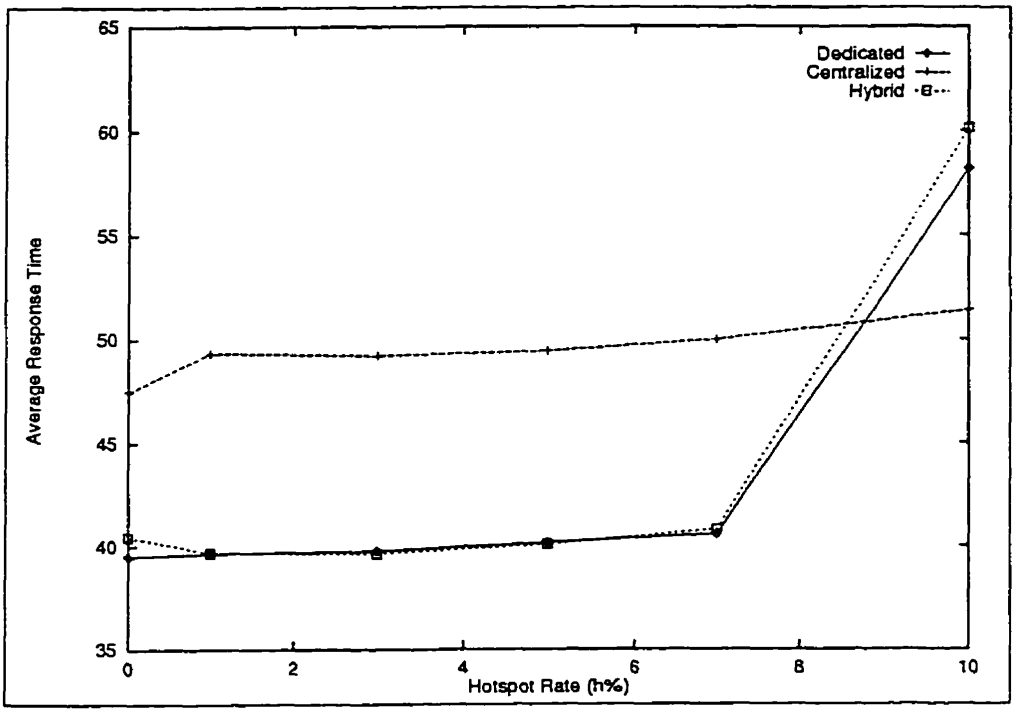


Figure 5.16: Hot-spot traffic: regular messages delay, mean 200

more stable after the initial overhead.

5.5.2 Regular delay

The set of results for this experiment for regular messages is given in Figure 5.16. The trend of the results is similar to the results obtained in this experiment for the total delay category. The hybrid is slightly worse than the dedicated at higher h.

5.5.3 Hot delay

The delay suffered by the hot-messages with varying percentage of h are given in Figure 5.17. The data presented in this figure are particularly interesting. Until $h \leq 7$ the three buffer organizations deliver the hot-messages with similar delay. After that the centralized has a much better performance than the dedicated and the performance of the hybrid is intermediate to that of the other two.

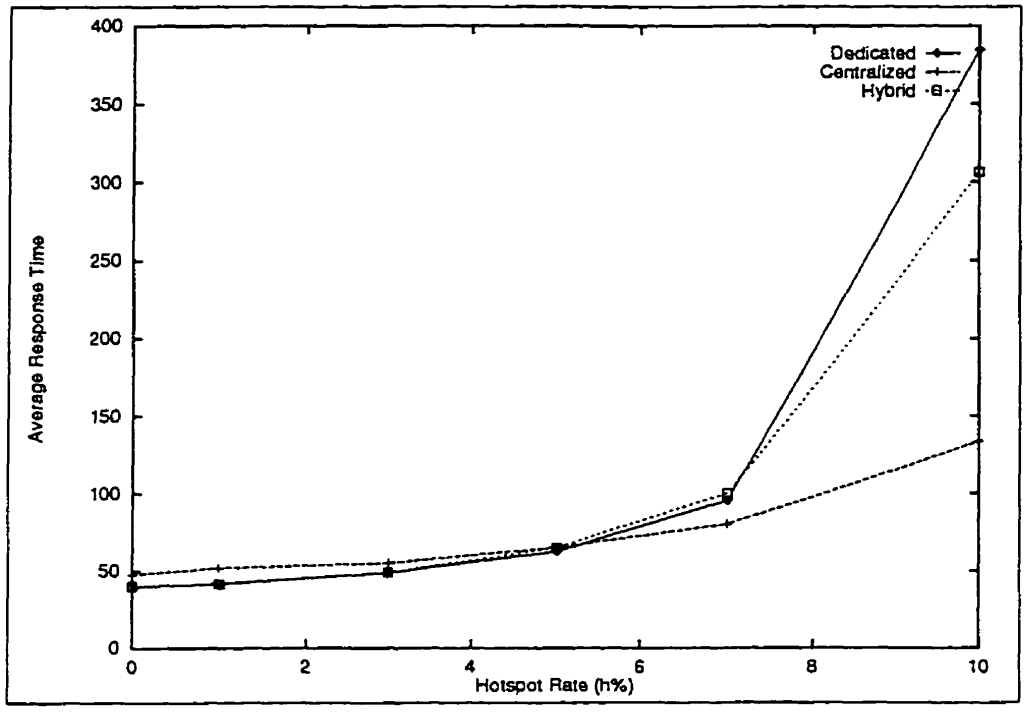


Figure 5.17: Hot-spot traffic: hot messages delay, mean 200

In summary, the percentage of hot-spot messages has a significance impact on the performance of the network. Until $h \leq 7$ dedicated and hybrid have similar behavior for all the categories. The centralized takes more time (total and regular) intially but its behavior is more robust. The centralized has a similar delay as the dedicated until $h = 7$ for hot-messages. For $h > 7$, the centralized is much better than dedicated. The hybrid is in the middle of the other two. Therefore, depending upon the percentage of hot-spot messages switching between centralized and dedicated is beneficial.

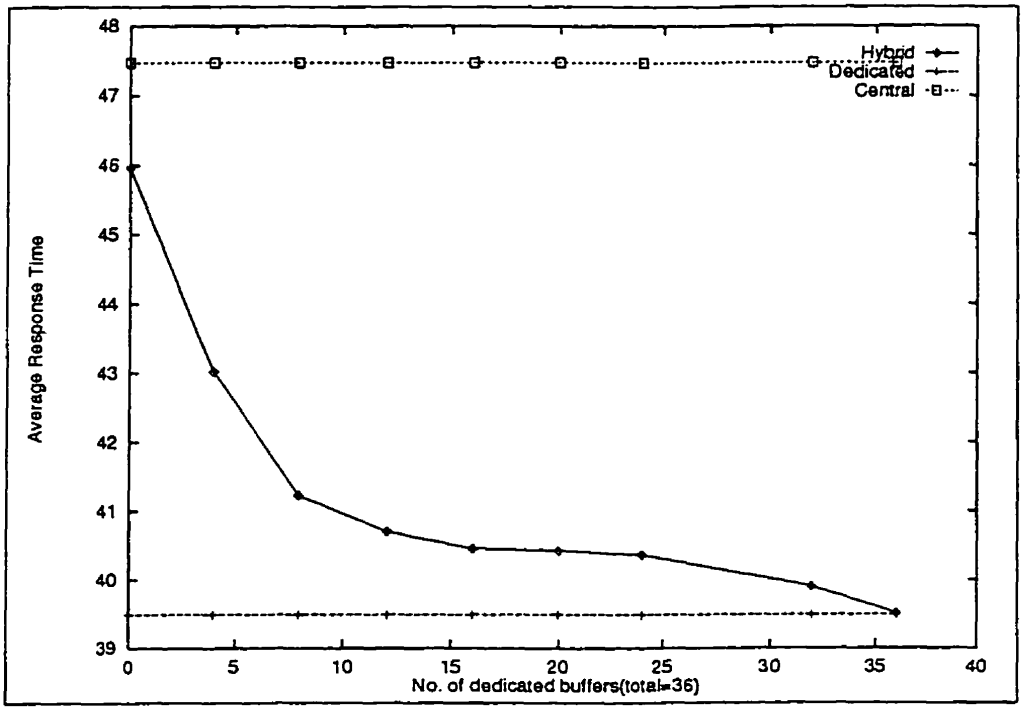


Figure 5.18: Hybrid dedicated buffers: uniform Traffic $\lambda = 200$

5.6 Right Combination of Buffers for Hybrid Organization

The final experiment is to find the right combination of dedicated and centralized buffers for hybrid buffer organization. In all the experiments conducted before the number of dedicated buffers is fixed at 16 and the centralized buffers are fixed at 20 (with a total of 36 buffers per node). In this experiment the number of dedicated buffers is changed from 0 to 36. For these experiments the messages are generated using the exponential distribution, the message length is fixed at 20 flits, and the mean message generation rate (λ) is 200.

5.6.1 Uniform traffic

The results of this experiment for the uniform traffic are given in Figure 5.18. The results indicate that when the number of dedicated buffers is zero, the hybrid buffer

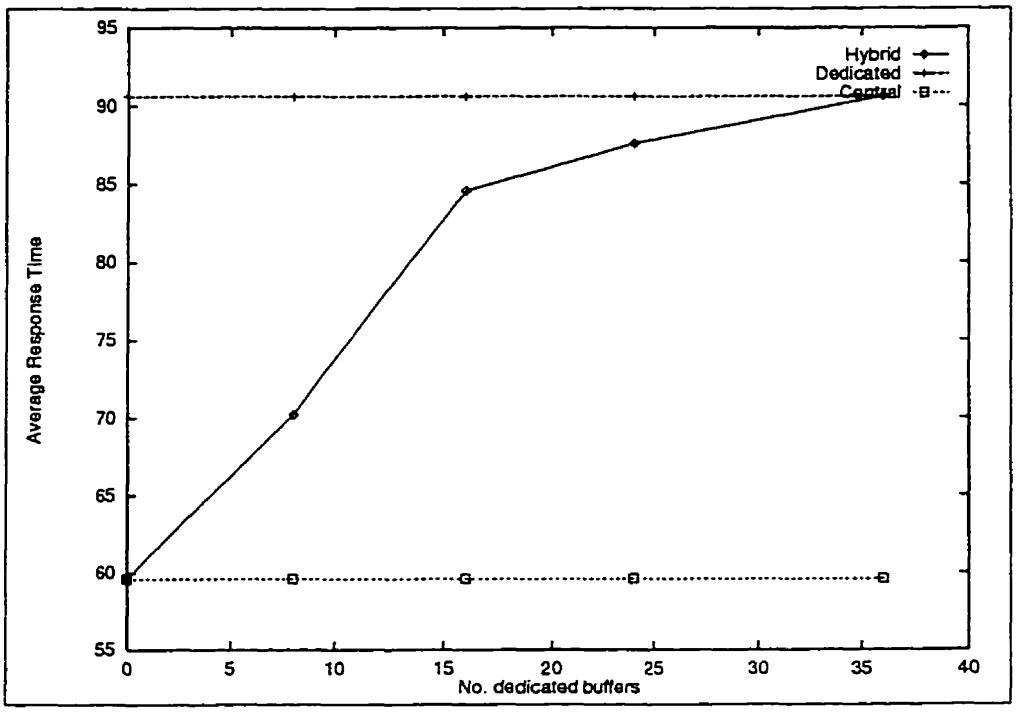


Figure 5.19: Hybrid dedicated buffers: hot-spot traffic (Total delay) $\lambda = 200$

organization is similar to centralized buffer organization, and when it is 36 the hybrid is similar to dedicated. The performance of the hybrid improves with increase in dedicated buffers. But, the degree of improvement is low after 15 dedicated buffers. The performance is almost stable until the number of dedicated buffers increases to more than 25.

5.6.2 Hot-spot traffic

The results of this experiment for hot-spot traffic are given in three different figures for total, regular and hot delays. Figure 5.19, figure 5.20, figure 5.21 present the results for this case for total, regular, and hot delays respectively. The results indicate that the performance of hybrid buffer organization is similar centralized when all the buffers are centralized . Hybrid performs like dedicated when all the buffers are dedicated. The performance of hybrid is decreasing with increase in the number of dedicated buffers under hot-spot traffic.

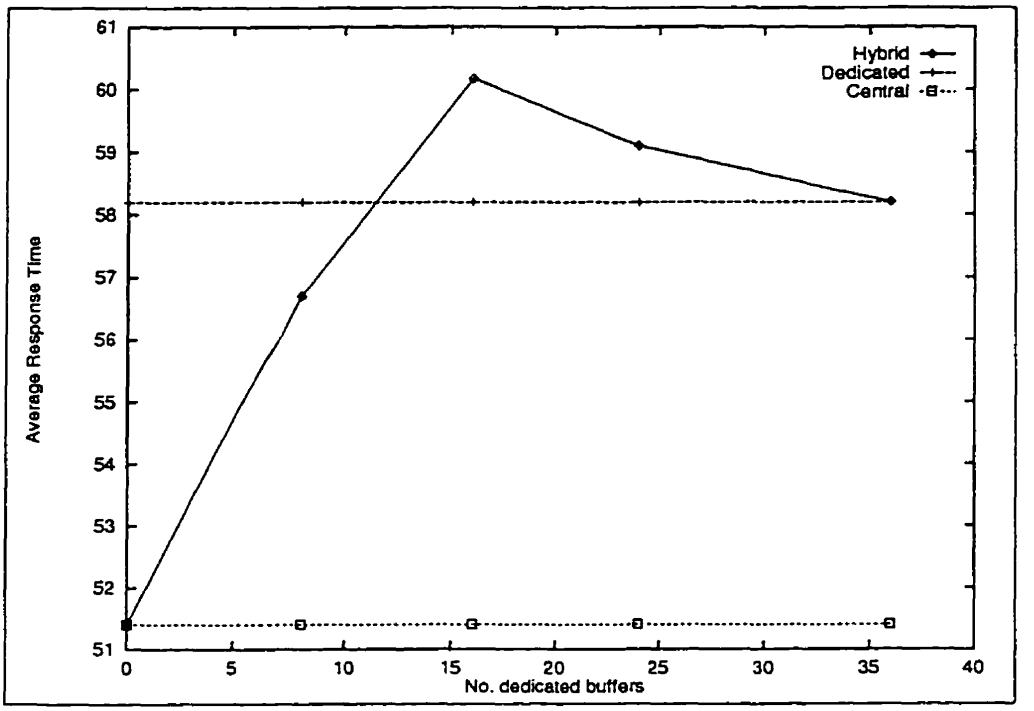


Figure 5.20: Hybrid dedicated buffers: hot-spot traffic (Regular delay) $\lambda = 200$

5.7 Conclusions

The dedicated buffer organization is better for uniform traffic in all scenarios except the one in which many short massages are transmitted after a very long message. The centralized buffer organization has very poor performance for uniform traffic in comparison to dedicated. The centralized is better than the dedicated in the workload that consists of several short messages and few very long messages (ex. video transmission).

The centralized buffer organization has a better performance and more robust under hot-spot traffic. It is extremely good in transmitting hot-messages when compared to the dedicated organization. The regular messages take little more time at lower λ when compared dedicated, but with the initial overhead, the centralized buffer organization gives a stable performance.

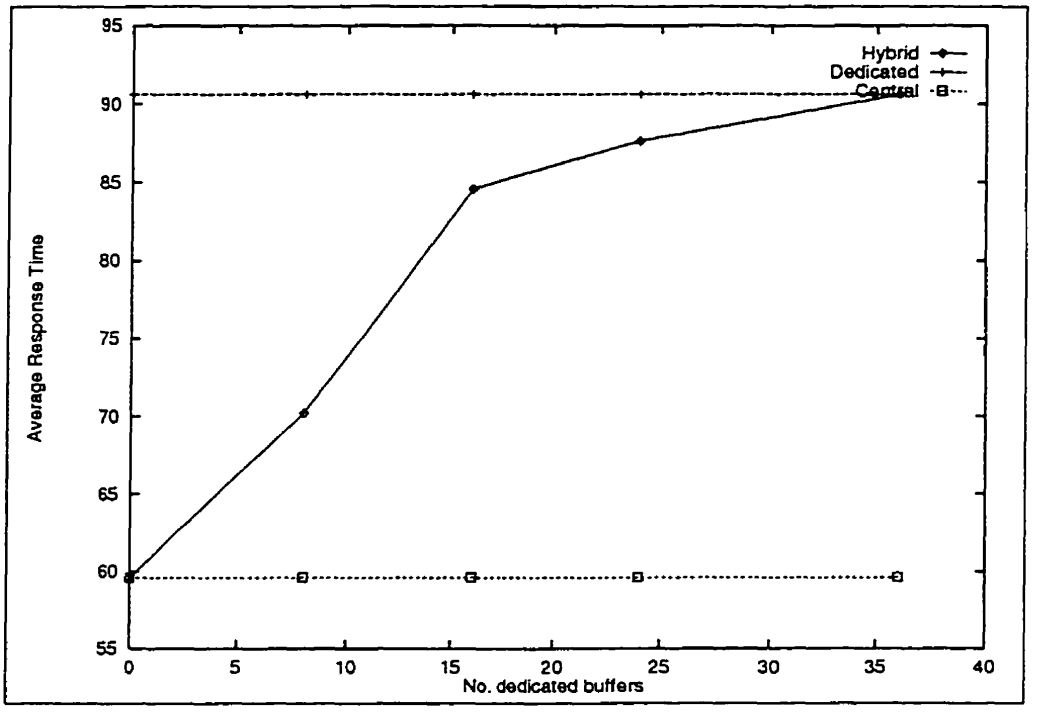


Figure 5.21: Hybrid dedicated buffers: hot-spot traffic (Hot delay)$\lambda = 200$

Hybrid buffer organization closely follows dedicated buffer organization in uniform traffic and its performance is intermediate to that of the other two organizations in hot-spot traffic.

The experiments conducted to study performance sensitivity to h indicate that at lower h (say $h < 7$), the dedicated has a better performance than the centralized organization and after that point centralized outperforms the dedicated organization. Hybrid follows the dedicated at lower h and after that it provides intermediate performance between that of the centralized and dedicated organizations. Therefore, depending upon the dynamic traffic conditions the hybrid organization can configure itself from dedicated to centralized, which could result in a good performance.

Chapter 6

Conclusions and Future Work

6.1 Summary

This section provides a summary of the thesis. Chapter 1 introduced several basic concepts of multicomputers. The important issues covered were: parallel architectures, multicomputer networks and the goal of this thesis.

Chapter 2 provided a detail description and discussion of the various issues in wormhole routed multicomputer networks. The main topics covered were: deadlocks, virtual channels, deterministic and adaptive routing, and NHop routing algorithm. nCUBE system, an example multicomputer system that uses wormhole switching was also described.

Chapter 3 presented a close look at the buffer management issues in wormhole routed multicomputer networks. The focus of this chapter had been on dynamic assignment of buffers, amount of buffer space and buffer organizations.

Chapter 4 described the system model used in the simulation experiments. Chapter 5 presented the results of the simulations experiments. Each experiment was run for both uniform and hot-spot traffics.

6.2 Contributions of the thesis

The commonly used buffer organizations for recent multicomputers are centralized and dedicated buffer organizations. The results presented in this thesis indicate that the dedicated buffer organization has a better performance than the centralized under the uniform traffic in all scenarios except one when the nodes are generating messages in clusters of message length.

The centralized buffer organization has a better performance and more robust under the hot-spot traffic. It is extremely good in transmitting hot-messages when compared to the dedicated organization. The regular messages take little more time at lower message generation rates when compared the dedicated. but the centralized buffer organization gives a stable performance.

The *hybrid* buffer organization proposed in this thesis inherits the merits of the centralized and dedicated organizations. Hybrid buffer organization performs similar to the dedicated buffer organization under uniform traffic and its performance is intermediate to that of the other two organizations for hot-spot traffic. The hybrid buffer organization can be designed to configure itself dynamically from the dedicated to centralized depending on the traffic conditions.

In practice, the traffic is bounded between the hot-spot and uniform traffic. Therefore. it is necessary to handle these two cases in an efficient way. Hybrid buffer organization can act as the centralized organization for the hot-spot traffic and it can work like the dedicated organization for the uniform traffic. Thus, this new organization is beneficial for the wormhole routed multicomputer systems.

6.3 Future Work

There are several aspects of the *hybrid* buffer organization that need to be studied or expanded in the future. Performance of hybrid buffer organizations needs to be

analyzed with more realistic workloads.

One area that requires study is in the design of a programmable router that uses the hybrid buffer organization which could respond to the dynamic conditions in the network.

Another area is to study the performance of *-channel algorithm with these three different buffer organizations. Also, a comparison of NHop and *-channel is interesting.

References

[1] D.A. Reed and R.M. Fujimoto, "Multicomputer Networks: Message Based Parallel Processing", MIT Press, Cambridge, Mass., 1987.

[2] J. Duato, S. Yalamanchili, and L.Ni, "Interconnection Networks: An Engineering Approach," IEEE Computer Society Press, 1997.

[3] S.P. Dandamudi, "Hierarchical Hypercube Multicomputer Interconnection Networks," Ellis Horwood Series in Computers And their Applications, 1991.

[4] L.W. Tucker and G.G. Robertson, "Architecture and Applications of the Connection Machine", *IEEE Transactions on Computer*, vol. 21, no. 8, pp. 26-38.

[5] R. Duncan, "A Survey of Parallel Computer Architectures", *IEEE Transactions on Computer*, Feb 1990.

[6] K. Hwang, ed., "Tutorial Supercomputers: Design and Applications," Computer Society Press, Los Alamitos, Calif., 1984.

[7] K. Hwang and F. Briggs, " Computer Architectures and Parallel Processing", *McGraw Hill*, New York, 1984.

[8] T. Lee "Content-Addressable Memories," *2nd edition*, Springer-Verlag, New York, 1987.

[9] H.T. Kung "Why Systolic Architectures?," *Computer*, Vol. 15, No. 1, Jan. 1982, pp. 37-46.

[10] L.M. Ni and P.K. McKinley,"A Survey of Wormhole Routing Techniques in Direct Networks", *IEEE Transactions on Computer*, Feb., 1993.

[11] P.E. Berman, et.al, "Adaptive Deadlock- and Livelock-free Routing with all Minimal paths in Torus networks," *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 3-12, June 1992.

[12] R.V. Boppana and S. Chalasani, "A Comparison of Adaptive Wormhole Routing Algorithms," *Proceedings of the 20th International Symposium on Computer Architecture*, pp. 351-360, May 1993.

[13] S. Chalasani and R.V. Boppana, "Fault-Tolerant Wormhole Routing in Tori," *Proceedings of the 8th International Conference on Supercomputing*, pp. 146-155, July 1994.

[14] R.V. Boppana and S. Chalasani, "A Framework for Designing Deadlock-Free Wormhole Routing Algorithms.",*IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 2, pp. 169-183. Feb 1996.

[15] W.J. Dally, "Performance Analysis of K-ary N-cube Interconnection Networks.", *IEEE Transactions on Computers*, vol. C-39, no. 6, pp. 775-785.

[16] W.J. Dally, "Virtual-Channel Flow Control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-205, March 1992.

[17] W.J. Dally and H. Aoki, "Deadlock-free Adaptive Routing in Multicomputer Networks using Virtual Channels," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466-475, April 1993.

[18] W.J. Dally and C.L. Seitz,"Deadlock-free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547-553, May 1987.

[19] S. Ramany and D. Eager, "The Interaction between Virtual Channel Flow Control and Adaptive Routing in Wormhole Networks," *ACM Internation Conferance on Supercomputing*, Manchester, 1994.

[20] M. Schmidt-Voigt, "Efficient Parallel Communication with the nCUBE 2S Processor," *Parallel Computing*, April 1994.

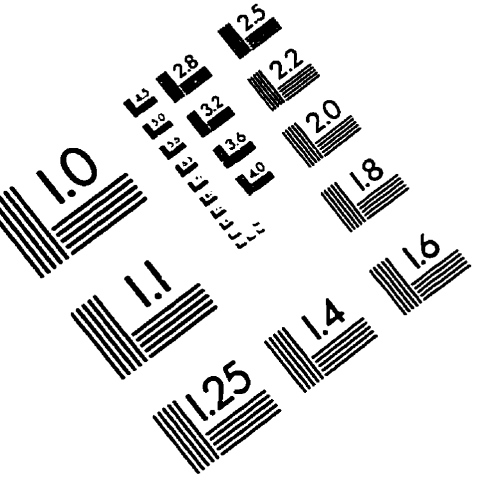
[21] J. Struckemier and F.J. Pfreundt, "On the Efficiency of Simulation Methods for Boltzmann equation on Parallel Computers," *Parallel Computing*, 19(1993)-119.

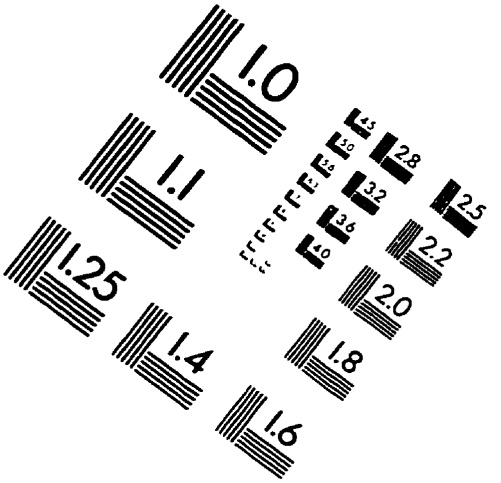
[22] J. Bozman, "Bank Moves to Oracle on Supercomputer", *Computerworld*, 27(7) (Feb. 1993).

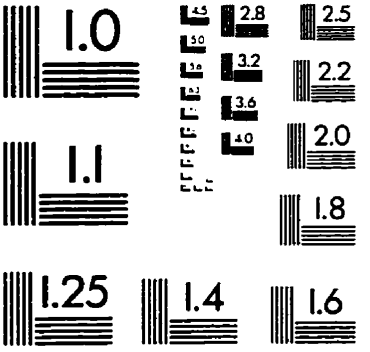
[23] D.M. Dias and J.R. Jump, "Packet Switching Interconnection Networks for Modular Systems," *IEEE Computer*, Vol. 14, No. 12, pp. 43-45, December 19981.

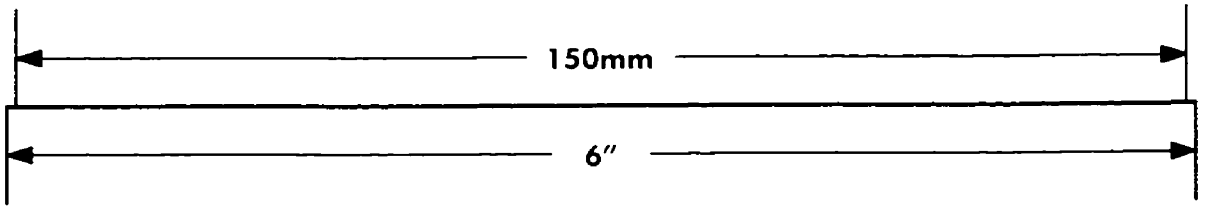
[24] K. Kotapati and S.P. Dandamudi, " Buffer Management in Wormhole Routed Torus Multicomputer Networks," Technical Report, School of Computer Science, Carleton University, 1997 (Submitted for publication).

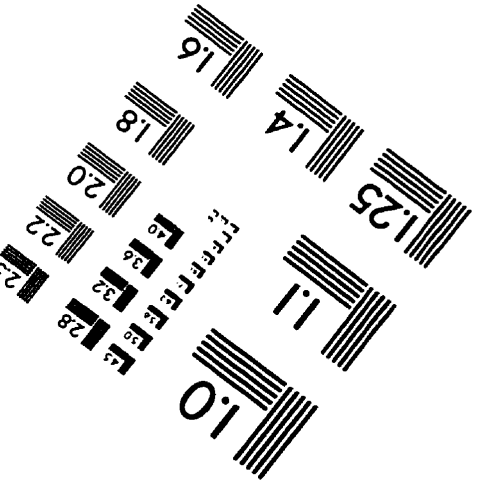
IMAGE EVALUATION
TEST TARGET (QA-3)

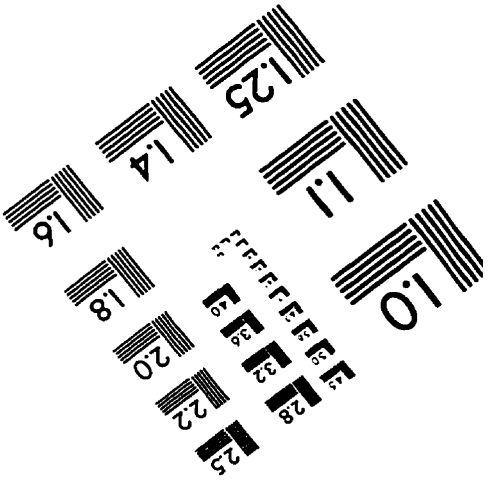












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989