

**DATA RECOVERY IN WORMHOLE ROUTING NETWORKS
IN HYPERCUBES AND MESHES**

BY

Mohammad S. Alowayed

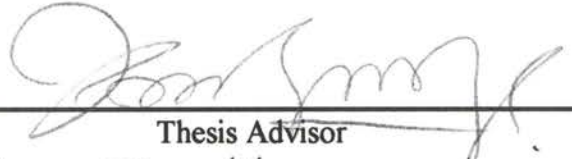
**Bachelor of Science
King Saud University
Riyadh, Saudi Arabia
1989**

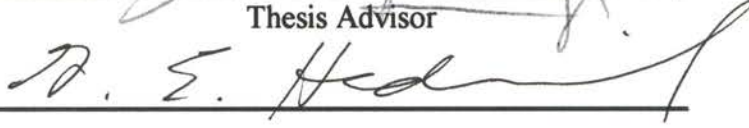
**Master of Science
Western Michigan University
Kalamazoo, MI
1993**

**Submitted to the Faculty of the Graduate College
of the Oklahoma State University in partial
fulfillment of the requirements for
the Degree of
DOCTOR OF PHILOSOPHY
December, 1997**

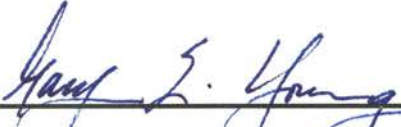
DATA RECOVERY IN WORMHOLE ROUTING NETWORKS
IN HYPERCUBES AND MESHES


Thesis Approved:



Thesis Advisor


H. Lu





Dean of the Graduate College

ACKNOWLEDGMENTS

All the praises and thanks be to the Al-Mighty **ALLAH**. This study could not have been completed without the blessing, grace, and guidance of the Al-Mighty **ALLAH**.

I wish to express my deep gratitude and sincere appreciation to my major advisor **Prof. K.M. George** for his supportive suggestions, constructive comments, and critical insights throughout this thesis. My appreciation extends to my other committee members **Dr. G. Young, Dr. G. Hedrick and Dr. H. Lu** for their valuable suggestions and encouragement during the course of this project.

I would like to give my special appreciation to my parents for the love and support they have given me over the years. I would like to thank my wife, **Amal**, for the sacrifice, patience, and support during the period I have spent with her in my graduate study.

I would like to thank **Abdul-Rahman Aljadhi, Hussain Alsalman, Faisal Alruhaili, Mohammad Alahmad, Wail Alrashed, Dukhail Al-Dukhail, and Abdul-Malik Alsalman** for their support and encouragement.

TABLE OF CONTENTS

Chapter	Page
1. INTRODUCTION	1
1.1 Related Work	3
1.2 Thesis Outline	5
2. INTERCONNECTION NETWORK	6
2.1 Distributed vs. Shared Memory.....	6
2.2 Network Topologies.....	8
2.2.1 Hypercube.....	9
2.2.2 Mesh	10
2.3 Switching Techniques.....	11
2.3.1 Circuit Switching.....	14
2.3.2 Store and Forward Switching.....	15
2.3.3 Wormhole Switching.....	17
2.4 Virtual Channels.....	20
2.5 Message Passing.....	22
3. ROUTING ALGORITHMS.....	27
3.1 Deadlock, Livelock and Starvation.....	28
3.1.1 Deadlock Avoidance.....	30

Chapter	Page
3.1.2 Deadlock Recovery.....	36
3.2 Deterministic Routing.....	38
3.3 Adaptive Routing.....	41
3.3.1 Turns Restrictions Techniques.....	42
3.3.2 Virtual Networks Technique.....	45
3.3.3 Escape Channels Technique.....	46
3.3.4 Adaptive Algorithms Based on Deadlock Recovery.....	49
3.4 Fault Tolerant Algorithms.....	51
4. FAULT RECOVERY	58
4.1 Reliable Router (RR).....	61
4.1.1 Router Design in RR and Hardware Requirements.....	64
4.2 Fault-tolerant Compressionless Routing (FCR).....	68
4.2.1 Hardware Support for Data Recovery in FCR.....	70
4.3 Acknowledged Pipelined Circuit-Switching (APCS).....	72
5. SOFTWARE BASED RECOVERY PROTOCOL.....	75
5.1 General Concepts of Software Recovery Protocols.....	76
5.1.1 Fault Recovery Handler	77
5.1.2 Absorbing Orphan Flits.....	78
5.1.3 Resending Handler	78
5.2 Implementation.....	79
5.2.1 SBRP-0.....	79
5.2.2 SBRP-1.....	80

Chapter	Page
5.2.3 SBRP-2.....	84
5.3 Software Overhead in SBRP.....	85
5.4 Comparison between the approaches of SBRP.....	86
5.5 Comparison between SBRP and other recovery protocols.....	87
6. SIMULATION AND PERFORMANCE ANALYSIS.....	89
6.1 Simulation Configuration.....	91
6.2 Performance Metrics.....	92
6.3 Fault-Free Performance.....	94
6.3.1 Virtual Channels.....	96
6.3.2 Routing Algorithms.....	96
6.3.3 Traffic Patterns.....	101
6.4 Static Fault Performance.....	106
6.5 Dynamic Fault Recovery Performance.....	107
7. CONCLUSIONS.....	116
7.1 Summary of work.....	116
7.2 Conclusion.....	117
7.3 Open Topics.....	119
REFERENCES.....	120

LIST OF TABLES

Table		Page
1	One-way message overhead.....	24
2	Comparison between SBRP protocols.....	87
3	Comparison between SBRP protocols.....	88

LIST OF FIGURES

Figure		Page
1	A generic multiprocessor machine connected by interconnection network.....	1
2	A generic node architecture.....	2
3	Shared memory and distributed memory.....	7
4	8-nodes hypercube.....	9
5	16-node hypercube.....	9
6	4×4 2-dimensional mesh.....	10
7	3×3×3 3-dimensional mesh.....	10
8	Router model.....	12
9	The physical connection of a uni-direction link.....	13
10	The handshaking protocol.....	13
11	Circuit Switching.....	15
12	Store-and-Forward Switching.....	16
13	Wormhole Routing Switching.....	18
14	Format of wormhole packet in the CRAY T3D.....	18
15	Wormhole Routing.....	19
16	Four virtual channels share a physical channel.....	21
17	Virtual channels provide additional buffers allowing message B to pass blocked message.....	21

Figure	Page
18	Message transmission and reception..... 23
19	Two Phase Messaging implementation 26
20	A deadlock situation involving four messages 29
21	Interconnection graph and dependency graph of example-1..... 32
22	Interconnection graph and dependency graph of example-2..... 33
23	Channel dependency graph of example-3..... 34
24	The extended dependency graph of example-3..... 35
25	Dimension Order (XY) Routing in a 2d Mesh..... 40
26	Deterministic and adaptive routing..... 42
27	Possible and allowable turns in a 2d Mesh..... 43
28	Examples of West-First Routing in a 2d Mesh..... 43
29	Possible paths from 1010 to 0101 in P-routing..... 44
30	Virtual networks for a 2-d mesh in double-y algorithms..... 46
31	Possible paths from 1010 to 0101 in Duato -routing..... 47
32	Turns in opt-y 49
33	Router model of Disha..... 51
34	Fault tolerant routing in a faulty hypercube..... 53
35	Usage of virtual channels in Planar algorithm..... 55
36	Routing around faulty region in Planar algorithm..... 56
37	Subnetworks in planes A0 and A1 used by Planar algorithm..... 57
38	Reliable messaging in traditional systems..... 59
39	The UTP at the flit-level..... 61

Figure	Page
40-a	Message in RR before the fault..... 62
40-b	Message in RR after the fault..... 62
41	Flow control in reliable router..... 63
42	Input controller block diagram of RR..... 65
43	Virtual channel model of RR..... 66
44	State diagram of the Route FSM..... 67
45	Tearing down an interrupted circuit..... 69
46	Message Padding in FCR network..... 69
47	Data and control lines in FCR..... 70
48	The injector network interface in FCR..... 71
49	The message receiver interface in FCR..... 72
50	Acknowledge signal in APCS..... 73
51	A message pipeline with a faulty node..... 77
52	Normal message format..... 82
53	A n-flit message fragmented to two sub-messages..... 82
54	Virtual channel module for SBRP..... 82
55	Simulated routed model..... 90
56	Latency versus normalized load..... 95
57	Normalized throughput versus normalized load..... 95
58	Effect of virtual channel number in latency (Mesh topology)..... 97
59	Effect of the number of virtual channels in throughput (Mesh topology)..... 97
60	Effect of virtual channel number in latency (Hypercube topology)..... 98

Figure	Page
61	Effect of the number of virtual channels in throughput (Hypercube)..... 98
62	Performance of routing algorithms in Mesh (Load vs. Latency)..... 100
63	Performance of routing algorithms in Mesh (Load vs. Throughput)..... 100
64	Performance of routing algorithms in Hypercube (Load vs. Latency)..... 102
65	Performance of routing algorithms in Hypercube (Load vs. Latency)..... 102
66	Effect of routing algorithms in latency under Transpose traffic (Mesh)..... 104
67	Effect of routing algorithms in throughput under Transpose traffic (Mesh)..... 104
68	Effect of routing algorithms in latency under Transpose traffic (Hypercube)..... 105
69	Effect of routing algorithms in throughput under Transpose traffic (Hypercube) 105
70	Performance with static faults..... 106
71	Overall message latency in Hypercube with fault rate = 10^{-4} and message length = 12..... 109
72	Corrupted message latency in Hypercube with fault rate = 10^{-4} and message length = 12..... 109
73	Overall message latency in Hypercube with fault rate = 10^{-4} and message length = 1024..... 110
74	Corrupted message latency in Hypercube with fault rate = 10^{-4} and message length = 1024..... 110
75	Overall message latency in Mesh with fault rate = 10^{-4} and message length = 28 112

Figure	Page
76	Corrupted message latency in Mesh with fault rate = 10^{-4} and message length = 28..... 112
77	Performance of SBRP-0 under different fault rates (Latency of overall messages) 113
78	Performance of SBRP-0 under different fault rates (Latency of corrupted messages)..... 113
79	Comparison between SBRP protocols (Latency of overall messages, message length=8)..... 114
80	Comparison between SBRP protocols (Latency of corrupted messages, message length=8)..... 115
81	Comparison between SBRP protocols (Latency of overall messages, message length=128) 115
82	Comparison between SBRP protocols (Latency of corrupted messages, message length=128)..... 115

GLOSSARY OF TERMS

<i>Flit</i>	A flit is flow control unit. It is the basic message unit upon which flow control is performed in a pipelined network.
<i>Message</i>	A message is a group of flits, consisting of a header flit that contains routing information, data flit(s) that contain information, and a tail flit.
<i>Message Head</i>	A message head is the flit or flits containing routing information of the message. Also, it might contains some control information such as the type and the length of the message and the message sequence number.
<i>Phit</i>	The amount of information that can be transferred in one channel cycle.
<i>Physical channel</i>	A physical channel is a hardware connection between two nodes over which data is transmitted.
<i>Virtual Channel</i>	A virtual channel is a logical entity associated with a physical link used to distinguish multiple data streams traversing the same physical channel. Multiple virtual channels are time multiplexed over a physical channel.
<i>Virtual Path</i>	A virtual path is a set of virtual links through a network that connect source and destination nodes. The path is used to transmit a message from source to destination.
<i>Profitable link</i>	A link over which a message moves closer to the destination.

<i>Minimal Path</i>	A minimal path is the path that contains the minimum number of links from a source to a destination node. Minimal path contains only profitable links. Minimal paths are also called optimal or profitable paths.
<i>Deterministic Routing</i>	A routing protocol is deterministic if there is only one path connecting any pair of nodes.
<i>Adaptive Routing</i>	A routing protocol is adaptive if it is capable of using more than one path from a source to a destination.
<i>Fault-tolerant Routing</i>	A routing protocol is fault-tolerant if the message is deliverable in a network in the presence of faults.
<i>Misrouting</i>	Sending a routing header farther away from the destination by using unprofitable link. This is also called a nonminimal routing.
<i>Router</i>	A logical block that contains a configurable switch that logically connects input channels to output channels. Also, it performs message routing, data pipelining and channel multiplexing.
<i>Bisection</i>	The number of channels that must be removed to partition the network into two equal subnetworks
<i>Diameter</i>	The maximum distance between any two nodes in the network.
<i>Buffer</i>	A temporary storage area in memory. Many methods of routing messages between nodes use intermediate nodes routers' buffers to store messages.
<i>Channel</i>	A point-to-point connection through which messages can be sent
<i>Latency</i>	The time taken to deliver a message.

<i>Multicomputer</i>	A computer in which processors can execute separate instruction streams, can have their own private memories, and can not directly access one another's memories. Most multicomputers are disjoint memory machines, constructed by joining nodes via links.
<i>Node</i>	Basic building block of a multicomputer. Typically a node refers to a processor with a memory system and a mechanism for communicating with other processors.
<i>Virtual Channel</i>	A logical point-to-point connection between two processors. Many virtual channels may time share a single link to hide latency or to avoid deadlock.
<i>Route</i>	The act of moving a message from its source to its destination. A routing algorithm is a rule of deciding, at any intermediate node, where to send a message next.

Chapter 1

INTRODUCTION

Parallel computers with hundreds or even thousands of processors are considered the most promising technology to achieve high performance. For example, Intel is building the “ultra” computer, which will contain 9,200 Pentium Pro Processors [68]. Nodes in a multiprocessor machine are interconnected through a direct network (Figure 1). In multiprocessor machine, each node has its own processor, local memory and routing device. Each node has a direct connection to some number of other nodes. Figure 2 shows the architecture of a generic node. Distributed multiprocessor nodes do not physically share memory. They must communicate by passing messages through the network. Neighboring nodes may send messages to one another directly, while nodes that are not directly connected must depend on other nodes to send messages from source to destination. In many systems each node contains a separate router to handle communication-related tasks.

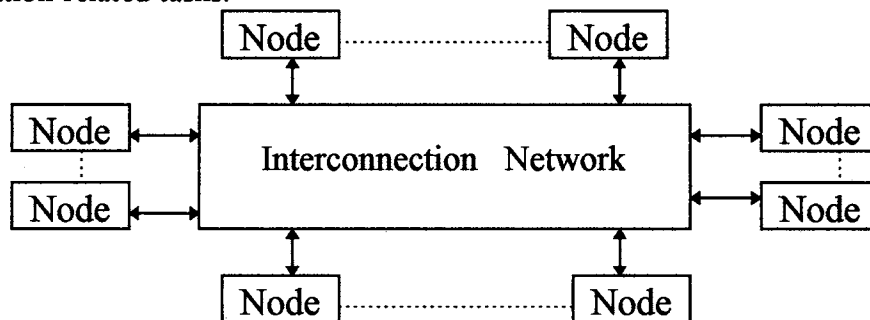


Figure 1. A generic multiprocessor machine connected by interconnection network.

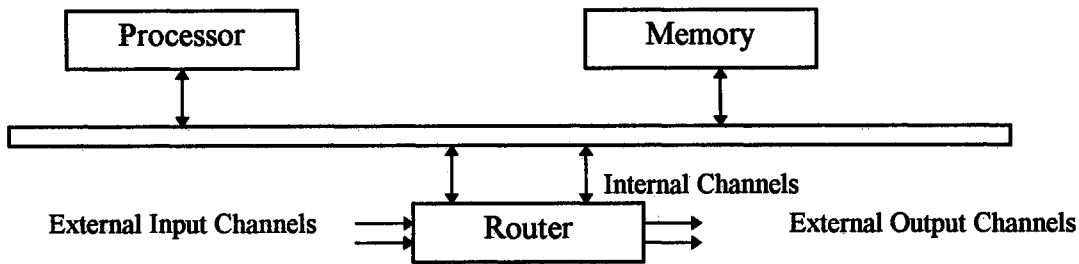


Figure 2. A generic node architecture.

Parallel architectures rely on fast inter-processor communication to exploit concurrence in computational tasks. Low message latency and high network throughput are necessary to exploit parallelism. Massively Parallel Processors (MPP) and Networks of Workstations (NOW) critically depend on internal communication performance[69]. Interconnection networks are used to pass messages between the nodes of concurrent computers. However, as networks get larger, the probability of component failure increases as well. We would like large networks to continue to operate correctly in the presence of hardware failures. The ability of a network to reliably deliver messages in the presence of network component failures has become an important topic.

Wormhole routing has emerged as the dominant communication mechanism in special purpose and commercial high performance multicomputer interconnection networks. It is being used in several multicomputers such as Cray T3D [25], The MIT J-machine[20], Intel Paragon[26], nCUBE[27], iWarp[28], and IBM Power Parallel Series[29]. Moreover, Wormhole routing is used in high performance LANs such as Myrinet[70] and TNet [71]. Wormhole routing is a pipeline mechanism where data flits (defined in chapter 2) follow the header flit. The path is torn down by the tail flit. Also

some parallel machines (Cray T3D [25] for example) use the concept of virtual channels where multiple virtual links are multiplexed across a physical link.

Failures can occur on busy links or nodes and interrupt message transmission. Since only header flit contains routing information, data flits whose progress is blocked by a failure cannot progress. Moreover, the data flits who were in the failed node need to be recovered. Although the message interruption due to faults are considered rare, the current data recovery protocols impose overhead with every message. This dissertation proposes a new recovery protocol in which overhead is attached only with the corrupted messages. This dissertation also evaluates the performance of the current recovery protocols and the proposed protocol. The hardware and software requirements to get robust and fast router have been discussed.

This chapter will define the contribution of this thesis with respect to contributions made by others. This chapter contains many terms that are defined in chapter 2.

1.1 Related Work

W. Dally et al[1] propose “Unique Token Protocol” (UTP) to handle dynamic faults. This protocol depends on a second copy of every flit kept in preceding node and the head flit is stored in every node spanned by the packet. Also, at the end of each message there is a token. When a fault happens in the middle of a message the node preceding the faulty node constructs a new message using its own copy of the message’s header. After the fault, the token becomes two tokens as a flag to distinguish between the original message and the new message. This protocol reduces the buffer requirement in the sender side and does not require retransmission of interrupted messages. On the other

hand, UTP increases the buffer requirement in the routers by requiring the head of each message to be stored in each spanning node. Also there are two copies of each flit in the path which means the length of each message has been doubled. This protocol is not robust enough to handle multiple faults that occur in one message pipeline.

The main idea proposed in [2,3] is to use re-transmission mechanism to tolerate dynamic faults. When a fault is detected, the detecting routers send kill signals both forward and backward along the message path. These kill signals follow the virtual circuits back to the source and destination and release reserved buffers and notify the source that the message was not delivered and the destination to ignore the message currently being received. The protocol proposed by Kim et al[2] requires padding extra flits at the end of each message. Although the protocol proposed by Gaughan et al[3] does not require these extra flits, it requires an acknowledge signal[9]. This mechanism can handle more than one fault. But re-transmitting will increase message latency and will decrease the throughput. Control lines required by this mechanism make the hardware cost high and complicate router's design.

Recent advances in message passing techniques show substantial improvement in latency (see Table 1). In this research we propose a new software based recovery protocol (SBRP). SBRP depends on control messages instead of control lines. By sending control message to recover from faults when it happens we eliminate the need for penalizing every message. The basic concept is when a fault happens a control message is sent to the sender to inform it that a message have been corrupted. There is a control message for each corrupted message. This control message is send either by the node that precedes the faulty node in the message pipeline, by a node after the faulty node in the message

pipeline or by the receiver. Once the sender gets this message it sends the remainder of the corrupted message as a new message. The main advantage of the software protocol is that the recovery overhead is attached only with corrupted messages.

1.2 Thesis Outline

In this dissertation, our focus is recovery protocols in wormhole routing networks. We evaluate the current recovery protocols and explain their advantages and disadvantages. We also propose a new recovery protocol to overcome the shortcoming of current protocols. The requirements of the current protocols and the proposed one are also explained thoroughly. Extensive simulation study have been conducted to evaluate all protocols.

Chapter 2 illustrates the different aspects of interconnection networks such as topologies, switching techniques, virtual channels, messaging layers and performance measurements.

Chapter 3 explains the routing algorithms and the relation between the routing algorithm and fault tolerance. The current recovery protocols are explained and discussed in Chapter 4. The proposed protocol is illustrated in chapter 5. Chapter 6 reports the performance evaluation and explains the simulator used in this research. The conclusions are presented in chapter 7 together with recommendations and suggestions for further research.

Chapter 2

INTERCONNECTION NETWORKS

A multiprocessor architecture can be described as a set of processing elements, PE's, and an interconnection network. An interconnection network is used to logically connect the processors. Processors communicate between them by exchanging data through an interconnection network. Thus, interconnection network plays a major role in the performance of modern parallel computers. Interconnection networks have been classified according to the network type (distributed or shared memory) and network topology (hypercube or mesh).

2.1 Distributed vs. Shared Memory

There are two possibilities when designing a multiprocessor. A multiprocessor can have either shared memory or distributed memory. Figure 3 shows a simple shared memory architecture. There are k global memory modules, denoted GM. There are also n processing elements, PE, where each PE has a local memory and a processor. The PE is connected to the interconnection network through the network interface, NI. The interconnecting network allows each processing element access to the global memory. Figure 3 shows a simple distributed memory multiprocessor. The PE's perform computations, send messages and receive messages. The interconnection network routes

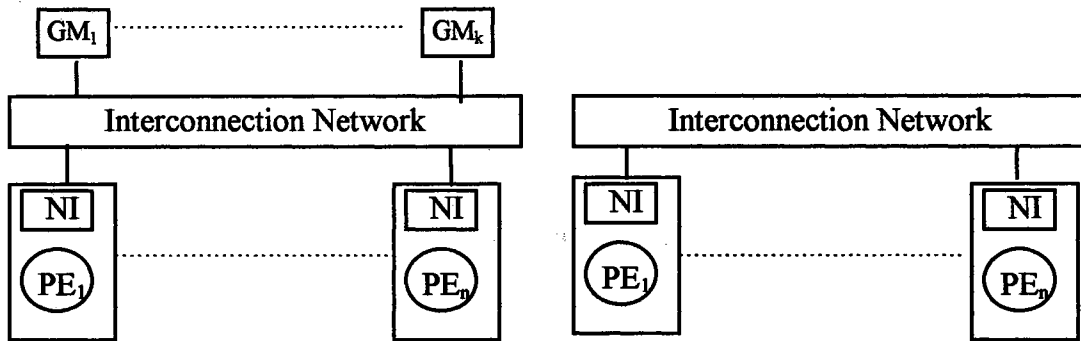


Figure 3 Shard Memory and Distributed Memory

messages between PE's and handle necessary allocation of network resources to transmit messages from source to destination.

The programming model can use either a shared-memory paradigm or a message passing paradigm. The shared memory paradigm is generally easier from a programmer standpoint because all the data are assumed to be available in the shared memory. However, building scaleable shared-memory multiprocessor is difficult, because access to the global memory modules is a bottleneck. Explicit synchronization is required to prevent processors from reading results before they are written and from overwriting results before they are read. Synchronization is often a costly process. As the number of processors increases, the minimum network latency increases.

Distributed-memory multiprocessors are considered more scaleable and they communicate faster. Therefore, most existing large-scale multiprocessor computers use distributed memory. The message latency on a distributed-memory multiprocessor is partially dependent on the distance between the processors, where the distance is measured as the number of network channels that must be traversed to reach the destination. If the program is partitioned in a good manner, a large part of messages can be eliminated by communicating with neighboring processors.

The main drawback of distributed-memory multiprocessors is the difficulty of writing efficient parallel programs. In order to support communication on a distributed-memory multiprocessor, the message passing model requires the program code to designate both the sending and receiving processor for each message. This increases the complexity of programming on distributed-memory multiprocessors. Distributed Shared Memory (DSM) has been proposed to address this disadvantage. In this programming paradigm, the memory is physically distributed but logically shared. The programmer can assume a global address space. The operating system converts memory request for non-local data into messages. Although distributed-shared memory simplifies the task of writing parallel programs, the issue of efficient communication remains.

Our research has focused on distributed-memory multiprocessor because it has the most efficient communication scheme. Also most of the large-scale multiprocessors use distributed memory.

2.2 Network topologies

The topology of a network is simply the node interconnection pattern. It is generally modeled as a graph where the vertices represent the nodes and the edges denote the channels. A network topology with more channels can reduce the diameter and average distance. However, completely connected network is not practical. Fault-tolerant networks must be multi-path networks. Each pair of PE's must be connected by more than one physical path through the network.

There are many ways to interconnect nodes in a direct network. The most popular topologies are Hypercubes and n-dimensional meshes

2.2.1 Hypercube

A hypercube consists of $n=2^d$ nodes numbered 0 to $n-1$ linked together in d -dimensional cube network. Each node is assigned a d -bit binary representation $u_{d-1} \dots u_1 u_0$. The relative address of two nodes a and b is the bitwise exclusive-OR, \otimes , of their binary representations. The Hamming distance, $H(a, b)$, between two nodes a and b is the number of ones in $a \otimes b$. Two nodes a and b in a hypercube share a communication link if and only if $H(a, b)=1$. The nodes of a d -dimensional hypercube correspond to all binary strings of length d . The d neighbors of processor i are those processors j such that the binary representation of the numbers i and j differs by exactly one bit. Figures 4 and 5 show a hypercube with $n = 2^3 = 8$ nodes and a hypercube with $n = 2^4 = 16$ nodes, respectively.

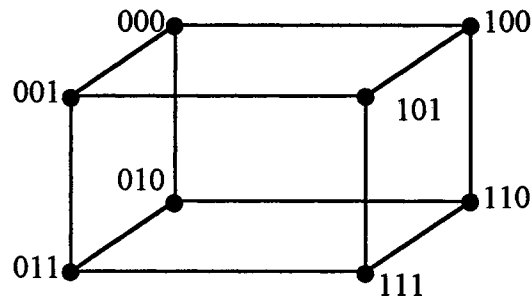


Figure 4 8-nodes hypercube.

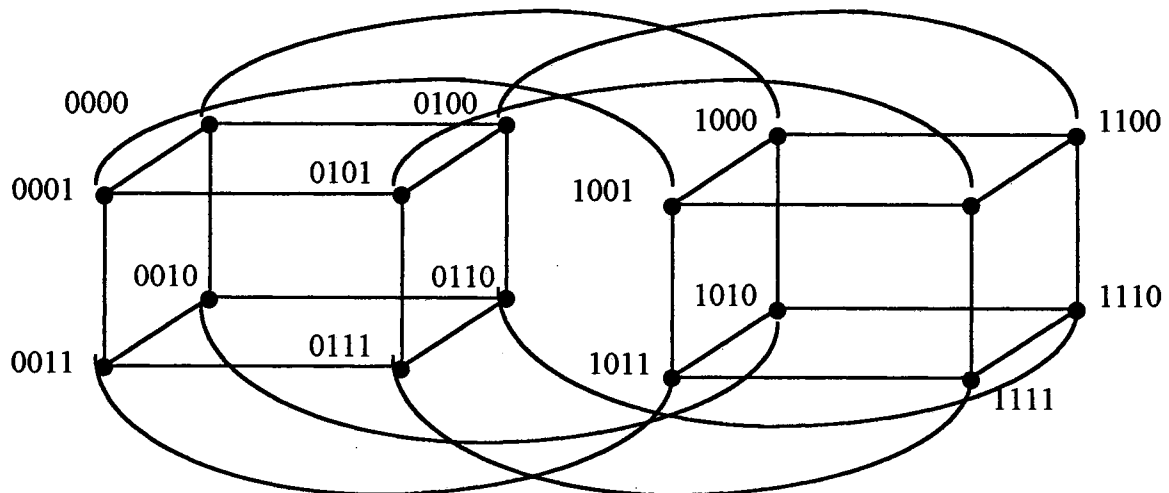


Figure 5 16-nodes hypercube.

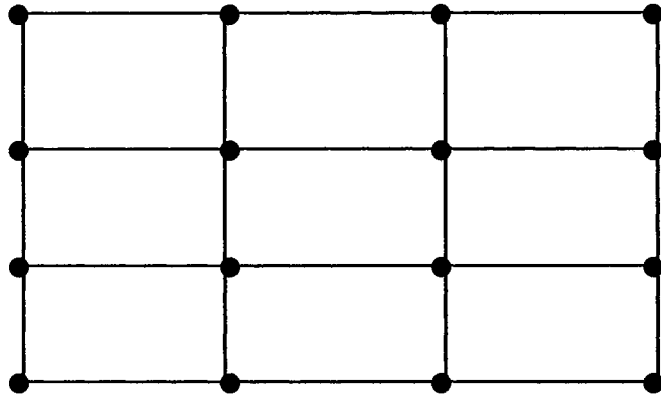


Figure 6 4 × 4 2-dimensional Mesh.

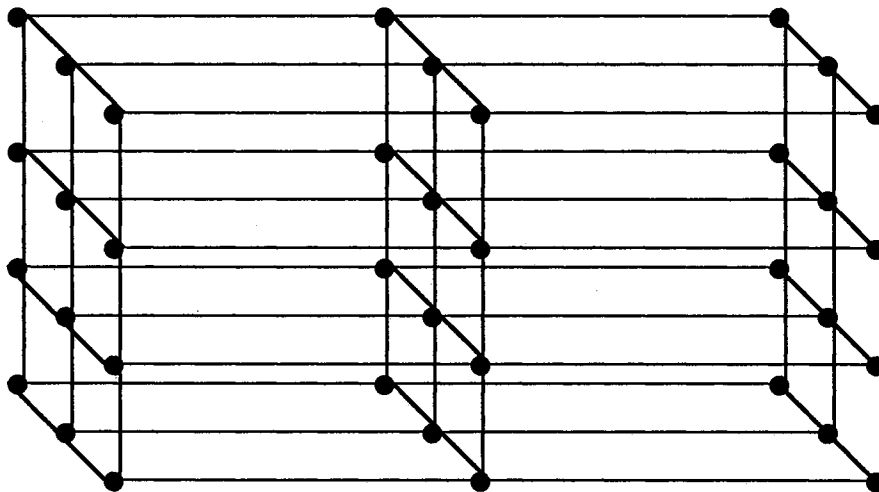


Figure 7 3×3×3 3-dimensional Mesh

2.2.2 Mesh

The d -dimensional mesh has $k_0 \times k_1 \dots \times k_{d-2} \times k_{d-1} = N$ nodes, k_i nodes along each dimension i . Each node q is identified by n coordinates $(x_0, x_1, \dots, x_{n-2}, x_{n-1})$ where $0 < x_i < k_i$. Two nodes a and b are neighbors if and only if $x_i(a) = x_i(b)$ for all i , $0 < i < d$, except one, j , where $x_j(a) = x_j(b) \pm 1$. Figure 6 shows a 2-dimensional mesh and Figure 7 shows 3 dimensional mesh.

2.3 Switching Techniques

Communication between nodes in a direct network is performed by passing messages from one node to another. A message may be divided into one or more equal or variable size packets for transmission. A packet is the smallest unit of information that contains routing and sequencing information. Since it is not feasible to provide a channel between every pair of nodes, the channels are shared among the nodes. For some non-adjacent source-destination pairs to communicate, messages must traverse intermediate nodes along the path from the source to the destination.

The switching techniques determine when and how internal switches are set to connect router inputs to outputs, and the time at which message components may be transferred along these paths. These techniques are coupled with flow control mechanisms for the synchronized transfer of units of information between routers, and through routers in forwarding messages through the network. Flow control is tightly coupled with buffer management algorithms that determine how message buffers are requested and released, and as a result determine how messages are handled when blocked in the network.

The architecture of a general router is shown in Figure 8 and it consists of the following major components:

- **Buffers:** These are first-in-first-out (FIFO) buffers for storing messages in transit. In the above model, a buffer is associated with each input virtual channel and each output virtual channel (The concept of virtual channel will be explained later). In alternative designs, buffers might be associated only with inputs or outputs.
- **Switch:** This component is responsible for connecting router input buffers to router output buffers.

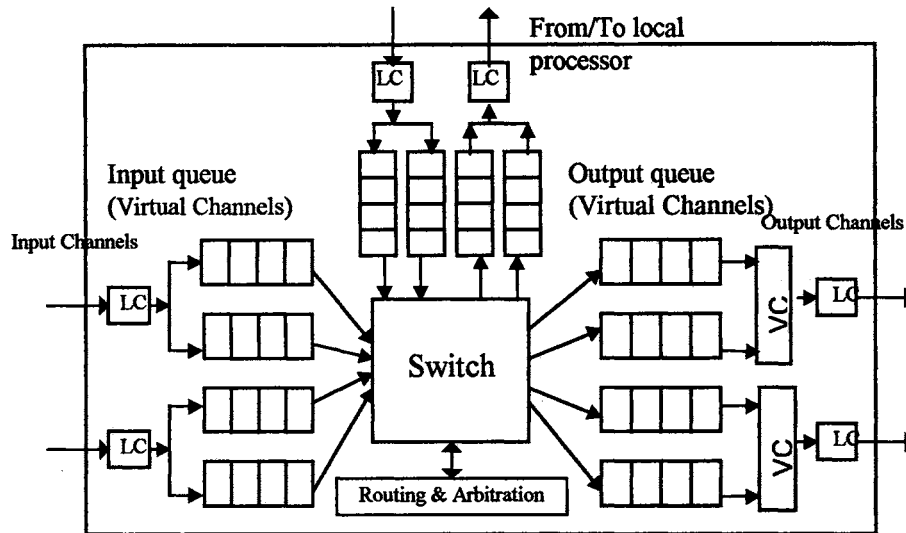


Figure 8 Router Model (adopted from [15])

- **Routing and Arbitration Unit:** This component implements the routing algorithms, selects the output link for an incoming message, and accordingly sets the switch. If multiple messages simultaneously request the same physical output link, this component must assign for every message a virtual channel if any available. If all virtual channels in the requested physical channel are busy, the incoming message remains in the input buffers or routed to other physical channel (depending on the routing algorithm as will be explained in chapter 3).
- **Link Controllers (LC):** The flow of messages across the physical channel between adjacent routes is implemented by the link controller. The link controller on either side of a channel coordinate to transfer units of flow control.
- **Virtual Channel Controller (VC):** This component is responsible for multiplexing the virtual channels over the physical channel.
- **Processor Interface:** This component simply implements a physical channel interface to the processor. It consist of one or more injection channels from the processor and one or more ejection(delivery) channels to the processor.

When a message first arrives at a router, it must be examined to determine the output channel over which the message is to be forwarded (*routing delay*). The time which is required to forward the message through the switch is called *switch delay*. This time include the propagation delay through the switch (intra-router delay) and the signaling rate for synchronizing the transfer of data between the input and output buffers.

The hardware protocol to transfer a flit from node A to node B is illustrated in Figure 9. The *data* lines are validated by the hand shaking signals *request* and *acknowledge*. The acknowledge line is high when the receiving end is ready to take in another flit. The request line is high when sending end has a data out in the data lines. When the receiving end is ready and detects a high in the request line, it will lower the acknowledge and simultaneously latch the data lines as shown in Figure 10.

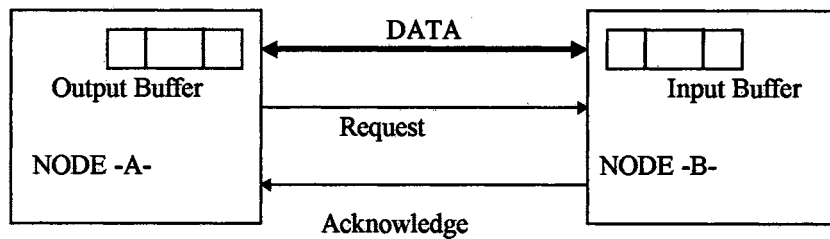


Figure 9 The physical connection of a uni-direction link

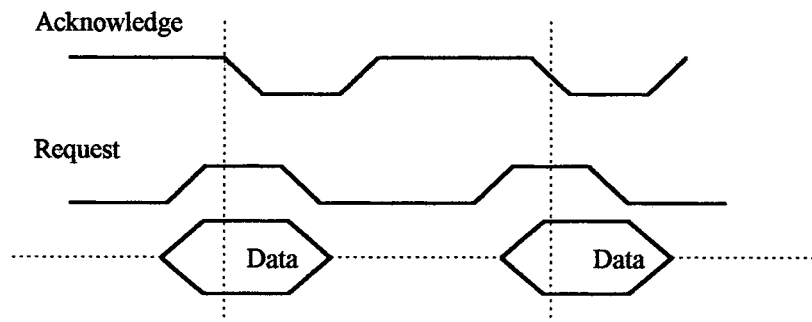


Figure 10 The Handshaking Protocol

The switching technique defines how messages are propagated through the network. There are three main switching techniques circuit switching, packet switching and wormhole switching. To derive a lower bound expression for message latency, t , we denote L for message length, L_h for head length, L_a for acknowledgment length, h for number of hops, W for channel width, t_r for routing time, and t_s for switching time. The packet size and flit size are assumed to be equivalent and equal to the physical data channel width. Also, the router's internal data paths are assumed to be matched to the channel width.

2.3.1 Circuit Switching

In circuit switching, a dedicated path is established between the source and the destination before the initiation of data transfer. Once the data transfer is initiated, there is no blocking of the message. Circuit switching first sends a probe (head) from the source to the destination and returns an acknowledgment once the path is established. If the probe encounters a busy channel, the partial path that has been established is aborted and the probe is retried later.

There are a few advantages to circuit switching. For long messages, the message latency is almost independent of the distance between the source and destination. In addition, there are no storage requirements at the routers. The minimum latency for a message that travels two hops through the network is shown in Figure 11. The formula for the minimum message latency, t , is:

$$\begin{aligned}
 t_{\text{circuit switch}} &= t_{\text{setup}} + t_{\text{data}} \\
 t_{\text{setup}} &= h * (t_r + t_s + L_h/W + t_s + L_a/W) \\
 t_{\text{data}} &= L/W
 \end{aligned}$$

The main disadvantages of circuit switching is the significant overhead needed to establish the path. Since the head is transmitted to the destination and acknowledged before any data is sent, the time to establish the path can dominate the time required to send the message, especially when the diameter of the network is high and the messages are short. Moreover, a circuit requires exclusive use of all the channels on the path so many headers can be blocked by a single circuit. Finally, having to retry blocked headers can waste network bandwidth and reduce the network throughput.

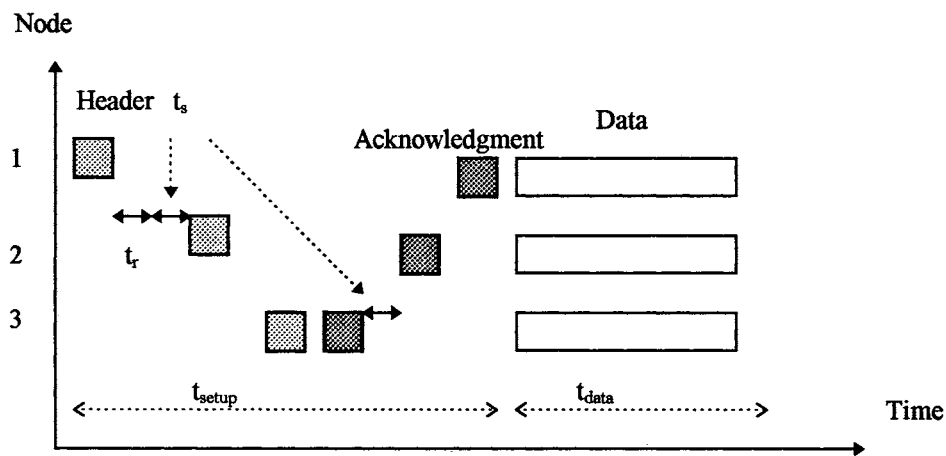


Figure 11 Circuit Switching (adopted from [36])

2.3.2 Store-and-forward (Packet) Switching

As opposed to circuit switching, store-and-forward routing sends the message before the entire path has been established. Store-and-forward routing is implemented via packet switching or message switching. Packet switching divides each message into packets and sends each packet individually, while message switching sends the entire message as a single unit. The network treats each packet as a separate message, so from the perspective of the network, the only difference between the two is that packet switching has messages of fixed length. For this reason, we use the terms message and

packet interchangeably. Packet switching transmits the packet header and all the packet data to a neighboring node. The entire packet is then stored in a buffer at this node and later forwarded to the next node in the path.

The entire packet is buffered at each intermediate node before any part of the packet is forwarded. Therefore, the message latency is the product of the packet length and the distance between the source and the destination. This is acceptable for small packets on networks with relatively small diameter, however, the message latency for large packets or for packets in large diameter networks can be unacceptable. The minimum latency for a message that travels three hops through the network is shown in Figure 12. The formula for the minimum message latency in store-and-forward switching,

$t_{\text{store-and-forward}}$, is:

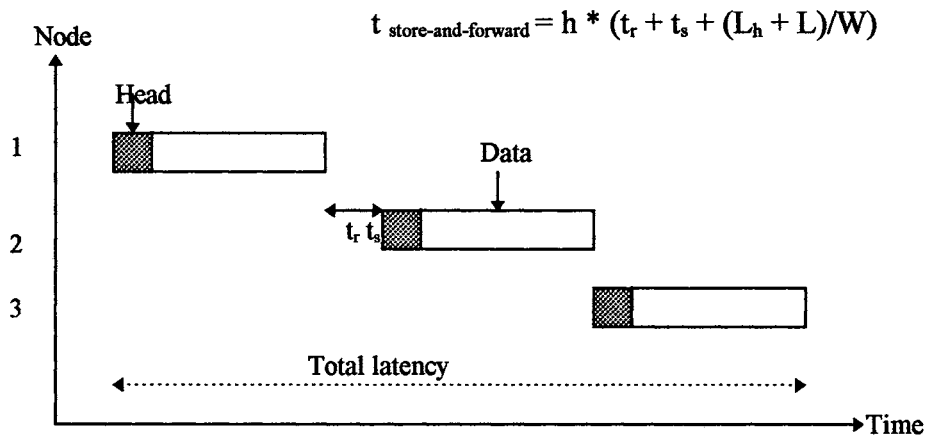


Figure 12 Store-and-Forward (Adopted from [36])

The packets are usually too large to be placed on the router, so the packet is transferred to processing node and stored in system memory at each intermediate router. This increases the message latency, because the packet incurs setup overhead at each node. Furthermore, each intermediate node has to provide system memory to store the packets.

2.3.3. Wormhole Routing

As we have mentioned before the wormhole switching technique has been widely used in the recent multicomputers. In wormhole routing, a packet is divided into sequences of flits. A flit is the smallest unit of a packet on which flow control can be performed. The length of a flit corresponds to the width of a network channel. A packet consists of one or more header flits, the data flits and a tail flit to mark the end of the packet. Wormhole routing operates, as Figure 15 illustrates, by advancing the head of a packet directly from incoming to outgoing channel. The head is the only flit that has routing information. As flits are forwarded the message becomes spread out across the channels between the source and destination. It is possible for the first flit to arrive at the destination node before the last flit of the message has left the source. Because most flits contain no routing information, the flits in a message must remain in contiguous channels of the network and cannot be interleaved with flits of other messages. A packet holds a channel from the time the header acquires the channel until the tail releases the channel. When the header flit of a message is blocked, all of the flits of a message stop advancing and block the progress of any other message requiring the channels they occupy[7]. The primary drawback of wormhole routing is the contention that can occur with even moderate network traffic, which causes higher message latency. A packet that uses several channels can block many messages while being transmitted. These blocked messages can in turn block other messages, which further increases message latency and reduces the throughput. To solve this problem the concept of virtual channel have been proposed. Multiple virtual channels share a physical channel. Each virtual channel has a separate buffer, with multiple messages multiplexed over a physical channel. Virtual channels

reduce contention because more channels are available and messages can pass the blocked messages. Virtual channel is explained in detail in the next section. The formula for the minimum message latency in wormhole switching(as shown in Figure 13), $t_{\text{Wormhole Switching}}$,

$$t_{\text{Wormhole Switching}} = h * (t_r + t_s + t_w) + \max(t_s, t_w) * L/W$$

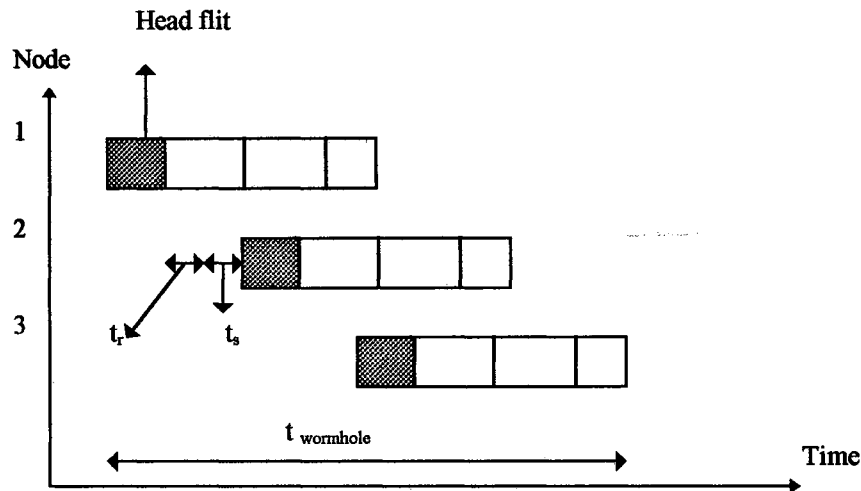


Figure 13 Wormhole Routing Switching (Adopted from [36])

Due to channel width constraints, multiple physical channel cycles may be used to transfer a single flit. A phit is the unit of information that can be transferred across a physical channel in a single cycle. Flits represent logical units of information but phits represent the physical units. The relationship between the size of phits, flits and packets differs across machines. Many machines have the phit size equivalent to the flit size as in the IBM SP2 [37]. Alternatively, in the Cray T3D [25] each flit consists of eight 16 bit phits.

The format of a wormhole packet in the Cray T3D is shown in Figure 14. In Cray T3D, a phit is 16 bits and a flit consists of eight phits. A word is 64 bits and thus four phits. A message consists of header phits and data phits. The header information may include destination address, source address, sequence number, control information, message length and process id.

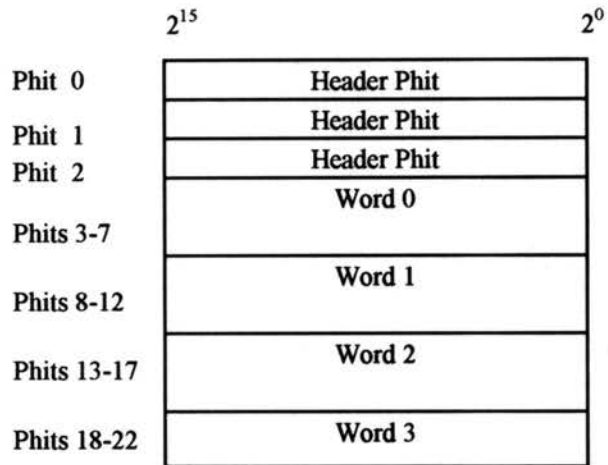


Figure 14 Format of wormhole packet in the CRAY T3D

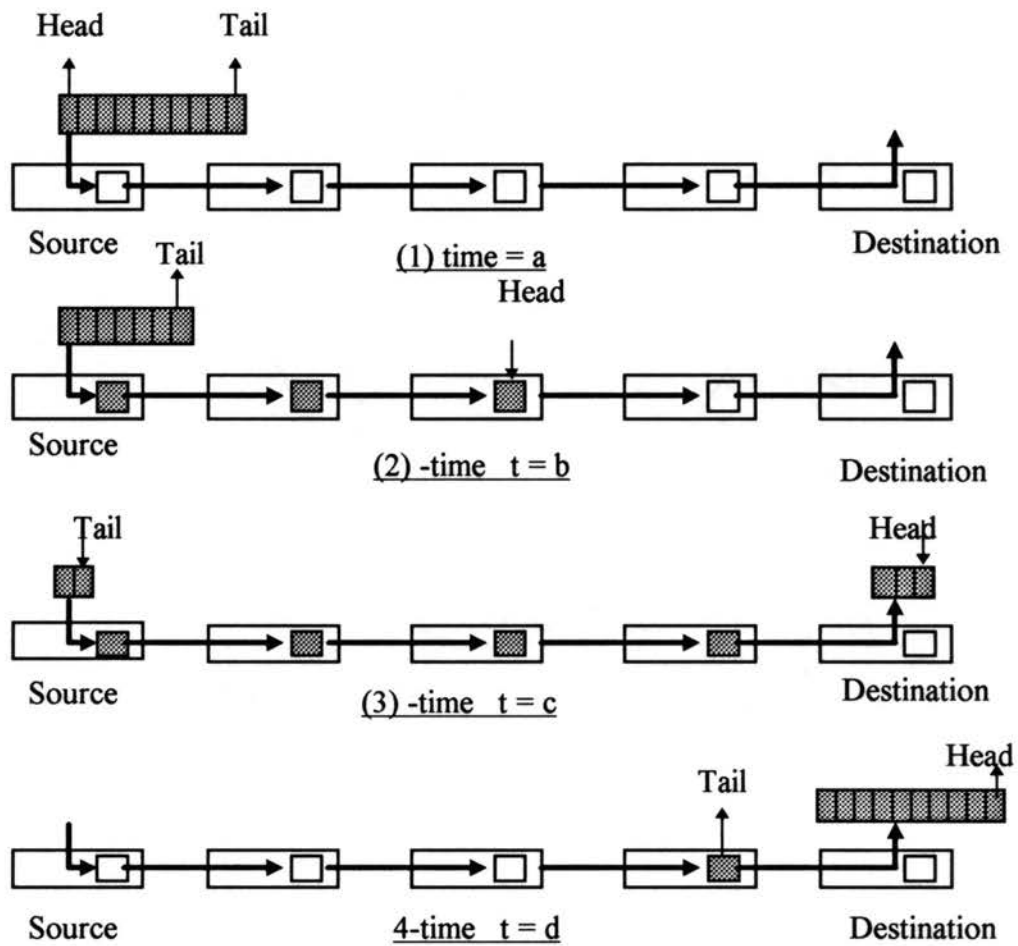


Figure 15 Wormhole Routing.

2.4 Virtual Channels

A virtual channel consists of a buffer that can hold one or more flits and associated information. Several virtual channels may share the bandwidth of a single physical channel. Any arbitrary algorithm can be used to allocate physical channel bandwidth among virtual channels including random, round-robin, or priority. Adding virtual channel flow control to a network makes more effective use of both channels' bandwidth and memory buffers by decoupling their allocation. The only expense is a small amount of additional control logic[7].

The virtual channel strategy allocates buffers and channel bandwidth to flits. Because flits have no routing information, the allocation must be done in a manner that keeps the flits associated with a particular packet together. This may be done by associating a set of buffers and some control state (implemented by hardware) together into a virtual channel.

A network using virtual channel flow control organizes the flit buffers associated with each channel into several lanes as in Figure (16). The control lines decide which virtual channel uses the physical channel at any point of time. The buffer in each lane can be allocated independently of the buffers at any other lanes. This added allocation flexibility increases channel utilization and thus throughput.

Adding virtual channels to an interconnection network is analogous to adding lanes to a street. A network without virtual channel is composed of one lane streets. Adding virtual channels to the network adds lanes to the street allowing blocked packets to be passed. A blocked message, even one that extends through several nodes, holds only single lane idle and can be passed using any of remaining lanes as in Figure (17-b).

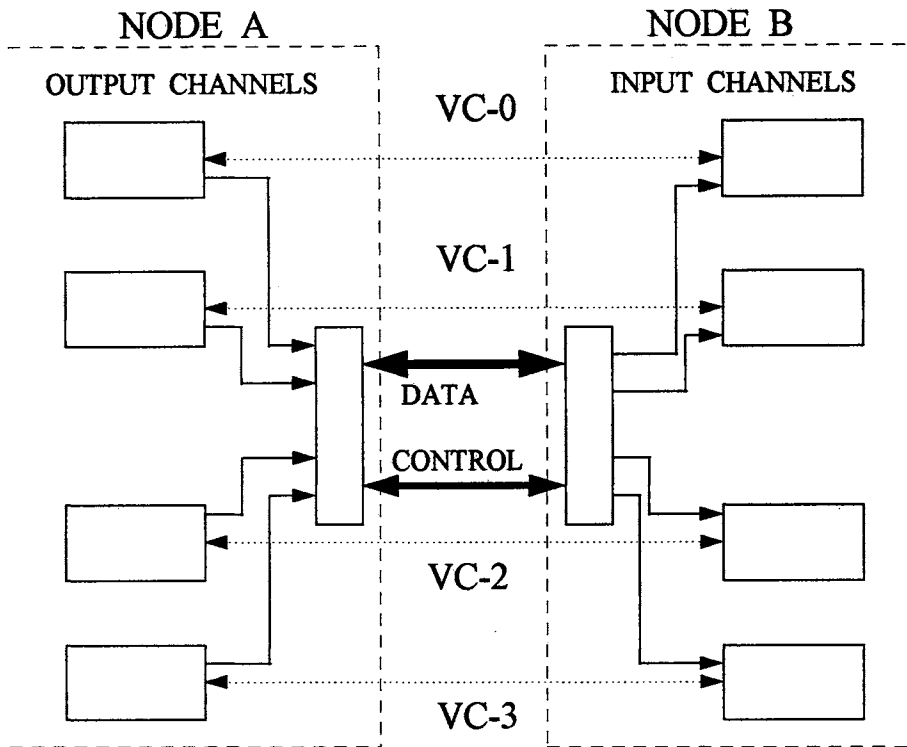


Figure 16 Four virtual channels share a physical channel

To preserve bandwidth, only those virtual channels that have a nonempty flit buffer at the sender side and a nonfull flit buffer at the receiver side may participate in the scheduling decision.

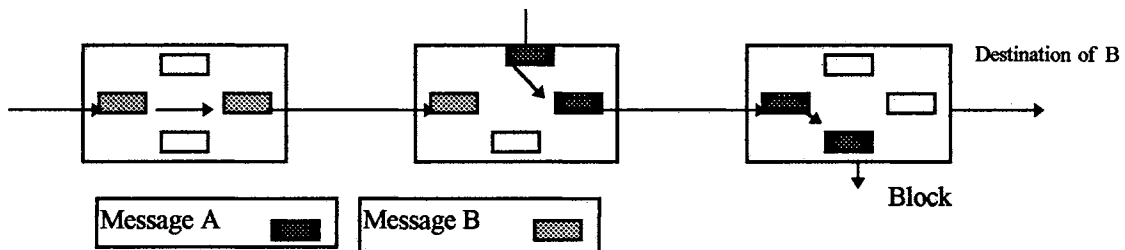


Figure 17-a Message B is blocked behind message A while all physical channels remain idle

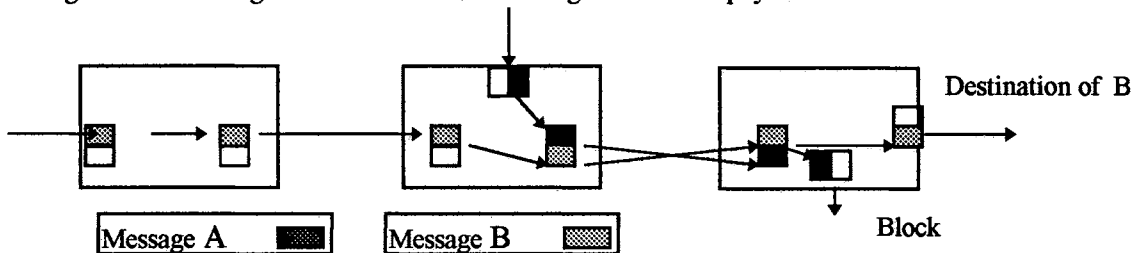


Figure 17-b Virtual channels provide additional buffers allowing message B to pass blocked message A

2.5 Message Passing

Applications on parallel machines rely on inter-node communication to transfer data, synchronize and coordinate the parallel computation. Messaging layers on parallel machine provide communication services to application, implementing in software whatever is not provided directly by the underlying hardware. Typical features include reliable and in-order delivery of messages with flow control. Messaging layers must provide high bandwidth with low latency and overhead, and provide these services robustly.

The major elements of message passing latency are: 1) software: interrupt serving, system calls, buffering and copying, 2) network interface delays: message injection, message ejection and hardware flow control, and 3) network time: hardware latency due to congestion, routing and data transmission[31].

The function of the messaging layer is to deliver messages between nodes. An example of the flow of data from the location in the source node's memory to the destination node's memory location is shown in Figure 18. Let us suppose that a user process wants to send a message to another process on another node. Transmission of this message is initiated via a call to a message passing procedure such as *send(buf, nwords, dest)*, where *buf* contains the *nwords* to be transmitted to node *dest*. A message packet must be created (packetization) with a header containing information required to correctly route the packet to its destination. Access to the network interface at the sender may not be available immediately. Therefore, the message packet may be buffered in system memory, copying and buffering, prior to injection into the network while the *send()* call returns to the main program. Traditionally, drivers that control messaging are available

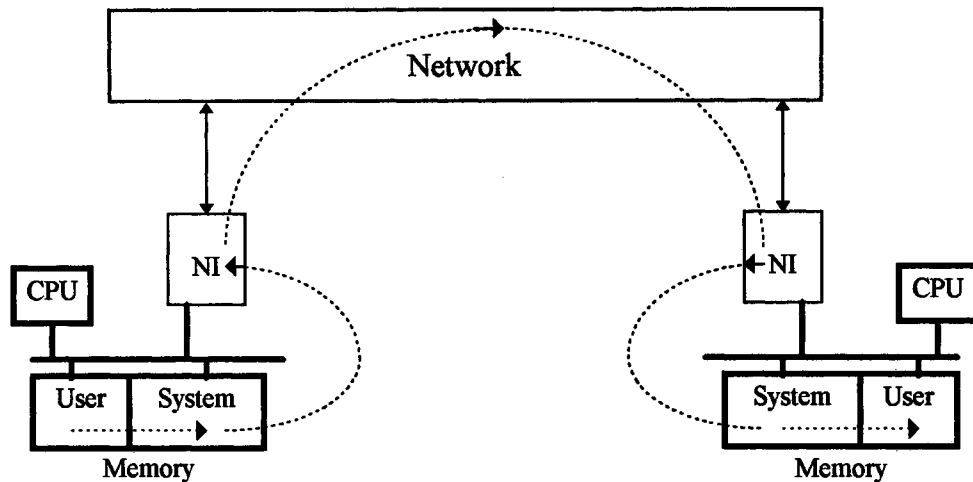


Figure 18 Message transmission and reception

through system calls (User/Kernel Transitions). Once the network interface has the message packet, it is injected into the network, where the routers cooperate in delivering the message to destination node interface. When the message is received at the node, there must be some way to invoke the messaging layer software (interrupts or polled access). Similar services are then invoked to transfer the message from the network interface into system buffers and then to user buffers.

Early message layer implementations had the message handlers execute in the kernel causing expensive context swaps on each call. Messages were first read from the network into system buffers and then copied into user memory data structures. In the new implementations, the message handler executes in the user context and have direct access to the network interface from the user level. In addition to the substantial saving in buffering and copying, the high overhead of frequent context switching is avoided.

Traditional message passing mechanisms incur huge overhead by going through many operating systems layers. Also, poor hardware interface between processing

elements and communication network leads to inadequate performance. However, recent advances in messaging implementation and improved network interface have reduced the software cost of messaging significantly. One of the recent message handling mechanisms is the Active Mechanism[17,16]. Active Mechanism eliminates unnecessary overhead between the processor and the network. In an active-messages interface, data transfers are coupled with invoking a handler at the destination[17].Active Mechanism allows the user to specify the address of the handler to be invoked upon message arrival. Active messages eliminates the need to buffer data at both the source and destination.

Table 1 shows the great enhancement achieved by using such new mechanism compared to old mechanisms.

Machine	T_s	T_b	Cycles/msg	Cycles/byte
nCUBE/2	160.0	0.45	3200	9
CM-5	86.0	0.12	2838	4
DELTA	72.0	0.08	2880	3
nCUBE/2	23.0	0.45	460	9
CM-5	3.3	0.12	109	4
J-Machine	0.9	0.04	11	0.5

Table 1 One-way message overhead. T_s is the sum of the fixed overheads of send and receive. T_b is the injection overhead per byte. The data is adopted from [20].

The network interface, NI, is responsible for injecting and receiving messages from the network. Packets are injected into the network by storing the destination node number and data arguments to the NI send buffer. Messaging costs are strongly influenced by a machine's network interface architecture. Processor involvement for message reception increases communication overheads. In contrast, decoupling message reception from

processor activity produces high performance messaging. Therefore, some implementations, Cray T3D for example, dedicate hardware for messages' operation, decoupling data transfer from local computation[32]. The Paragon dedicates a general-purpose processor to communications and uses an identical chip as a main processor[34].

Many multiprocessor machines use the one phase (non-blocking) protocol for messaging especially for short messages. The sending processor specifies the destination processor, transmits message directly to the interconnection network

Figure 19 shows the four steps in two phase messaging. The source first sends a buffer allocation message to the destination (step 1). On receiving a buffer allocation message, the destination allocates a buffer of the appropriate size and responds with the segment identifier (step 2). The source then initiates the transfer by splitting the message into several packets, each of which carries the destination segment identifier (step 3). The destination receives the data packets, reassembles them into the allocated buffer and invokes the specified user handler on transfer completion(step 4).

Several factors cause message reception to be more expensive than sending. Sending is synchronous to the computation whereas reception is asynchronous. Reception involves a dispatch, while sending does not[33].

There are many implementations to send or receive messages. As a typical example, we examine the J-Machine. On the J-Machine, a series of SEND instructions is used to inject messages at a rate up to two 32-bit words per cycle. Each SEND appends one or two words to the message currently being composed. The first word of the message contains the destination address, the length of the message and the address of the

code to run at the destination. A bit in the SEND instruction indicates the end of the message and completes its injection into the network[33].

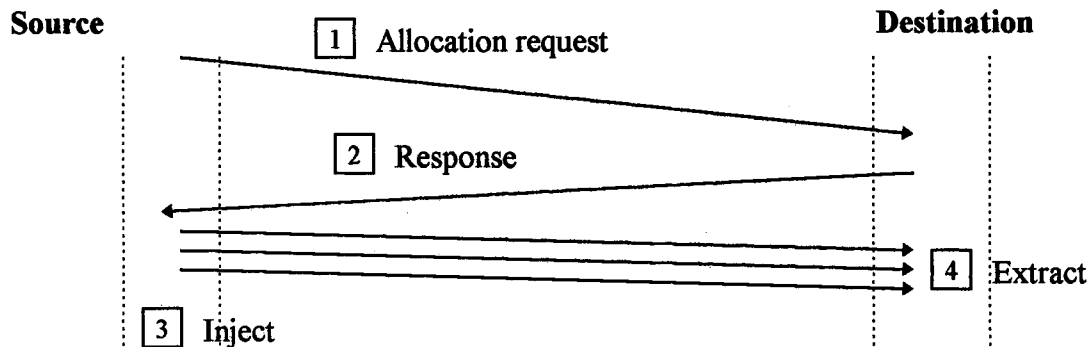


Figure 19 Two Phase Messaging Implementation

The J-machine implements asynchronous message reception by directly storing messages into an on-chip queue and dispatching to the code indicated by the first word of the message at the head of the queue[33]. A task is dispatched to handle the message in four processor cycles. During these cycles the instruction pointer, IP, is loaded from the message header and an address register is set to point to the new message[20]. Synchronization is provided by the ability to signal events effectively using low-latency primitives.

Chapter 3

ROUTING ALGORITHMS AND FAULT TOLERANCE

The routing layer of the interconnection network implements the routing protocol used to route messages from source to destination. The routing protocol consists of the routing function and the selection function. The routing function provides a set of available candidate output channels. The selection function chooses one of the candidate channels as the outgoing channel for the message.

A routing algorithm should be able to route packets from any source node to any destination node. Also, the routing algorithm should be able to guarantee that packets will not block or wander across the network forever (deadlock and livelock freedom). The ability to route packets through alternative paths in the presence of contention or faults is one of the most important properties of routing algorithms.

Routing algorithms can be minimal, non-minimal, progressive, backtracking, deterministic, partially adaptive, or fully adaptive[10]. A routing algorithm is said to be minimal if the path selected is one of the shortest paths between the source and destination pair. A nonminimal routing algorithm allows packets to follow a longer path, usually in response to current network conditions.

Routing algorithms can be classified as deterministic or adaptive. Deterministic routing algorithms always supply the same path between a given source/destination pair. Adaptive routing algorithms use information about network traffic and channel status to avoid congested or faulty regions. Adaptive routing algorithms can be classified according to their progressiveness as progressive or backtracking. Progressive routing algorithms move the header forward, reserving a new channel at each routing operation. Backtracking algorithms allow the header to backtrack, releasing previously reserved channels. Backtracking algorithms are mainly used for fault-tolerant routing and they are explained in detail in [38].

3.1 Deadlock, Livelock and Starvation

In direct networks, packets usually travel across several intermediate nodes before reaching the destination. However it may happen that some packets are not able to reach their destination, even though there are fault free paths connecting the source and destination nodes for every packet. A packet may be traveling around its destination node and never reach it because the channels required to do so are occupied by other packets. This situation is known as livelock. It can only occur when packets are allowed to follow non-minimal paths. Thus, to avoid livelock the minimal paths only should be used. However, using non-minimal paths some times is a must to achieve fault tolerance or to get adaptiveness. Livelock can be prevented even with non-minimal paths being used by limiting the number of misrouting operations.

Starvation might happen for a packet if its requests to the resources(usually channels or buffers) are always denied and the resources granted to other packets

requesting them. Starvation occurs when an incorrect resource assignment scheme is used to arbitrate in case of conflict. Starvation problem can be solved by using a fair assignment scheme. A simple round robin scheme is enough to produce a fair use of resources. When some packets must have a higher priority, some bandwidth must be reserved to serve the low priority packets.

Deadlock is a situation that occurs when a set of messages get blocked forever in the network. In such a situation, the packets hold certain resources and request for other resources that are held by other messages involved in the deadlock configuration. Figure 20 shows a deadlock scenario in a two-dimensional mesh. Four messages are being routed from source S1, S2, S3, and S4 to destinations D1, D2, D3, and D4, respectively. In the illustrated figure, all the messages are waiting for a channel that will never be available, thus resulting in a deadlock situation.

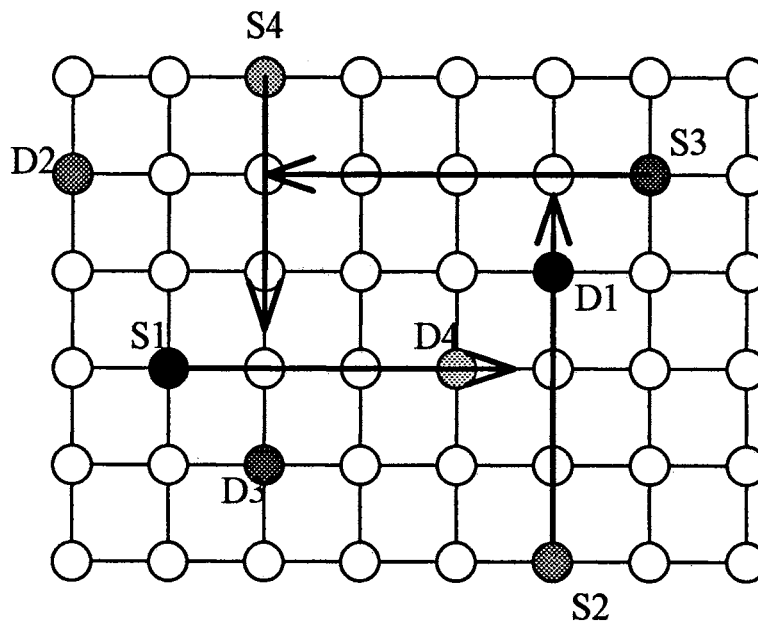


Figure 20 A deadlock situation involving four messages

There are three strategies for deadlock handling: deadlock prevention, deadlock avoidance and deadlock recovery. In deadlock prevention, resources are granted to a packet in such a way that a request never lead to a deadlock. It can be achieved by reserving all the required resources before starting packet transmission, this leads to wasted overhead. In deadlock avoidance, resources are requested as a packet advances through the network. Deadlock is avoided by establishing an ordering between resources and granting resources to each packet in decreasing (or increasing) order. In deadlock recovery strategies, resources are granted to a packet without any check. Therefore, deadlock is possible and some detection mechanisms are required. If a deadlock is detected, some resources are deallocated and granted to other packets. Deadlock recovery is used if deadlocks are rare which is the case in interconnection networks with virtual channels[2].

3.1.1 Deadlock Avoidance

The interconnection network I is modeled by using a strongly connected directed graph with multiple arcs, $I = G(N,C)$ where N is the set of vertices and C the set of arcs. The vertices of the graph represent the set of processing nodes. The arcs of the graph represent the set of communication channels. More than a single channel is allowed to connect a given pair of nodes. Bi-directional channels are considered as two unidirectional channels. The source and destination nodes of a channel $c_i \in C$ are denoted by $s_i \in N$ and $d_i \in N$, respectively. A routing function, R , supplies a set of alternative output channels to send a message from current node n_c to the destination node n_d .

The model of deadlock avoidance relies on the concept of channel dependency[46]. When a packet is holding a channel, and then it requests the use of another channel, there is a dependency between those channels. If wormhole switching is used, those channels are not necessarily adjacent because a packet may hold several channels simultaneously. It is necessary to remove all the cyclic dependencies between channels to prevent deadlocks. There is direct dependency from c_i to c_j if and only if c_j can be used immediately after c_i by a message.

Dally and Sitez [46] introduce the concept of channel dependency graph which represents the relation between channels when a specific routing function, R , is used. A channel dependency graph D for a given interconnection network I and routing function R , is a direct graph, $D= G(C,E)$. The vertices of D are the channels of I . The arcs of D are the pairs of channels (c_i, c_j) such that there is a direct dependency from c_i to c_j . In [46], the following theorem is stated.

***Theorem:** A routing function R for an interconnection network I is deadlock free iff there are no cycles in the channel dependency graph D .*

The formal proof of this theorem is in [46]. The basic concept of the proof is once acyclic dependency graph is formed, a total order between channels can be obtained. The next examples illustrate how to build a dependency graph from a routing function.

Example 1

Consider the example of a unidirectional ring with four nodes denoted n_i , $i=\{0,1,2,3\}$ and a unidirectional channel connecting each pair of adjacent nodes. Let c_i , $i=\{0,1,2,3\}$ be the outgoing channel from node n_i . Figure 21-a shows the network. The

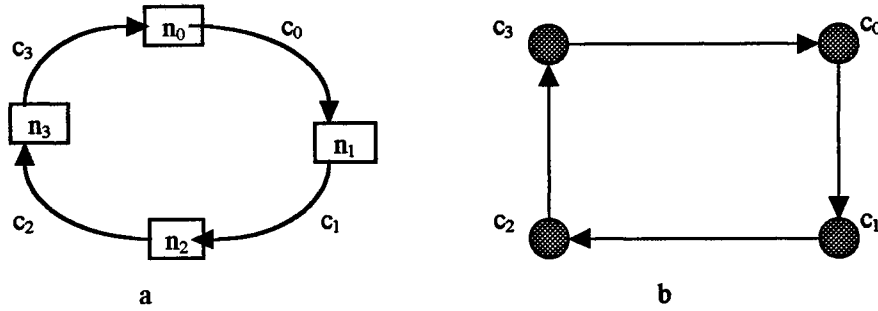


Figure 21 Interconnection graph and dependency graph of example 1. Adopted from [46]

routing function can be stated as follows: If the current node n_i is equal to the destination node n_j then store the packet, otherwise, use $c_i, j \neq i$. A packet at node n_0 destined for n_2 can reserve c_0 and then request c_1 . A packet at node n_1 destined for n_3 can reserve c_1 and then request c_2 . A packet at node n_2 destined for n_0 can reserve c_2 and then request c_3 . A packet at node n_3 destined for n_1 can reserve c_3 and then request c_0 . It is easy to see the deadlock because every packet has reserved one channel and is waiting for a second channel occupied by another packet. Figure 21-b shows the dependency graph of above example. It is clear from the graph there is a cycle in the dependency graph. Therefore, according to the above theorem the routing function of this example is not deadlock free.

Example 2

Let us consider that every physical channel c_i is split into two virtual channels, c_{0i} and c_{1i} , as shown in Figure 22-a. The routing function can be stated as follows: If the current node n_i is equal to the destination node n_j , then store the packet, otherwise, use c_{0i} , if $j < i$ or c_{1i} , if $j > i$. The cyclic dependency has been removed. Figure 22-b shows the channel dependency graph. Channel c_{12} is devoted to serve the messages destined for n_3 . Therefore, it is not dependent on any other links. Channel c_{11} depends on channel c_{12} .

Channel c_{10} depends on c_{11} and c_{12} . Same argument can be used for the relation between channels c_{01}, c_{02} and c_{03} . Channel c_{03} depends on channels c_{01} and c_{11} to forward messages from n_3 to n_1 and n_2 . It is clear from the graph there is no cycle dependency between channels.

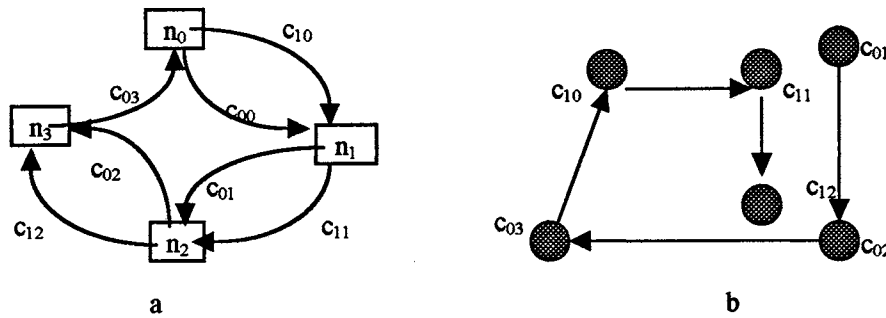


Figure 22 Interconnection graph and dependency graph. Adopted from [46]

Example 3

Let us now consider the same network in Figure 22-a but with some modifications to the routing function to give more flexibility in using channels. The new routing function is: If the current node n_i is equal to the destination node n_j , then store the packet, otherwise, use either c_{0i} , $\forall j \neq i$ or c_{1i} , for all $j > i$. Now, node n_2 can use either c_{02} or c_{12} to send messages to n_3 . Node n_1 also can use either c_{01} or c_{11} to send messages to n_2 or n_3 . Node n_0 also can use either c_{00} or c_{10} to send messages to n_1 , n_2 or n_3 . We increase the channel utilization but there are cyclic dependencies as it shown in Figure 23. However, there is no deadlock. Channel c_{12} is devoted to serve the messages destined for n_3 . Therefore, its buffers should be eventually empty. Messages in c_{11} can be consumed by n_2 or use c_{12} . Thus, channel c_{11} buffers should be eventually emptied. Messages in c_{10} or in

c_{00} can be consumed by $n1$ or use c_{11} . Thus, c_{10} and c_{00} buffers should be eventually emptied. Since c_{00} is empty, c_{03} , c_{02} and c_{01} can be eventually emptied.

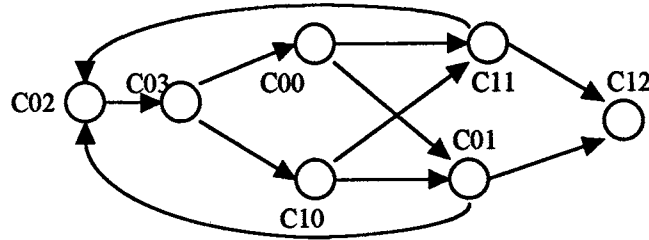


Figure 23 Channel Dependency Graph of Example 3

As it is clear from above example, deadlock can be avoided even if there are cyclic dependencies between some channels. The key idea is to provide a path free of cyclic dependencies to escape from cycles. There is at least one packet from each cycle that should be able to select the escape path at the current node. The main point is to have acyclic escape path. In order to do so, we can restrict routing function in such a way that it only supplies channels belonging to the escape paths as routing choices. If a routing function supplies a given set of channels to route a packet from the current node toward its destination, the restricted routing function will supply a subset of those channels. The restricted routing function will be referred to as routing subfunction. If R is a routing function and $R1$ is a routing subfunction of R , we have $R1(x,y) \subset R(x,y) \forall x,y \in N$. The set of channels provided by $R1$ is $C1 = \cup_{x,y \in N} R1(x,y)$.

Channels supplied by $R1$ for a given packet destination will be referred to as escape channels for the packet. However, packets can be routed by using all the channels supplied by the routing function, R . To get a deadlock free routing, the routing algorithm given by $R1$ should be deadlock free. Therefore, there must be no cycles in channel dependency channels given by $R1$. In other words, the channels of $C1$ be connected and

not involve in cyclic dependency. When we consider only the channels given by subfunction, $R1$, we call the graph the *extended channel dependency graph*[18]. The extended dependency graph for

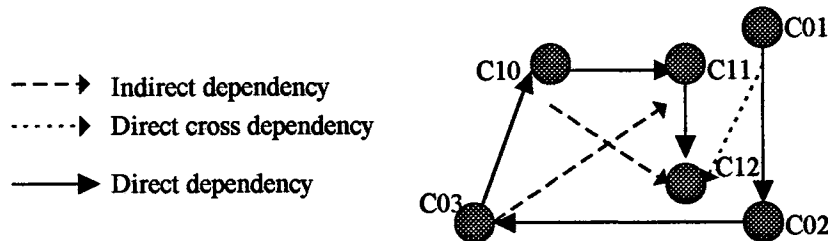


Figure 24 The extended dependency graph of example 3

the previous example is shown in Figure 24. In previous examples we consider only direct dependency. However, other kinds of dependency could exist in wormhole routing because the message can hold several channels simultaneously. Direct dependency, direct cross dependency and indirect dependency are explained in [18].

The following theorem states the necessary and sufficient conditions for deadlock-free adaptive routing.

Theorem : A connected and adaptive routing function R for an interconnection network I is deadlock free if and only if there exists a routing subfunction $R1$ that is connected and has no cycles in its extended channel dependency graph.

For proof of this theorem, the reader is referred to [18].

The widely-used approach to deadlock avoidance in wormhole routing is using virtual channels to prevent circular wait by imposing constraints on the allocation of the virtual channels. The main concept is to divide the virtual channels into classes (adaptive and deterministic). A packet is first routed on the adaptive channels and is switched to the

deterministic channels when a possibility of deadlock arises. Some deadlock-free adaptive routing algorithms will be discussed in section 3.3.

3.1.2 Deadlock Recovery

To achieve high degree of adaptiveness, deadlock recovery techniques do not impose any restriction on routing functions, thereby allowing deadlocks to form. These techniques require a mechanism to detect and resolve potential deadlock situations. When a deadlock is detected, one or more packets are obliged to release the buffer resources they are keeping, allowing other packets to use them, and breaking the deadlock. The deadlock recovery is useful if deadlocks are rare which is the case in interconnection networks with virtual channels as been proven in [2][47].

In [47], the factors which effect the probability of deadlock formation have been studied. As found in [47], the following interrelated factors influence the probability of deadlock formation: routing freedom, the number of blocked messages in the network and the number of resource dependency cycles. The deadlock in interconnection networks can be highly improbable when sufficient routing freedom is provided by the network and fully exploited by the routing function. Routing freedom corresponds to the number of routing options available to a message being routed at a given node within the network. It can be increased by adding physical channels, adding more virtual channels per physical channel, and by increasing the adaptivity of the routing function.

Blocked messages are those messages in the network which can not acquire any of the alternative channels required to make progress at a given point of time. The number of

blocked messages decreases as the network's capacity to hold messages increases. Thus, the number of blocked messages can be decreased by adding physical channels, adding virtual channels, increasing virtual channels buffer depth and decreasing packet length. The injection rate also effects the number of blocked messages.

As routing freedom is increased, the probability of deadlock decreases exponentially because the number of blocked messages decrease exponentially and the requirements needed to form a deadlock is increased exponentially. It has been found that using 2 virtual channels with fully adaptive routing function eliminates all deadlocks in a 2-d mesh with wraparound links (tours) topology[47].

Since the deadlock occurrences are extremely rare, it is not appropriate to limit the adaptivity of the routing algorithm to solve an infrequent event. Also assigning some virtual channels to prevent deadlock complicates the router design and causes poor channel utilization.

As we mentioned earlier, the deadlock recovery scheme needs deadlock detection mechanism. A deadlock configuration often involves several packets. Thus, completely accurate deadlock detection mechanisms are not feasible because they require exchanging information between nodes. Therefore, less accurate heuristic mechanisms are usually used. Deadlock detection mechanism using a time-out heuristic can be implemented. If a header flit is blocked for longer than a certain time, it should be considered in a deadlock situation. However, a heuristic deadlock detection mechanism may not detect deadlock immediately and they may indicate that a packet is deadlocked when it is simply waiting for a channel occupied by a long packet.

Once deadlock is detected, there are several alternative actions that can be used to release the buffer resources occupied by deadlocked packets. Deadlock recovery techniques can be classified as regressive or progressive[36]. Regressive techniques deallocate resources from deadlocked packets by killing them and resend the packet. A packet can be killed by sending control signal that release buffers and propagate along the path reserved by the header. After a random delay, the packet is injected again into the network[2].

Instead of killing a deadlocked packet, progressive recovery allows resources to be temporarily deallocated from normal packets and assigned to a deadlocked packet so that it can reach its destination. This technique has been used in [12] where in the router there is an additional buffer that can be used by deadlocked packets. These buffers form a deadlock-free lane which can be considered as a floating virtual channel shared by all physical dimensions of a router. When a deadlock is detected, a packet is switched to the deadlock-free lane and routed adaptively to its destination. To ensure that the special lane is deadlock free, only one packet should be allowed to use it at any given time.

3.2 Deterministic Routing

In deterministic routing, the path from the source to the destination is completely determined by the routing information obtained from the current address and destination addresses. For the same pair of source and destination, all packets will follow the same path. This method is also called oblivious routing. Deadlock is avoided in the deterministic routing by ordering the channels that a message need to traverse. Message traverses the channels either in ascending or in descending order. Thus, cycles are avoided

in the channel dependency graph.

Dimension ordered routing is a deterministic routing scheme where the routing algorithm selects a path that traverses network dimensions in sequence. A message traverses channels in the lowest or highest dimension with non-zero displacement until that dimension has displacement of zero. As the message never traverses in a reverse direction of the dimension ordering, there can not be any cycles and thereby non deadlock can be formed.

An example of deterministic routing is the XY routing in a 2-D mesh which always routes a message along the row, X, first and then along the column, Y. The routing directions and the possible turns that a message can make are shown in Figure 25. The turns clearly indicate that cycles can not be formed using the XY routing. Thus, the XY routing is deadlock free. Examples of routing paths between two source-destination pairs are also shown in Figure 25.

Another well-known fixed-path routing algorithm is the e-cube routing algorithm for hypercube[60]. This algorithm routes messages in a hypercube in a fixed order of dimensions (usually in increasing or decreasing order). The algorithm works as follows in an n-hypercube with $N = 2^n$ nodes. Consider a source node $S = s_{n-1} s_{n-2} \dots s_0$ and a destination node $D = d_{n-1} d_{n-2} \dots d_0$. Assuming that the dimensions are traversed in the order from the least significant to the most significant bit, the message from source S moves from node to node by correcting the bits from the least-significant bit to the most significant bit to match the bits of destination D. For example, in a 4-hypercube, messages from $S = 0101$ to $D = 1010$ always follow the path $0101 \rightarrow 0100 \rightarrow 0110 \rightarrow 0010 \rightarrow 1010$.

This fixed ordering of the dimensions forces the e-cube routing algorithm to select a unique path between a given pair of nodes. The e-cube algorithm is deadlock free because the channels are requested in specific order according to the dimension. Assume increasing order, if a message holds the channel in direction x_1 in node n_1 and try to request the

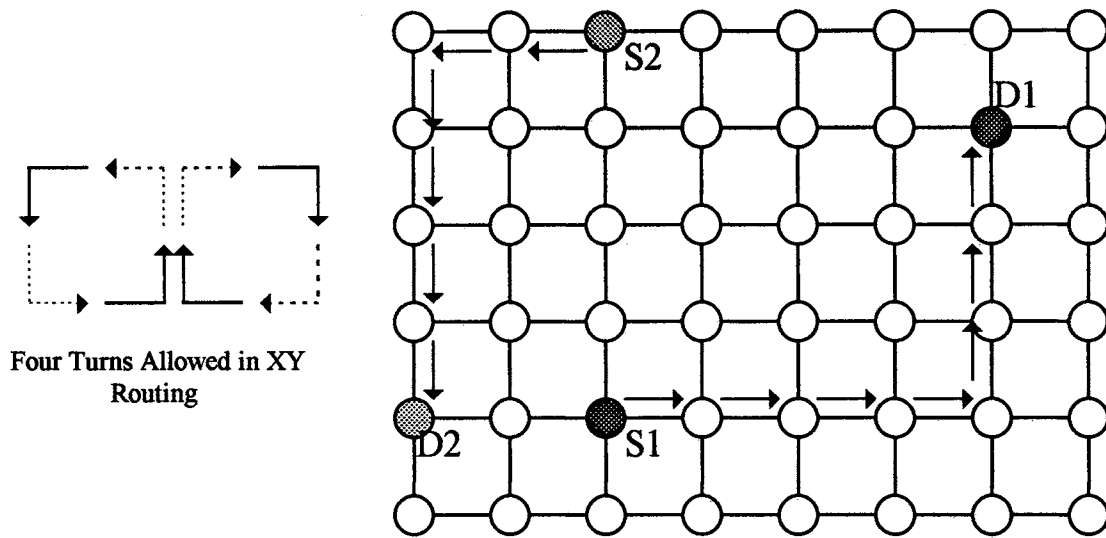


Figure 25 Dimension Order (XY) Routing in a 2d Mesh

channel in dimension x_2 in node n_2 , then we know that $x_1 < x_2$ because in e-cube the channel in lower dimension is requested before the channel in higher dimension. Let us assume a deadlock situation where another message holding the channel in dimension x_2 in node n_2 and requesting the channel in direction x_1 in node n_1 . That implies $x_1 > x_2$ which contradicts with what we have above.

The main disadvantage of deterministic routing is that it can not respond to dynamic network conditions. Some traffic patterns will produce bottleneck nodes and lead to poor performance as illustrated in Figure 26. Many studies show that adaptive routing outperforms deterministic routing [12,10]. Also when a network link or node is faulty, all

node pairs sharing the faulty link or node are disconnected even if alternative paths exist in the network between them.

3.3 Adaptive Routing

With adaptive routing, the paths can be modified to avoid faulty nodes. A routing technique is adaptive if, 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.

Adaptive routing improves both the performance and fault tolerance of an interconnection network by providing multiple paths between a source and destination (see Figure 26). Adaptive routing techniques can produce higher utilization of network resources and more robust performance. Many studies show that the performance of adaptive routing algorithm outperforms the performance of deterministic routing algorithms especially when we have non-uniform loads. However, deadlocks may appear if the routing algorithms are not carefully designed.

Many adaptive routing algorithms for wormhole networks have been reported in the literature. However, they can be divided into three groups based on how they deal with the deadlocks. The first group of adaptive algorithms avoid deadlocks by prohibiting some of the turns (changing dimension) to avoid deadlock. In many topologies, channels are grouped into dimensions. Moving from one dimension to another produces a turn in the packet route. Turns can be combined into cycles. The algorithms presented in [39,43,48,50] are examples of this group. The main advantage of this group is that it does not require virtual channels as the case in other groups. However, this group does not

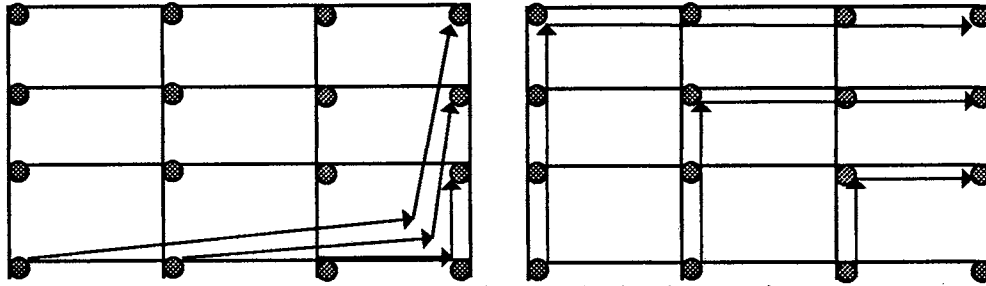


Figure 26 Deterministic and adaptive routing

provide the full adaptiveness because it prohibits some directions. Also it might form some of unwanted traffic patterns[35]. The second group avoids deadlock by using the concept of virtual channels to divide the physical network to a number of virtual networks. A separate virtual network is used for each of the possible turn (or for groups of possible turns). The algorithms proposed in [44][49][41] are examples of this group. This kind of algorithms either require large number of virtual channels as in [49] or restrict the adaptiveness as in [44]. The third group of routing algorithms partition the virtual channels to two partitions, adaptive partition and deterministic partition. The packets are routed adaptively in the adaptive partition and use the deterministic partition as an escape path if all the adaptive channels are blocked. The algorithms proposed in [18][40][13][52] are examples of this group. In the next sections some of these algorithms which represent each group will be explained.

3.3.1 Turn Restrictions Techniques

The turn model proposal in [39] provides a systematic approach to the development of adaptive routing algorithms in meshes. Deadlock occurs because the packets' routes contain turns that form a cycle. Deadlock can not occur if there is no cyclic dependency between channels. The fundamental concept behind the turn model is to

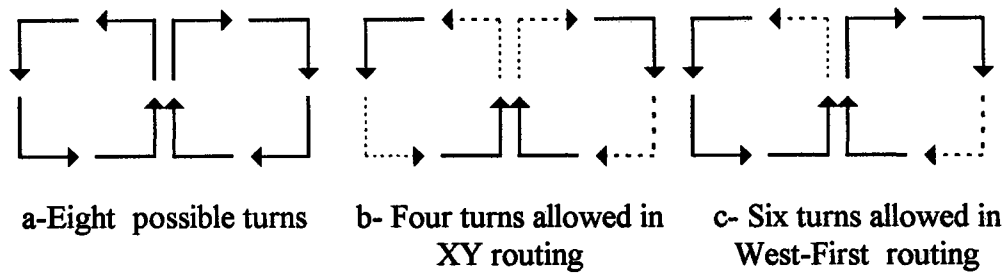


Figure 27 Possible and Allowable Turns in 2D Mesh

prohibit the smallest number of turns such that cycles are prevented. Thus, deadlock can be avoided by prohibiting enough turns to break all the cycles. In 2-D meshes there are 8 possible turns and two possible abstract cycles, as shown in Figure 27-a. The deterministic XY (row-column) routing algorithm prevents deadlock by prohibiting four of the turns, as shown in Figure 27-b. The remaining four turns can not form a cycle, but they do not allow adaptiveness. However, prohibiting fewer than four turns can still prevent cycles. For a 2-D mesh, only two turns need to be prohibited. Figure 27-c shows the six turns allowed. The two turns prohibited are the two turns to the west. Therefore, in order to travel west, a packet must begin in that direction. The west-first routing algorithm : route a packet west first, if needed, and then adaptively south, east or north. Three examples for the west-first algorithm are shown in Figure 28.

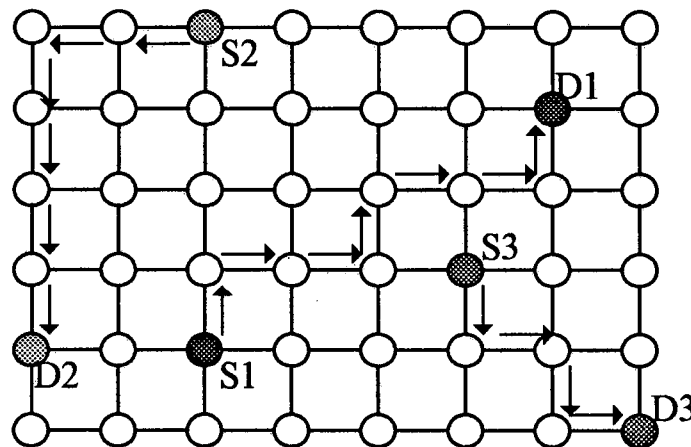


Figure 28 Examples of West-First Routing in a 2d Mesh

In addition to 2-d mesh networks, the turn model can be used to develop adaptive routing algorithms for hypercubes. The P-cube algorithm is the turn model algorithm for the hypercube. Let $s=s_{n-1} s_{n-2} \dots s_0$ and $d=d_{n-1} d_{n-2} \dots d_0$, be the binary representation for the source and destination respectively. The set E consists of all the dimension numbers in which s and d differ. The size of E is the Hamming distance between s and d . Thus, $i \in E$ if $s_i \neq d_i$. E is divided into two disjoint subsets, E_0 and E_1 , where $i \in E_0$ if $s_i = 0$ and $d_i = 1$, and $i \in E_1$ if $s_i = 1$ and $d_i = 0$. The fundamental concept of P-cube routing is to divide the routing selection into two phases. In the first phase, a packet is routed through the dimensions in E_0 in any order. In the second phase, a packet is routed through the dimensions in E_1 in any order[36]. Possible paths from 0101 to 1010 in P-cube are shown in Figure 29.

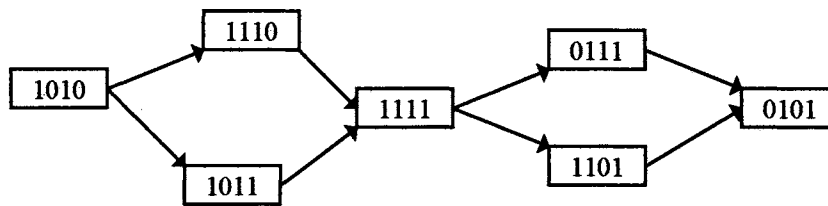


Figure 29 Possible paths from 1010 to 0101 in P-routing

Cycles can not exist because traversing dimensions in E_0 because it moves from a node to a higher numbered node (we traverse in ascending order). To form a cycle, at least one channel must exist from a node to a lower numbered node. In other words, if a packet holding a channel in node n_1 is requesting a channel in node n_2 , we can not find another packet holding a channel in n_2 and requesting a channel in n_1 . For similar reasons, cycles can not exist because traversing dimensions in E_1 . Finally, since packets only use dimensions in E_1 after traversing all of the dimension in E_0 , deadlock freedom is preserved.

The algorithm prohibits turns from dimension E_1 to dimension E_0 and this is sufficient to prevent cycles.

3.3.2 Virtual Networks Technique

One general adaptive routing technique works by partitioning the channels into disjoint subsets. Each subset constructs a subnetwork. Packets are routed through different subnetworks, depending on the location of destination nodes. Figure 30 illustrates the application of this method to a 2D mesh. As Figure 30 shows, the mesh contains an additional pair of channels added to the Y dimension. This extra pair can be added by using the concept of virtual channels. The network can be partitioned into two subnetworks called the +X subnetwork and the -X subnetwork, each having a pair of channels in the Y dimension and a unidirectional channel in the X dimension. If the destination node is to the right of the source, $d_x > s_x$, the packet will be routed through the +X subnetwork. If $d_x < s_x$, the -X subnetwork is used. If $d_x = s_x$, the packet can be routed using either subnetwork. This double Y-channel routing algorithm is minimal and fully adaptive. The algorithm is deadlock-free because no cyclic dependency between channels can be made in any of the two subnetworks.

Linder and Harden[49] extend the above concept to other topologies including the hypercube. Their method of designing adaptive routing algorithm is to create an independent virtual channel network for each combination of directions packets that can be routed and then overlay the virtual networks onto the physical network, dividing each physical channel into as many virtual channels as are mapped onto it. The large number of virtual channels is the main drawback of this method.

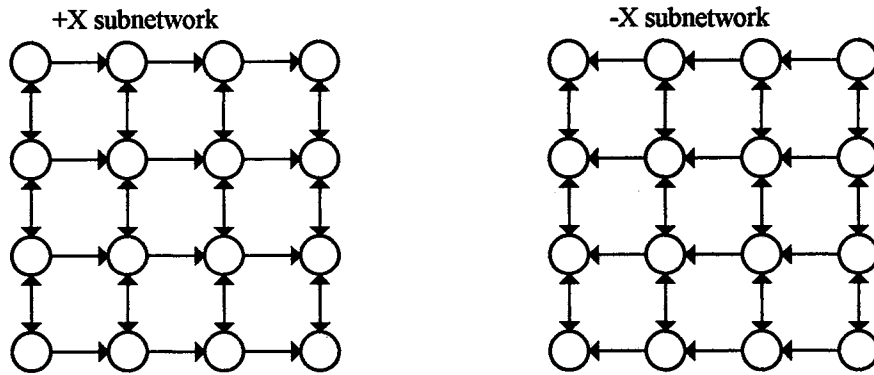


Figure 30 Virtual networks for a 2-d mesh in double-y algorithm

3.3.3 Escape Channels Technique

Dally and Aoki [13] have proposed an adaptive routing scheme based on the concept of dimension reversal. Each physical link is divided into $r+1$ virtual channels, numbered from 0 to r . A packet is allocated to virtual channels using a number count of dimensional reversals (DR). All messages are started with a DR of zero. Each time packet is routed from Y dimension to the X dimension, a dimension reversal, the DR of a message is incremented.

Two allocation algorithms, static and dynamic, were proposed. The static algorithm separates the virtual channels into classes numbered zero to r , where r is the maximum number of dimension reversals permitted. Messages with a $DR < r$ are allowed to route freely in only a virtual channel of class DR. If a message has a $DR = r$, it must be routed in dimension-order in the virtual channel of class r . This algorithm divides the physical network into $r+1$ virtual subnetworks. Every time a turn has been made the packet is moved to the next virtual subnetwork. The move from a subnetwork to the next subnetwork happens in specific order. This algorithm is deadlock free because no cycle

can be made between subnetworks. When no more subnetwork available we switch to deterministic subnetwork which is always deadlock-free.

The dynamic algorithms allow messages to route in any direction with no limit on the number of dimension reversals. The virtual channels are divided into two classes, adaptive and deterministic. Messages are routed first on the adaptive channel. Whenever a packet acquires a channel, it labels the channel with its current DR number. A message with a higher DR can not wait for a channel labeled with a lower DR. If all channels with equal or lower DR are occupied, a message must change to the deterministic channels and is not allowed to use adaptive channel again. This algorithm is deadlock free because no circular wait can be formed. The packet with high DR never waits for a channel occupied by another packet with lower or equal DR.

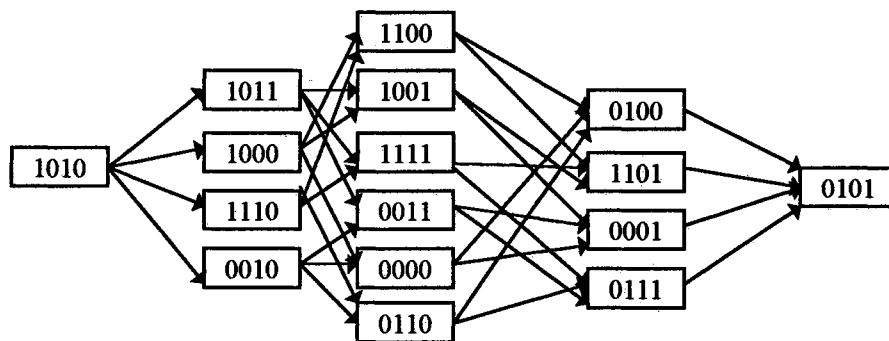


Figure 31 Possible paths from 1010 to 0101 in Duato routing algorithm

As in the previous algorithm, Duato proposes separating virtual channels into deterministic and adaptive partitions [18]. The packet can be routed adaptively in any minimal path using the adaptive portion of virtual channels. If all of them are busy, route over the deterministic channels. The packet can be routed again using the adaptive channels. The only requirement is that the routing subfunction over the deterministic channel must have an acyclic extended channel dependency graph.

For the hypercube, Duato uses the e-cube algorithm for deterministic channels and uses Idle algorithm[10] for adaptive channels. E-cube is a deadlock free deterministic routing algorithm. In contrast, Idle routes over any profitable link at an intermediate node and is not deadlock free. By combining e-cube and Idle we get a deadlock-free adaptive routing algorithm. Figure 31 shows possible paths between 0101 and 1010 when using Duato algorithm without deadlock. This algorithm is deadlock free because its extended channel dependency graph has no cycles as been proven in [40].

For 2d-mesh, Duato uses the XY algorithm of deterministic channels and uses the minimal adaptive algorithm for adaptive channels. XY is a deadlock free deterministic routing algorithm. Minimal adaptive algorithm routes over any profitable link at an intermediate node and is deadlock prone. By combining XY and minimal adaptive we get a deadlock-free adaptive routing algorithm.

Schwieber and Jayasimha [53][54] used Duato's technique to propose an optimal fully adaptive routing algorithm for meshes, opt-y algorithm. They improved Duato's algorithm by allowing limited adaptiveness in deterministic channels. Like double-y algorithm (explained in section 3.3.2), opt-y algorithm uses one virtual channel per physical channel for X dimension and two virtual channels(Y1 and Y2) per physical channel for Y dimension (6 virtual channels per node). In opt-y, virtual channel Y2 can be used adaptively without any restrictions but the use of virtual channel Y1 is restricted so it can be used as an escape channel. Double-y routing algorithm uses one of Y channels, Y1, for packets traveling X-, and the second set of Y channels, Y2, for packet traveling X+. To increase the adaptiveness, opt-y allows all the turns between X and Y2 channels as well as turns between X+ and Y1 channels. Turns from Y1 to X- channels are prohibited.

Turns from X- to Y1 channels and 0-turns between Y1 and Y2 channels are restricted. The restricted turns are only allowed when the packet has completed its movement along X- channels (the X-offset is zero or negative). Figure 32 shows the turns allowed by the opt-y algorithm. Moreover, Jayasimha et al [55] propose a minimal adaptive routing algorithm for 2-D meshes which reduce the virtual channels requirement to only 5 channels per node (double virtual channels in only in X+ direction or in X- direction).

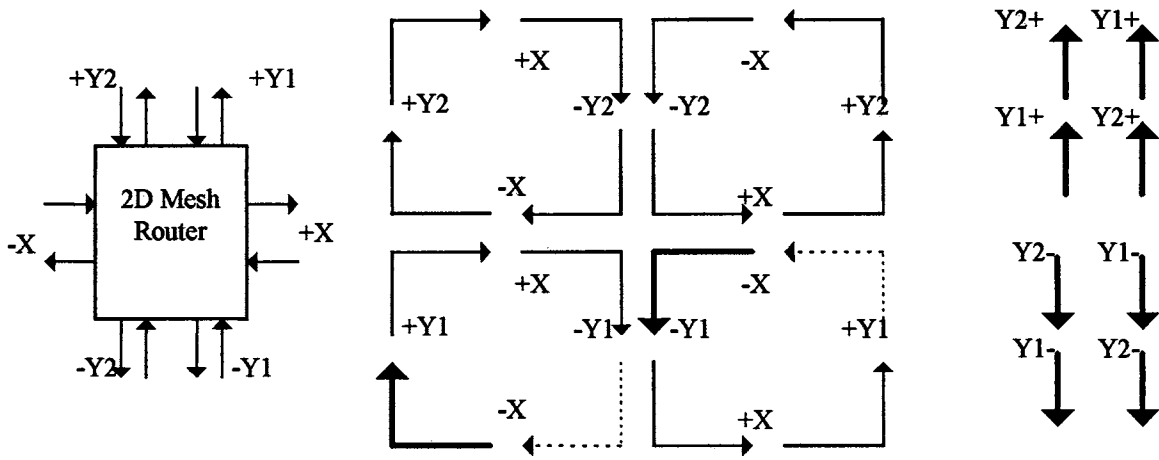


Figure 32 Turns in opt-y. Thin solid lines used for unrestricted turns. Thick sold lines are used for restricted turns. Dotted lines are used for prohibited turns. (adopted from [54])

3.3.4 Adaptive Algorithms Based on Deadlock Recovery

As we mention in section 3.1.2 the simulation studies in[2] and [47] show that deadlocks are extremely rare and even might not exist at all if more than one virtual channel is available. Therefore, deadlock recovery seemed more attractive than deadlock prevention. Devoting some virtual channels for deadlock avoidance is unjustifiable resource wasting to solve a rare problem. In [2] and [12], routing strategies based on deadlock recovery have been proposed. The algorithm proposed in[2] is explained in details in chapter 4. In [12] a deadlock recovery scheme , Disha, have been proposed.

Disha permits unrestricted routing on all exiting virtual channels which give us true fully adaptive routing. If none of these channels are free during this routing cycle, the packet is blocked. After several attempts to route the packet, the router may consider that this packet is deadlocked. A deadlocked packet is moved to the recovery path. The recovery path is constructed from deadlock buffers which are in the center of the router and can be accessed from all neighbor nodes. Figure 33 shows the general design of Disha router. Deadlock buffers form a deadlock-free lane. Once a packet is considered deadlocked, it is moved to deadlock-free lane until it reaches its destination. At any point of time, only one packet can use the deadlock-free lane. Deadlocks can be detected with a time out. If a packet is unable to make progress for a time duration corresponding to a time-out, it presumes a potential deadlock situation and becomes eligible for recovery. A packet time-out does not necessarily imply deadlock; the time-out mechanism is simply sufficient guarantee that deadlock will never occur.

Deadlock recovery mechanism provides the chance to design true fully adaptive algorithms. When routing is not restricted, no virtual channels are dedicated to avoid deadlocks. Instead, virtual channels are used for the sole purpose of improving channel utilization and adaptivity. Hence, true fully adaptive routing is permitted on all virtual channels within each physical channel. Routing restrictions on virtual channels are completely relaxed so that no ordering among these resources is enforced. In [51], true fully adaptive routing algorithm, Disha Routing Algorithm, been proposed based on deadlock recovery.

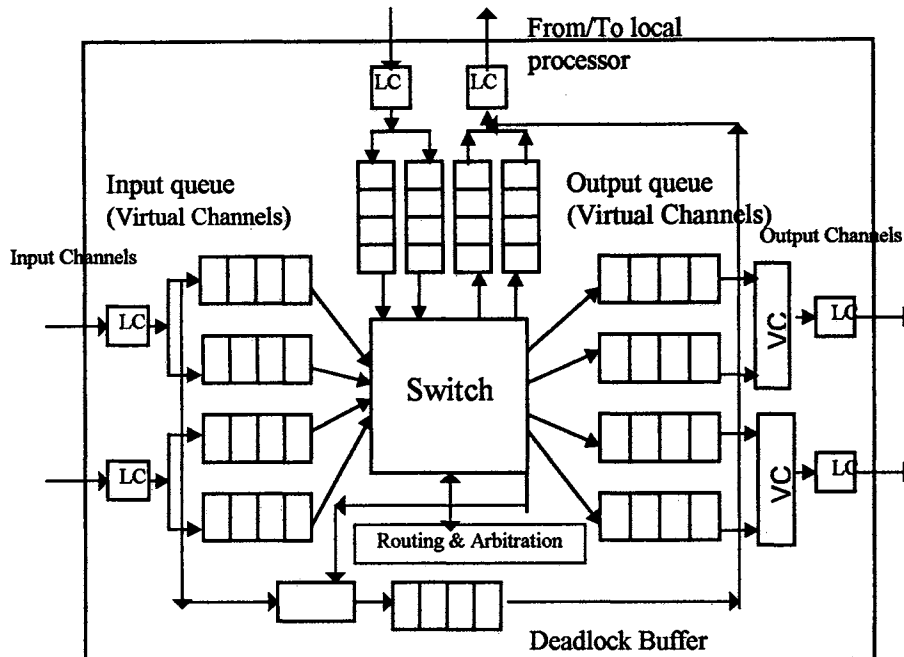


Figure 33 Router model of Disha

3.4 Fault Tolerant Algorithms

As the number of elements in a multicomputer increases, the likelihood of one or more elements failing increases too. Thus system reliability becomes a key issue in the design and implementation of large scale multicomputers. Fault-tolerant systems aim to provide continuous operation in the presence of faults. The reliability of the interconnection network is very important for the reliability of the whole system.

Two different types of faults are considered: node failure and communication channel failure. When a node fails, all physical links belonging to the failed node are also considered faulty. The failed node can no longer send or receive any messages and is removed from the network. When a physical link fails, all virtual channels on that particular physical link are considered faulty also. Only node faults are considered in this dissertation. Failures can be either static or dynamic. Static failures are known by the system before communication starts. Dynamic failures appear at random during the

execution time. Static failures are considered in this section. Dynamic failures will be discussed in chapter 4 and chapter 5. There are two fault models, coalesced fault model and random fault model [15]. In coalesced fault models, adjacent faulty nodes are coalesced into a fault regions. In random fault model, faults occur on random nodes and do not form a coalesced fault region. In current multiprocessors, failures occur in a few random nodes rather than correlated in continuous large blocks [15].

The fault-tolerant algorithms should grantee that all non faulty nodes should be reached in the presence of a given number of faults. Routing around faulty nodes (adaptiveness) is the key to achieve fault tolerance. But not all adaptive routing algorithms are fault-tolerant because in some of these algorithms it reaches a point where the packet have only one choice(path)[13], and if it happens that one of the nodes in that unique path is faulty then the packets who must use this path are undeliverable. However, fault tolerant routing schemes for n-dimensional hypercube and meshes have been proposed in [14][8][42].

In fault-tolerant algorithms, if all possible minimal paths are faulty or blocked packet must be misrouted by using a non-minimal path. Misrouting must be controlled so that livelock is avoided and newly introduced dependencies do not produce deadlock. A routing function is said to be fault tolerant if for any failed node in the network, the routing function is still connected and deadlock free [36].

In [8], every node is in one of three states, faulty, active or unsafe. Each node knows the states of its neighbors. Unsafe node is a node with at least two faulty or unsafe neighbors. The unsafe status of nodes serve as a warning that messages may become trapped if routed via these nodes. Therefore the routing algorithm is structured to

first route a message to an adjacent non-faulty node on a shortest path to the destination. If such a node does not exist, the message is routed to an unsafe node on a shortest path to the destination. If no such node exists, then the message is misrouted to an adjacent non faulty node.

In binary hypercube, every unsafe node is connected with at least one active node. A subcube is an unsafe subcube if it contains only unsafe or faulty nodes. The basic concept of this fault tolerant algorithm is to avoid unsafe subcubes. As illustrated in Figure 34 (P1 and P2), the minimal path can be obtained if either the receiver or the sender is an active node. If both of the sender and the receiver are unsafe but in different subcubes then also the minimal path can be obtained as in P3 in the Figure 34.

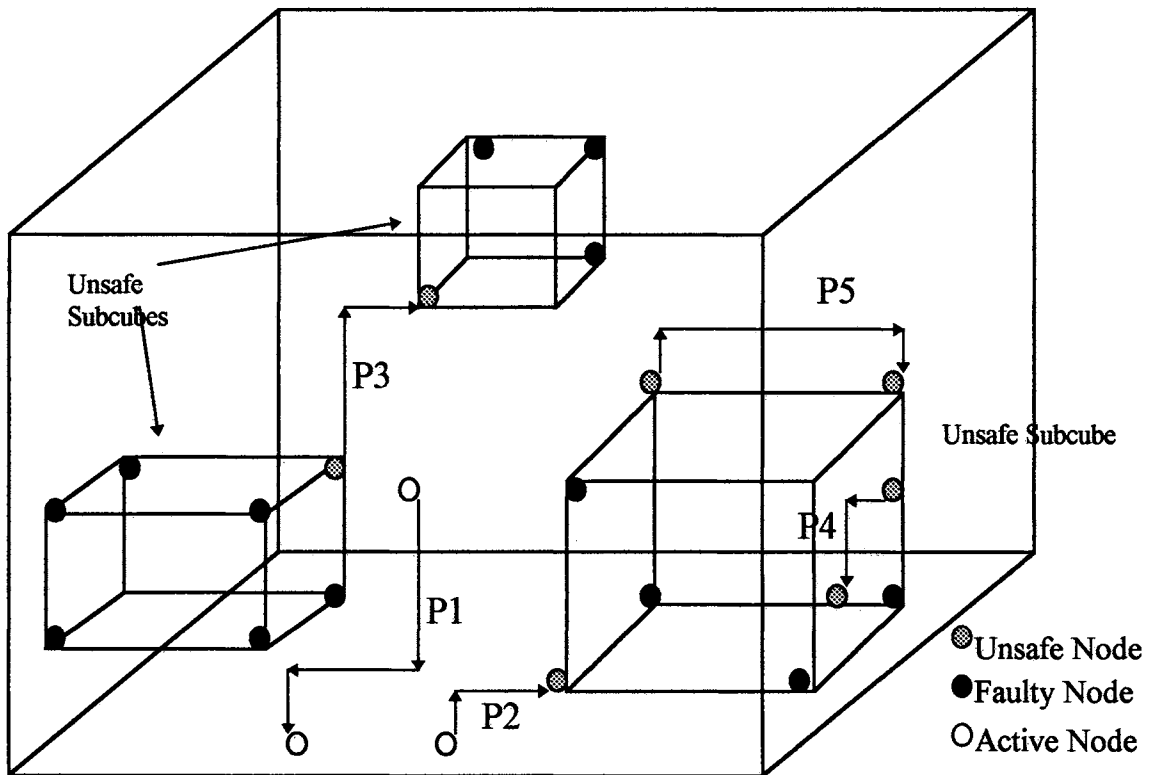


Figure 34 Fault Tolerant Routing in a Faulty Hypercube

In the last case, if both nodes (sender and receiver) are unsafe in the same subcube, then the path length is no more than the minimal plus two as in P4 and P5 in Figure 34. This algorithm is connected for the hypercube topology if the number of faults is less than $n/2$. To solve the deadlock problem in n -dimensional hypercube, Lee and Hayes [8] use $n+1$ virtual channels. However, Chiu and Wu [56] use five virtual channels to avoid the deadlock and Su and Shin [14] use only 2 virtual channels for any dimension. Moreover, the deadlock problem can be solved by deadlock recovery scheme as been explained in section 3.1.2.

Several fault tolerant algorithms have been proposed for 2-d meshes. Planar adaptive algorithm [44] have been used for fault tolerance in n -dimensional meshes. Instead of providing adaptivity in all dimensions, it restricts adaptivity to two dimensions at a time. As message progresses towards its destination, it passes through a series of adaptive two-dimensional planes and eventually reach its destination. There are two phases in Planar adaptive routing- the high level routing and the low level routing. The high level routing corresponds to the routing between the adaptive planes, and the low level routing corresponds to the routing within the adaptive planes. Let A_i represent the adaptive plane between dimension d_i and d_{i+1} . Within each plane, the plane adaptive algorithm acts as double-y algorithm (explained in section 3.3.2). The algorithm routes the packet in plane A_i till the distance in d_i is reduced to zero, then the packet is moved to plane A_{i+1} . In each plane, the set of virtual channels can be partitioned into two subnetworks: the increasing subnetwork and the decreasing subnetwork. In the increasing (decreasing) subnetwork messages only travel in increasing (decreasing) X coordinate. The Planar algorithm is deadlock-free because no deadlock can occur within each plane

and no deadlock can be formed between planes because the packet traverses planes in order. In double-y algorithm two virtual channels are needed in Y direction and one virtual channel is needed in X direction. The Planar adaptive algorithm needs three virtual channels in every direction because every node might participate in two planes, A_i and A_{i+1} . In A_i plane, a node uses two virtual channels in d_{i+1} dimension, and in A_{i+1} plane the same node uses one virtual channel in d_{i+1} dimension as shown in Figure 35.

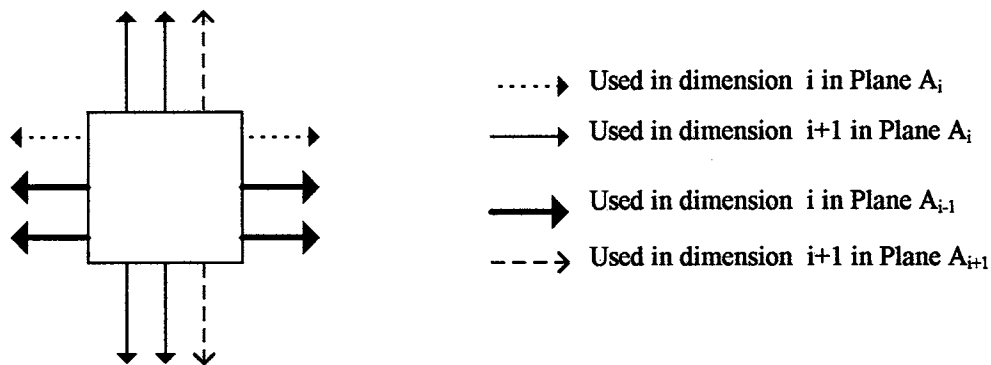


Figure 35 Usage of virtual channels in Planar algorithm

To achieve adaptiveness in n -dimensional meshes, Planar algorithm needs to traverse $n-1$ planes from plane A_0 to plane A_{n-2} . To achieve fault-tolerance, the routing algorithm must route around the faulty region. When a faulty node is encountered, the message is routed or misrouted in the vertical direction to the top (or to the bottom) of the faulty node as shown in Figure 36. In Planar adaptive algorithm, the misrouting in A_i plane must be done through the d_{i+1} direction because it is the only direction where packet can go in both ways using same subnetwork. Based on what we discussed so far, the Planar algorithm can route around faulty region, however, it can not route around the

faults in d_{n-1} dimension. Therefore, to achieve fault-tolerance, Planar algorithm added one more plane A_{n-1} which consists of d_{n-1} dimension and d_0 dimension. So, the fault in d_{n-1} dimension can be misrouted using d_0 dimension.

In 2-D meshes, to get the full adaptiveness we need to have only one plane A_0 where d_0 is the X direction and d_1 is the Y direction. But to get the fault tolerance we need two planes A_0 and A_1 . A_0 is same as in the previous case and A_1 will consist of d_1 as X direction and d_0 as the Y direction. When a packet header going along d_0 from left to

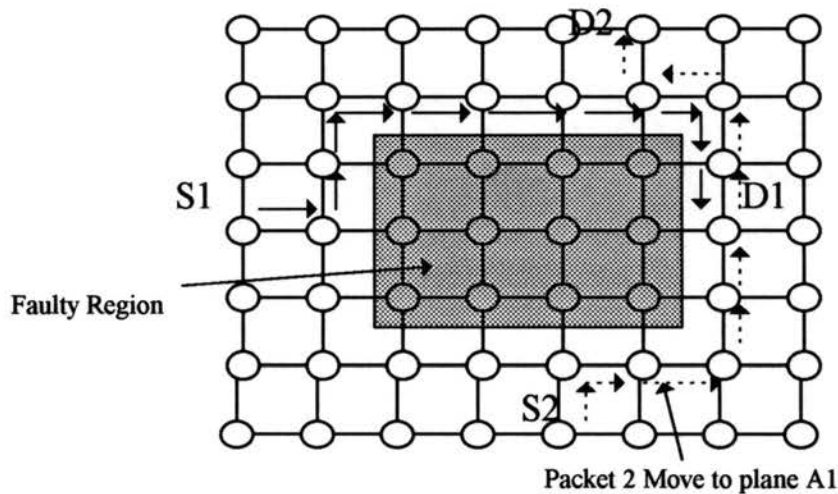


Figure 36 Routing around faulty region in Planar algorithm

right (or from right to left) encounters a faulty region and the distance is zero in d_1 direction it must route around the faulty region while it is in A_0 as in Figure 37. On the other hand, if a packet header going along d_1 from the bottom to the top and encounters a faulty region it routes adaptively till the distance in d_0 becomes zero then it moves to A_1 . It can not go around the fault using d_0 direction because the subnetworks in A_0 have only one way in the d_0 direction. In A_1 the packet can route around the faulty region because A_1 has two ways in d_1 direction.

This algorithm is connected and deadlock free if the faulty node is not a boundary node. Planar algorithm might not be able to route around a boundary faulty node sometimes because the direction of a packet has to be reversed which introduces dependency between channels and might lead to a deadlock. This problem can be solved by marking all nodes in the same boundary row with the faulty node as unsafe and prohibit routing to unsafe nodes. Other fault-tolerant algorithms can be found in [45,43,15,57,58,59].

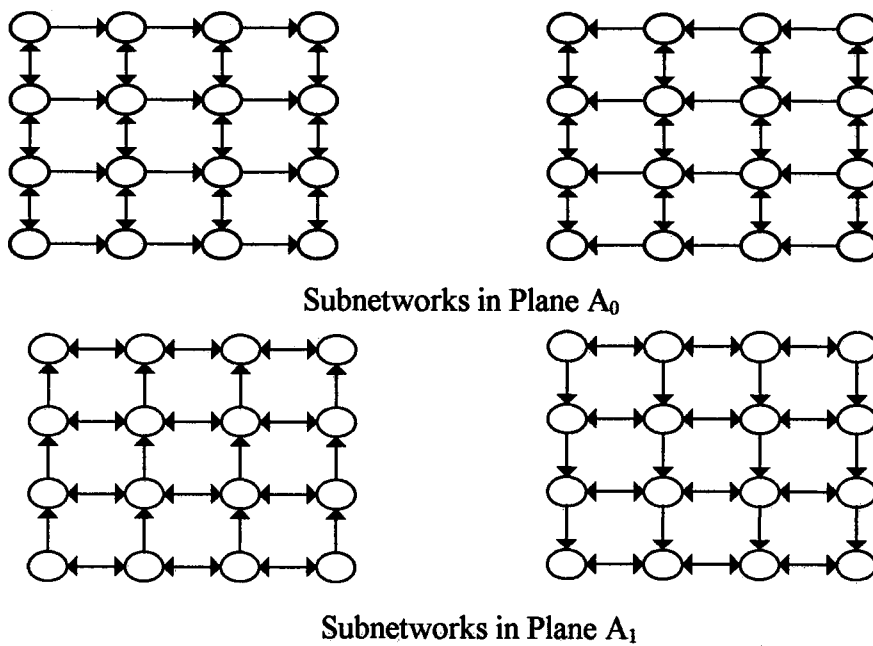


Figure 37 Subnetworks in planes A_0 and A_1 used by Planar algorithm

Chapter 4

FAULT RECOVERY

As parallel machines are scaled to large number of processors, the mean time between faults in a system decreases, making fault-tolerance issues more important. A 1000 processor MPP system would be expected to fail nearly once a day [60]. Failures can be either static or dynamic. In static faults model, as been explained in chapter 3, channels are known to be faulty prior to routing a message header over the channel. Thus, messages can be adaptively routed based on the fault status of a channel. In dynamic fault model, channels can fail at any time, and may therefore interrupt a message in progress. Since only header flit contains routing information, data flits preceding the faulty channel are blocked and cannot be routed, thus, these flits (called orphan flits) will remain in the network indefinitely occupying resources. Furthermore, the flits stored in the faulty node will be lost. A fault recovery mechanism should be able to guarantee that every packet reach its destination and no orphan flits stay in the network forever occupying resources. In this chapter we will discuss the current recovery protocols. Also, the advantages and the disadvantages of each protocol will be evaluated. The extra hardware requirements for every protocol are discussed. A message passing system should provide reliable message delivery. Traditional message passing systems ensure reliable delivery by keeping a copy of

the packet in the sender buffers until the sender gets the receiving acknowledgment from the destination[60,63]. If the receiver does not get acknowledgment in a specific amount of time, the packet will be considered as lost. Consequently, the sender resends the same packet to the receiver. For example, in B-HIVE Multiprocessor [60], when a message is entered in the transmit queue, it is marked with a timestamp. If transmission occurs normally, the message will be received by the destination, and the sender will receive a reply within the timeout period. If the timeout expired and the packet is considered as lost, the sender will re-queue the message.

The IBM SP2 with Active Messaging [63] provides reliable delivery of messages by using acknowledgment packets. Sequence numbers are used to keep track of packet losses. Unacknowledged messages are saved by the sender for retransmissions. When a message with the wrong sequence number is received, it is dropped and a negative acknowledgment is returned to the sender forcing a retransmission of the missing packet as well as subsequent packets.

Figure 38 shows the six steps required to send a message in traditional message passing systems[16]. The sender sends an allocation request to the receiver (step 1), which allocates a receiving buffer (step 2) and replies to the sender (step 3). The sender then

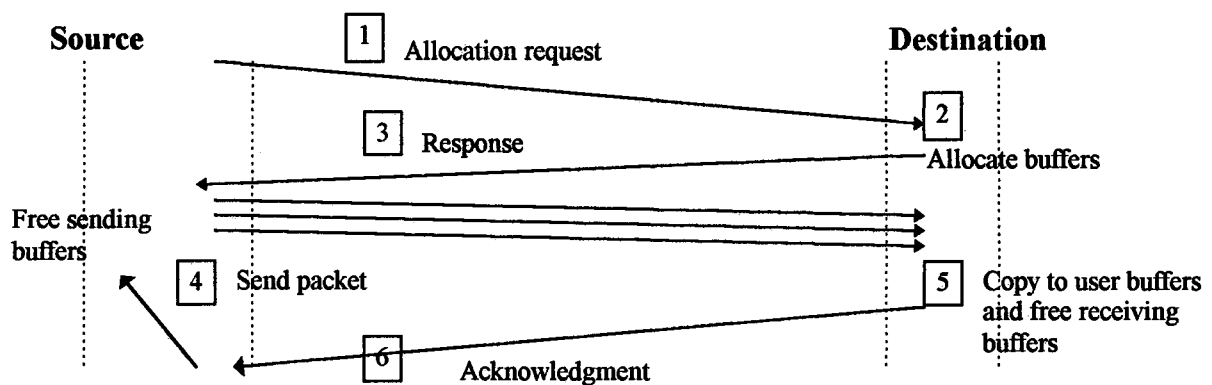


Figure 38 Reliable Messaging in traditional systems

sends the packet (step 4) and the receiver stores the packet in its receiving buffers. On the completion of the transfer, the receiver frees up the receiving buffers (step 5) after copying it into user buffers, and sends back an acknowledgment packet (step 6).

Since a copy of the data is maintained at the source waiting for acknowledgment of successful reception, fault tolerance is ensured. Studies of this message transmission scheme in [16] show that a large fraction of the end-to-end software communication cost, about 50%, is due to buffer allocation overhead and fault-tolerance overhead.

To reduce the fault-tolerance overhead, recent message passing implementations rely on the network to ensure reliable delivery. For example, the Message Passing Interface, MPI, does not provide mechanisms for dealing with failures in the communication system[64]. Also, Illinois Fast Messaging (FM) assumes reliable delivery, so protocols may be able to eliminate retransmission techniques to deal with lost packets[69]. End-to-end messaging might be accomplished via an end-to-end reporting physical network, which pairs a back-track path (using signal lines) with the corresponding parallel transmission path. The end-to-end hardware mechanism have the advantage of providing reliable messaging without additional message transmission and the corresponding CPU and software overhead. The network techniques to provide reliable delivery will be explained later in this chapter. The acknowledgment packet is not used in recent implementations. Also, no allocation messages are needed since the messages are sent in a pipeline fashion (like wormhole switching). Once the destination gets the header which contains information about the packet size, the destination allocates a buffer to store the message. In new protocols, there are no buffer allocation messages and no end-to-end acknowledgment messages.

There are at least three recovery protocols presented in the literature to handle dynamic faults[1,2,3]. However, in all these protocols a substantial overhead is involved in every message. Also, additional control lines, which complicate routers' design and increase bandwidth requirements, are required by these protocols. Re-transmitting the whole corrupted message is used in [2,3] to recover from faults.

Re-transmitting penalty may be significant when the average message length is high. Recent studies on a variety of applications demonstrate a wide range of message size from a few bytes to over 512K bytes at the application level[4]. Typical message libraries implement some form of packetization breaking long messages down into some maximum physical transfer size(512, 1024 bytes).

4.1 Reliable Router (RR)

Dally et al[1] propose "Unique Token Protocol" (UTP) to handle dynamic faults. This protocol depends on a second copy of every flit kept in preceding node and the head flit is stored in every node spanned by the packet (see Figure 39). Also, at the end of each message there is a token. When a fault happens in the middle of a message the node

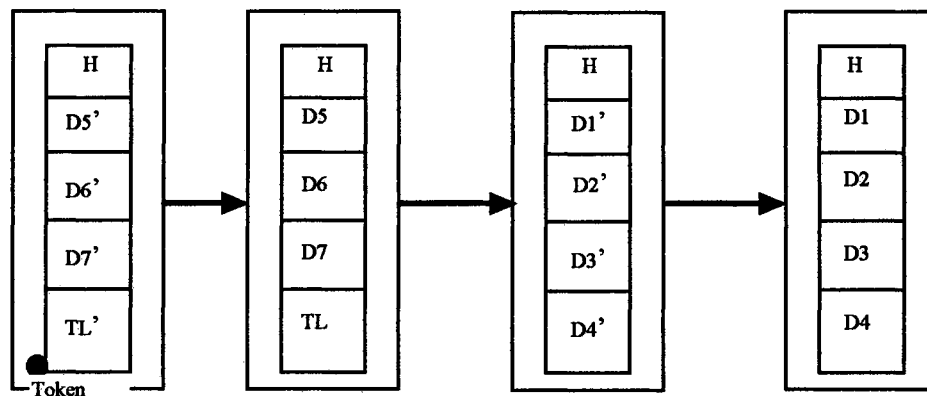


Figure 39 The UTP at the flit-level (adopted from [1]).

preceding the faulty node constructs a new message using its own copy of the message's header. The header of the new message is tagged as a special kind of head flit (restart head). A replica token is inserted at the first part of the message as a flag to distinguish between the original message and the new message. The destination assembles the two parts to get the original message. Figure 40 shows how a message is fragmented into two messages after a fault occurs.

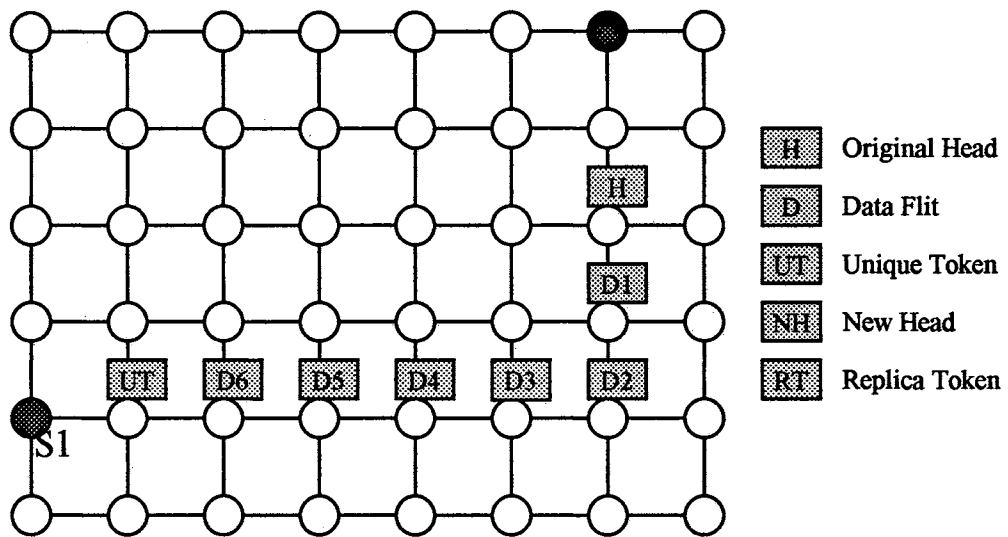


Figure 40-a Message in RR before the fault

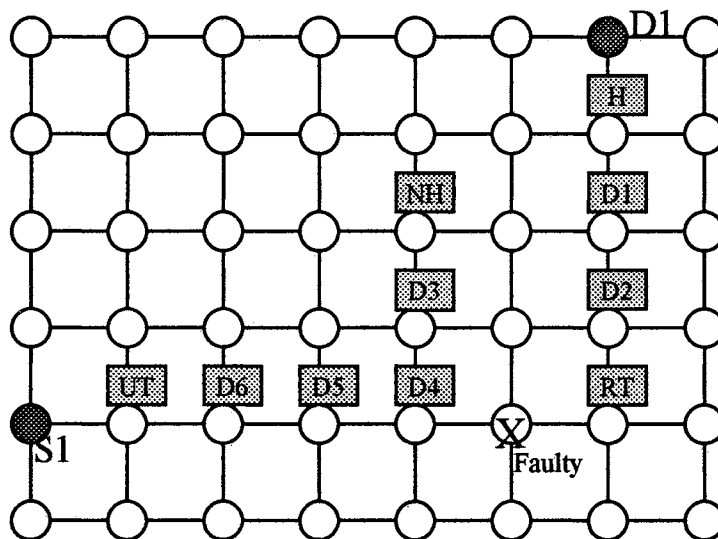


Figure 40-b Message is fragmented into two sub-messages in RR after the fault

Reliable router uses five virtual channels to divide the physical network to three virtual networks, minimally adaptive network, dimension-order network and fault-handling network. A packet using minimal adaptive network is able to route to any productive channel. The RR allocates two virtual channels to the adaptive network. A packet using dimensional-order network is routed in strict dimension order (XY algorithm). The RR allocates two virtual channels to the dimension-order network. A packet using fault handling network is permitted to use non-minimal steps to avoid faulty nodes. Even a non-minimal path is allowed, the number of turns a packet can make is restricted to avoid deadlock. The routing algorithm tries to request a channel from all the networks simultaneously. The routing algorithm will select a channel from minimal network if available; otherwise a channel from dimension-order network will be selected if available; otherwise the channel from fault-handling network will be selected.

The design drawback is that in order to keep two copies of flits all times within the network, flow control information must make two steps to the back using separate control lines. This is shown in Figure 41. Let us assume that node C copies a flit across to node D. It sends a copied signal to node B. Node B will use this information to invalidate its own copy of the flit. Reception of a copied signal will make node B send a freed signal to node A. Node A will receive the freed information and use it as indication that node B can receive one more flit.

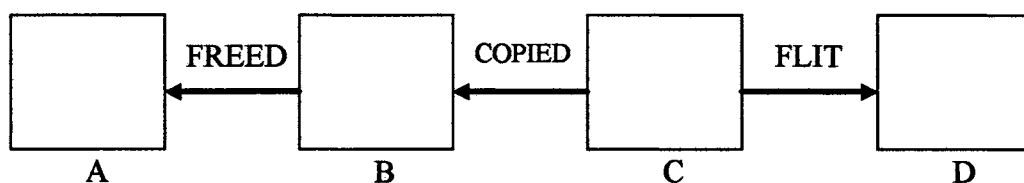


Figure 41 Flow Control in Reliable Router (adopted from [1]).

This protocol reduces the buffer requirement in the sender side and does not require retransmission of interrupted messages. On the other hand, UTP increases the buffer requirement in the routers by requiring the head of each message to be stored in each spanning node. Also, there are two copies of each flit in the path, as been illustrated in Figure 39, which means the length of each message has been doubled decreasing the performance significantly as we will see in chapter 6 when we compare the performance of RR with performance of other protocols. This protocol is not robust enough to handle multiple faults that occur in one message pipeline. If two adjacent nodes fail simultaneously, the RR can not recover the lost flits because both the original flits and the replica flits are lost. For example in Figure 39, if node 1 and node 2 fail at the same time there is no way to recover data flits D1, D2, D3 and D4.

4.1.1 Router Design in RR and hardware requirements

Figure 42 shows the Input Controller of RR. The Input Controller supports five separate virtual channels with decoupled resources. The functionality of the Input controller can be summarized as follows:

- It buffers flits in the FIFO module. The FIFO is divided into five separate banks, one for every virtual channel. Each bank behaves as a regular first-in-first-out buffer with some special state and functionality to implement retransmission in case of a fault.
- It computes the next step route of each packet based on head flit information and current output virtual channel state. The route is stored in dedicated registers so that it can be used by subsequent data flits. Route computation and storage resources reside in each Virtual Channel Module.

- It keeps track of the virtual channel buffer size in the receiving node and stops the corresponding virtual channel from transmitting any more flits in order to prevent FIFO overruns.

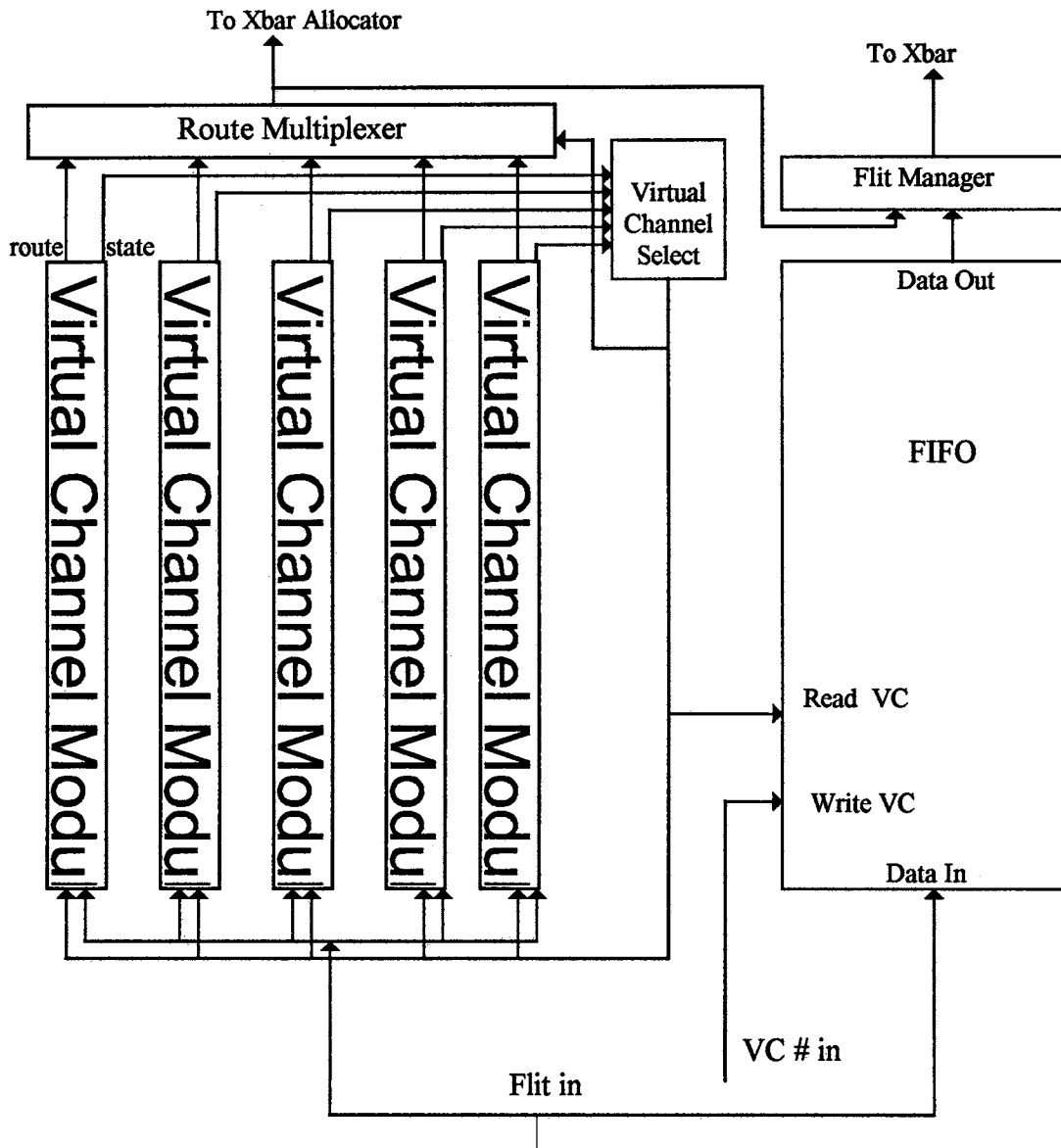


Figure 42 Input Controller Block Diagram of RR

Flits enter the Input Controller after being received from the sending neighbor. If the flit is a head, it is fed to the corresponding Virtual Channel Model and a copy of the flit is stored in the FIFO bank that corresponds to the virtual channel identifier of the incoming flit. On the other hand, if the flit type is data, it is directed to the appropriate FIFO bank. The design computes the output channel identifier (“Route”), selects a virtual channel and allocates the crossbar. The rest of the circuitry in the Virtual Channel Module is mainly concerned with internal bookkeeping. Figure 43 shows the block diagram of Virtual Channel Module. The Optimistic Router computes the address of output virtual channel based on the message’s destination and current state of output channels. It is called optimistic because it might need to be changed if another input channel requests the same output channel simultaneously.

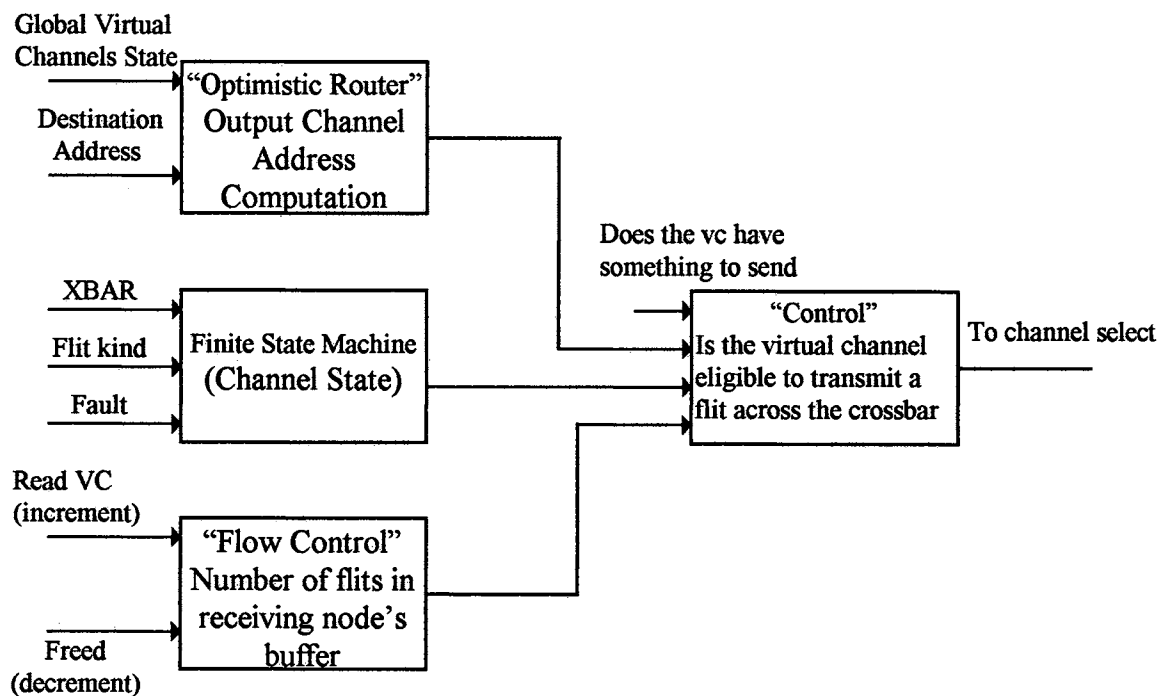


Figure 43 Virtual channel module of RR

A finite state machine, Route FSM, keeps track of the virtual channel state at all times. A simplified version of its state diagram is shown in Figure 44. When a head flit appears, the channel state switches to either NEEDS ROUTE or ROUTED state depending on whether the flit was actually accepted by the crossbar (Xbar Ack.). A channel is marked as routed only when the head flit succeeds in reserving an output controller and

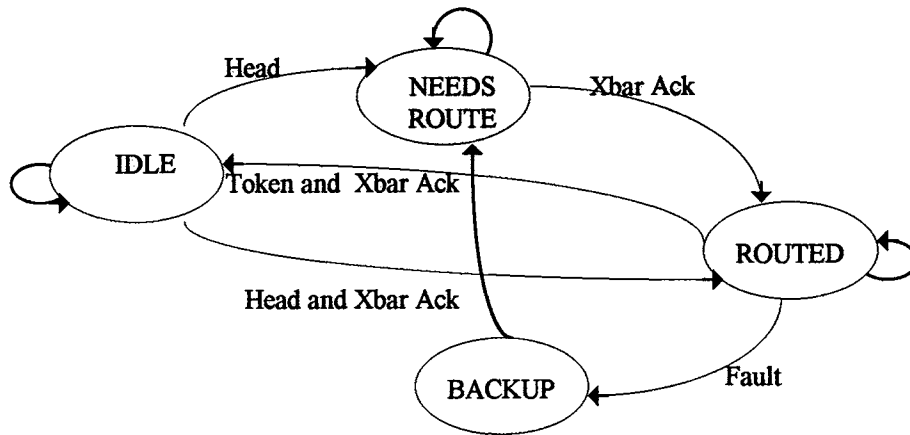


Figure 44 State Diagram of the Route FSM (adopted from [66])

gets transmitted across the crossbar. Otherwise, the Optimistic Router will try to recompute a route during next 2 cycles. The virtual channel switches back to IDLE state when a token flit goes through and the resource deallocated. If a fault occurs while the message is Routed, the channel switches to the Backup state where internal preparation for retransmission occurs. After 2 cycles, the channel goes back to the NEEDS ROUTE state and starts looking for an alternative route among the non-faulty output virtual channels.

The Flow Control Module contains a counter that is incremented each time a data flit goes across the crossbar into the virtual channel buffer storage of the neighboring node and gets decremented each time the neighboring node forwards a flit and frees up buffer storage.

Finally, the Control Module collects information from the Optimistic Router, the Route FSM and the Flow Control Module and decides whether it should declare the virtual channel eligible to transmit a flit across the crossbar. The Virtual Channel Select module selects one of eligible virtual channels to get the chance to push a flit through the crossbar. A round robin scheduler is used to arbitrate between the eligible virtual channels.

4.2 Fault-tolerant Compressionless Routing (FCR)

The main idea proposed in [2,3] is to use re-transmission mechanism to tolerate dynamic faults. When a fault is detected, the detecting routers send kill signals both forward and backward along the message path (see Figure 45). These kill signals (FKILL and BKILL) follow the virtual circuits back to the source and destination and release reserved buffers and notify the source that the message was not delivered and the destination to ignore the message currently being received. The protocol proposed by Kim et al[2] requires padding extra flits at the end of each message. To ensure reliable transmission, Fault-tolerant Compressionless Routing, FCR, requires that each message holds its path until the last data flit reaches the destination [2]. For fault tolerance, successful delivery of a message is granted if the message's entire data portion has been delivered to the destination without error. Thus, the message sender must hold the tail of the message until the last data flit is delivered. To determine whether all data flits have reached the destination, FCR takes advantage of the compressionless property of messages under wormhole routing. Worst case analysis can be used to determine the maximum number of flits that can be distributed between the sender and the receiver. The sender pads the message to ensure that all data flits reach the destination before the tail flit

has been injected into the network (Figure 46). The number of extra flits need to be padded is computed as follows: $\text{extra_flits} = B_{\text{cap}} * D$ where B_{cap} is channel capacity in flits and D is the path length.

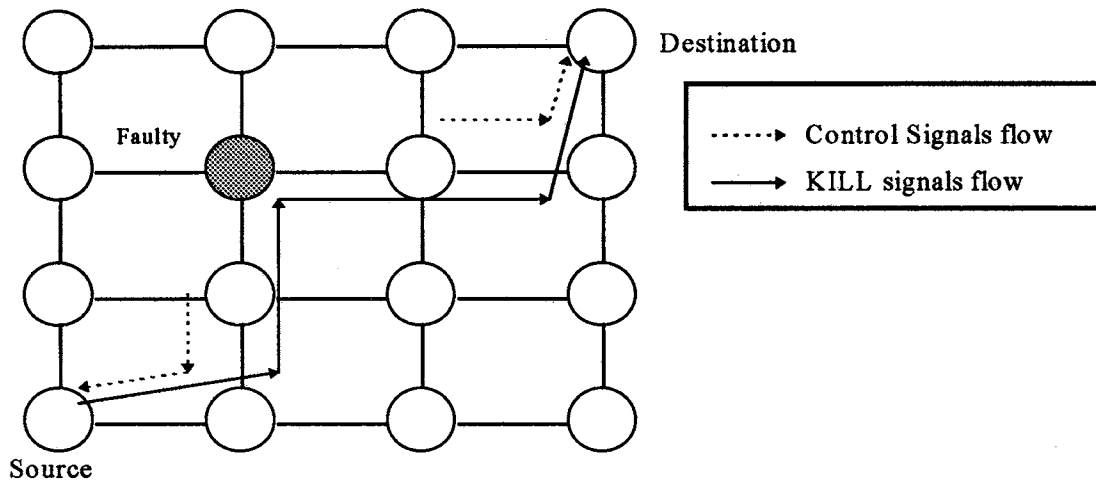


Figure 45 Tearing down an interrupted circuit. (Adopted from [3])

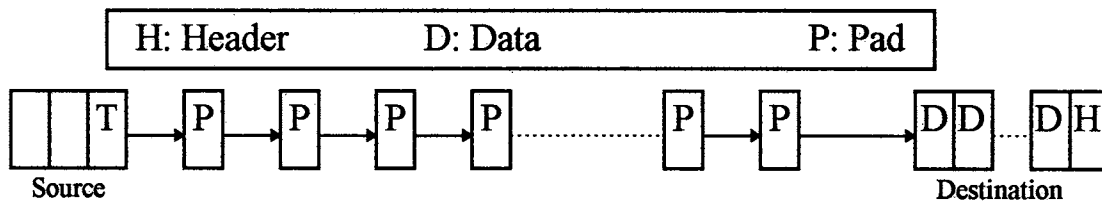


Figure 46 Message Padding in FCR networks.

When a fault is detected, both FKILL and BKILL signals are generated. FKILL signal propagates forward along the message path. If the destination receives the signal, the current message must be dropped. BKILL signal propagates backward along the message path. If the sender receives the BKILL the current message must be reinserted. When a KILL signal arrives at an intermediate node, it releases the input and output virtual channels associated with the message and the signal propagates towards the source (or the destination). If multiple faults occur in one message pipeline, this mechanism is applied recursively to fault free parts.

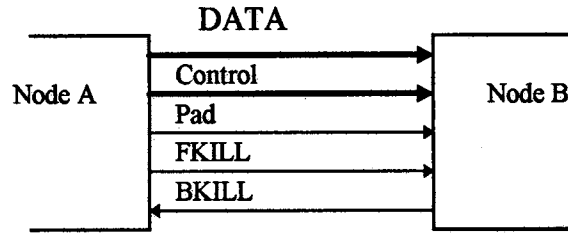


Figure 47 Data and control lines in FCR

FCR provides data recovery from dynamic faults. However, it requires additional hardware in the network interface and in the router. The channel interface for FCR router is shown in Figure 47. Three new control lines (FKILL, BKILL and Pad) are introduced. Pad signal is needed to differentiate pad flits from data flits and to indicate the end of each message

Re-transmitting will increase message latency and will decrease the throughput. Control lines required by this mechanism make the hardware cost high and complicate router's design. Most importantly, FCR attaches extra flits with every message which reduce the throughput considerably. For example, in 64-node hypercube with 2 flits buffers a 12 flits message must be padded with extra 12 flits which means 50% deterioration in the throughput.

4.2.1 Hardware Support For Data Recovery in FCR

Figure 48 shows a block diagram of the message injection subsystem in network interface. It receives formatted message from a processor and sends messages. If it receives BKILL, it restarts the sending process. The flit counter F_{inj} has the sequence number of the flit to be injected. The Ack signal from the router indicating that a flit was injected resets the elapsed time counter T_{elapse} and increases the flit counter F_{inj} . The

BKILL signal resets the flit counter, therefore the network interface will restart the sending process. When number of injected flits equals I_{min} all counters are initialized and a new message picked from input queue.

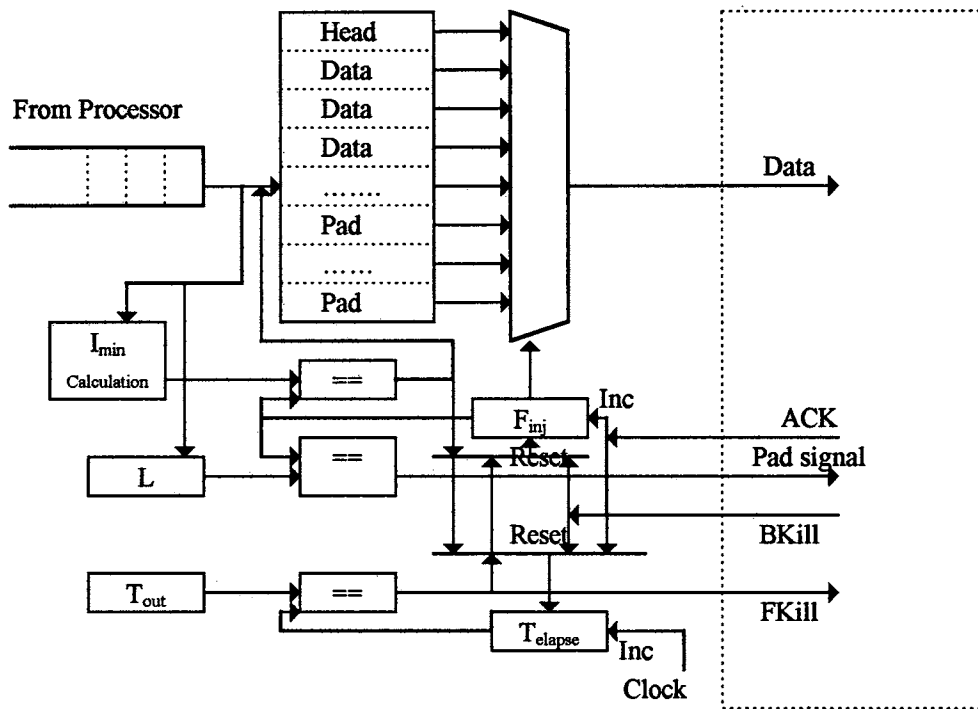


Figure 48 The Injector Network Interface in FCR (Adopted from [2])

Figure 49 shows the message receiver interface in FCR. The data flits are assembled in the interface's buffer until all data flits are arrived. The interface knows that all data flits have arrived when the Pad signal goes high, whereupon, the message is fed to the processor and the receiving interface is initialized to accept another message. The flit counter, F_{rec} , is incremented every time a flit arrived and it works as a pointer to where the next flit will be stored in the receiver buffer. When a FKILL signal goes high the flit counter, F_{rec} , is reset which means discarding all the data flits in the receiving buffers.

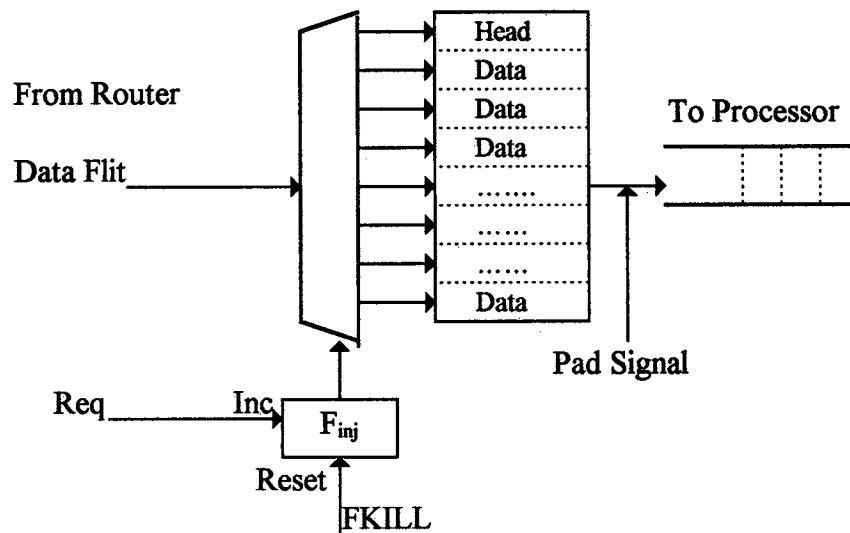


Figure 49 The message receiver interface in FCR (Adopted from 2)

4.3 Acknowledged Pipelined Circuit-Switching (APCS)

Another recovery protocol, acknowledged pipelined circuit-switching (APCS) is proposed in [3]. Same concept of KILL signal is used in this protocol. However, padding extra flits at the end of the message is not used here. Instead, a hardware message acknowledgment must be transmitted from the destination to the source on completion of message transmission (see Figure 50). The acknowledgment signal propagates through the complementary channel. In APCS, a unidirectional virtual channel is composed of data channel, a corresponding channel and a complementary channel. The routing header traverses the corresponding channel while the subsequent data flit traverse the data channel. The complementary channel is reserved for use by acknowledgment flits, kill flits and backtracking header flits.

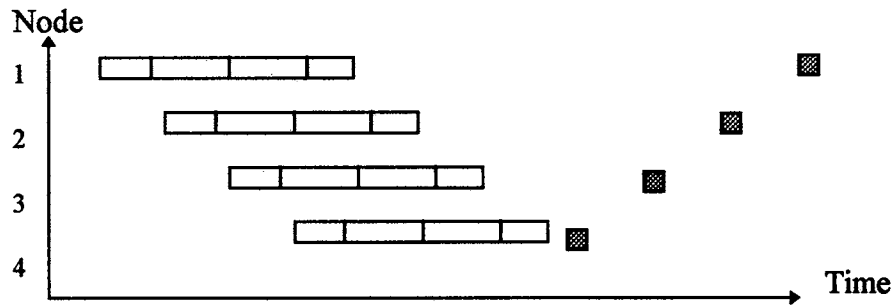


Figure 50 Acknowledgment signal in APCS

This mechanism holds virtual channels until the acknowledgment signal reaches the source via the complementary channel. Once the receiver gets the last flit of the packet, it sends the acknowledgment signal to the sender. The acknowledgment signal propagates backward from the receiver to the sender through the complementary channels and it frees all the buffers (channels) reserved for that packet. If a fault happens and interrupts a message pipeline, the interrupted message must be resend, the reserved channels must be released and the receiver must be informed to discard the flits which have been received before the fault happened. In order to release the reserved network resources, the link controller at the source end of the faulty virtual link introduces a kill flit into the complementary virtual channel of the virtual link upstream from the fault. This kill flit follows the complementary control circuit back to the source node. The link controller at the destination end of the faulty virtual link introduces a kill flit into the complementary virtual channel of the virtual link downstream from the fault. This kill flit is propagated towards the destination node. When a kill flit arrives at an intermediate node, it releases the input and output virtual links associated with the packet and is propagated toward the source (or destination).

In this protocol, the sender waits until it received the acknowledgment signal which increases the message latency for every message. When the message latency is considered, this protocol has similar effect as adding extra flits in the previous protocol. Also, the additional message acknowledgment introduces additional control flit traffic into the system. Message acknowledgments tend to have a throttling effect on injection of new messages[9]. Extra control lines are also needed in this mechanism.

Using the same concept, the nCUBE3 communication architecture [62] provides an end-to-end reporting mechanism that ensures reliable messaging, whereby the sender of a message can know quickly whether that message was delivered reliably or not. The receiver sends status information back to the sender before the established path is broken. The status information are sent through a back-track path with the corresponding transmission path. The message transmission network and back-track network are implemented as virtual network that share the same physical communication network.

CHAPTER 5

Software Based Recovery Protocols (SBRP)

In chapter 4, we have explained and evaluated the current recovery protocols in wormhole network. To achieve data recovery, the current recovery protocols incur an overhead with every message. As we have explained in chapter 4, UTP doubles every message length to be able to recover the lost flits. Also, FCR uses an end-to-end protocol by keeping the message pipeline until the sender makes sure the receiver got all data flits. This message length enlargement has a negative effect on system performance. Network bandwidth is wasted by using it to transfer redundant or dummy flits. Since the faults are infrequent events in multicomputer systems, it is unjustifiable to penalize every message to achieve data recovery. To enhance recovery protocol's performance, the data recovery overhead should be linked with the fault occurrence only. The normal (fault-free) operations should not be affected by recovery protocol. A data recovery protocol which does not penalize every message is more appealing.

We need a data recovery protocol which neither uses an end-to-end protocol nor requires multiple copies of data flits to exist in the same message pipeline. To achieve data recovery, however, there should be a mechanism to compensate for the lost flits. Since we are not in favor of keeping more than one copy of the flits in the message pipeline, the lost flits should be re-obtained from the original sender. There are two possible ways to inform the sender that some flits are lost and there is a need for retransmission. The first way is to use an end-to-end protocol to inform the sender by

hardware acknowledgment. This approach is used in [2,3] and is discussed in detail in chapter 4. The second way is to use regular messages to inform the sender. Using regular messages for data recovery was not attractive in traditional systems because of the high cost of messages and the poor communication between router's hardware and messaging system. The low cost messages and good router interfaces in current systems make the use of messages in data recovery more attractive. In this chapter, we will explain and evaluate the use of control messages instead of control lines in data recovery.

5.1 General Concept of Software Recovery Protocols

The duties of any data recovery protocol can be summarized as follows:

- The receiver gets the whole message without any missing flits.
- The receiver gets at most one copy of the whole message or any part of it.
- Any orphan flits must be removed from the buffers.
- All the buffers and channels which were reserved by a corrupted message must be released.

Messages can be used instead of control lines to inform the sender that a message has been corrupted and a whole message or a part of it need to be resend. As illustrated in Figure 51, let us assume that the faulty node is B, the node following B in the message path is A and the node preceding B in the message path is C. The recover request (RRQ) must be sent by one of the nodes who can know that a fault occurs (C, A or the receiver). A recover request (RRQ) is a small message which has at least the original (corrupted) message id and might have other information depending on the protocol used.

Since a tail flit of the corrupted message will never go through the reserved path between node A and the receiver, the reserved channel will not be released forever.

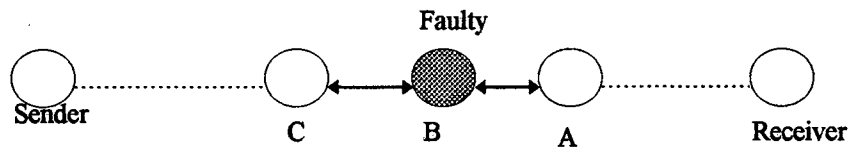


Figure 51: A message pipeline with a faulty node.

Therefore, node A must inject a tail (we will call it non final tail NFT) to release the reserved channels. Once the receiver gets the NFT, it knows that a fault happened and will act according to the specific protocol. The receiver either ignores the currently received message in case of whole message retransmission or waits for the remainder of the message in case of partial message retransmission.

To get rid of *orphan flits* and release the reserved channels between node C and the sender, node C must continue to absorb the orphan flits until it gets a tail flit. Node C will get a real message tail or a virtual tail which is injected by the sender. Once the sender gets RRQ of a message while it still injecting its data flits, the sender stops the injection of any data flits and injects a tail flit to release the reserved channels.

5.1.1 Fault Recovery Handler

Fault Recovery Handler (FRH) is a special handler executed by one of the participating nodes to respond to fault occurrence. This handler is executed by either node A, C or the receiver depending on the version of recovery protocol being used. The job of this handler is to collect information about the corrupted message. This information

include the message id, sender id and receiver id. Also, some version of the protocol might need other information as will be explained later in this chapter. After collection of such information, the FRH embeds them in a message and sends it to the sender.

5.1.2 Absorbing Orphan Flits

The orphan flits are the flits of the corrupted message located between the faulty node and the sender. These flits can not be forwarded to the receiver because the path which they must follow is not a valid path anymore. Allowing these flits to stay forever occupying network resource is not acceptable. There should be a mechanism to remove these flits from the network. The only node who is capable to remove the orphan flits is the node which precedes the faulty node (node C in Figure 51). Node C can absorb all the flits intended to the faulty node. The absorption can be done by receiving the flit in the virtual channel's receiving queue and clearing the queue immediately. The absorption process stops when a tail flit arrives.

5.1.3 Resending Handler

Once the recovery message reaches the source, the source must resend the corrupted message or part of it. We have two cases. The first case when the source is still injecting the rest of the corrupted packet. In this case, the sender restarts injecting the packet starting from the first flit as in SBRP-0 and SBRP-2 or starting from a specific flit as in SBRP-1. In the second case, the recovery message arrives after the sender finish injecting the rest of the corrupted message. In the second case, a specific handler must be invoked to reload the message from system buffers. In some systems, however, the message buffers are deallocated immediately after the packet is injected in the network.

SBRP requires that a message waits in system buffers until the message delivery is granted. The waiting time has no effect on messages' latency.

5.2 Implementation

In this section we describe three approaches of SBRP: SBRP-0, SBRP-1 and SBRP-2. All three approaches use messages to handle data lost due to nodes' fault. However, they differ in the way they handle the fault occurrence. In the first two approaches, SBRP-0 and SBRP-1, the node preceding the faulty node in the message pipeline, node C in Figure 51 sends the retransmission request to the sender. In SBRP-0 the sender sends the whole message again. In SBRP-1, however, the sender sends only part of the message as we will explained later. In the third approach, SBRP-2, the receiver sends the retransmit request after it receives a non-final tail. The following three sub-sections describe these three approaches.

5.2.1 SBRP-0

The direct way to recover flits lost is to ask the original sender to resend the whole message again. As in Figure 51, node C sends recover request to the sender. Node C needs to know the original message sender and the message id. Therefore, this approach requires a copy of a message's head is kept in every virtual channel.

In case node B goes down and corrupts a message pipeline, the actions taken by the nodes of Figure 51 are as follows:

Node A

Inserts a NFT flit at the end of the sub-message.

Node C

- Absorbs all non-head flits routed to the faulty node.
- Sends a retransmission request to the sender.

Sender node

After receiving retransmission request from node C, the sender performs the following tasks:

- Inserts a tail flit in the path of corrupted message in order to release the reserved channels.
- Retransmits the whole corrupted message again.

Receiver node

- After receiving a NFT, the receiver ignores the message currently receiving.

SBRP-0 requires that each virtual channel keeps a copy of the head of the message currently using it. This requirement is also used in RR [1] as was explained in chapter 4. Node C invokes the Fault Recovery Handler (FRH). The FRH needs to get the message id, source and destination from the head's copy. A recovery message of two fields, message id and message destination, plus a head and tail is composed in node C and is sent to the sender. Once the sender gets the recover message, it invokes the resending handler RSH to resend the corrupted message again.

5.2.2 SBRP-1

In this approach, as suggested in[1], an interrupted message can be broken down into two sub-messages. Both sub-messages have the same header but the headers will

differ in one field only to distinguish between the original head and the new head. The first sub-message consists of the original header and the flits that passed through the faulty node without being corrupted. The second sub-message is constructed by the source node after receiving the fault-handler message. We extend this concept by giving a sequence number for each head of all sub-messages. This sequence number is used by the destination to re-construct the original message even in case of the message being broken down into more than two sub-messages. The cost of this sequence number is few bits in the message head. Number of these bits equals to $\log_2(\text{number of possible sub-messages})$ which normally is small.

5.2.2.1 *Definitions*

Beside the terminology from the literature, we need the following:

- Non-Final Tail (NFT):

In case of message fragmentation, a non-final flit indicates the end of a sub-message. This tail comes at the end of each sub-message except the last one.

- Sub-message serial number field (SSNF):

In case of fragmentation, this field in a message's head shows the serial number of each sub-message. Without fragmentation, the field value is zero.

The sender increments this field by one every time it generates a sub-message.

- First flit's sequential number (FSN):

This is a flit embedded after the head of a sub-message to show the sequential number of the first data flit in a sub-message (s in Figure 53).

The receiver needs this value to deal with possible duplicates of flits. The cause of the duplication is described later.

- Maximum flits capacity between two nodes ($\text{Max}(i, j)$)

= Number of nodes between i and j * Buffer capacity per node

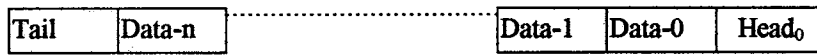
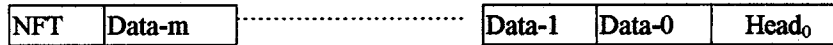
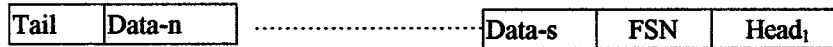


Figure 52 Normal Message Format



a- First Sub-message



b- Second Sub-message

Figure 53 A n-flits message fragmented to two sub-messages

This approach requires that each virtual channel keeps a copy of a message's head. Moreover, each virtual channel keeps track of the number of flits that have passed through it belonging to the current message. Such counter can be added easily to the current router designs. Figure 54 shows how the RR router can be modified to get such counter. The length of this counter depends on the maximum packet size.

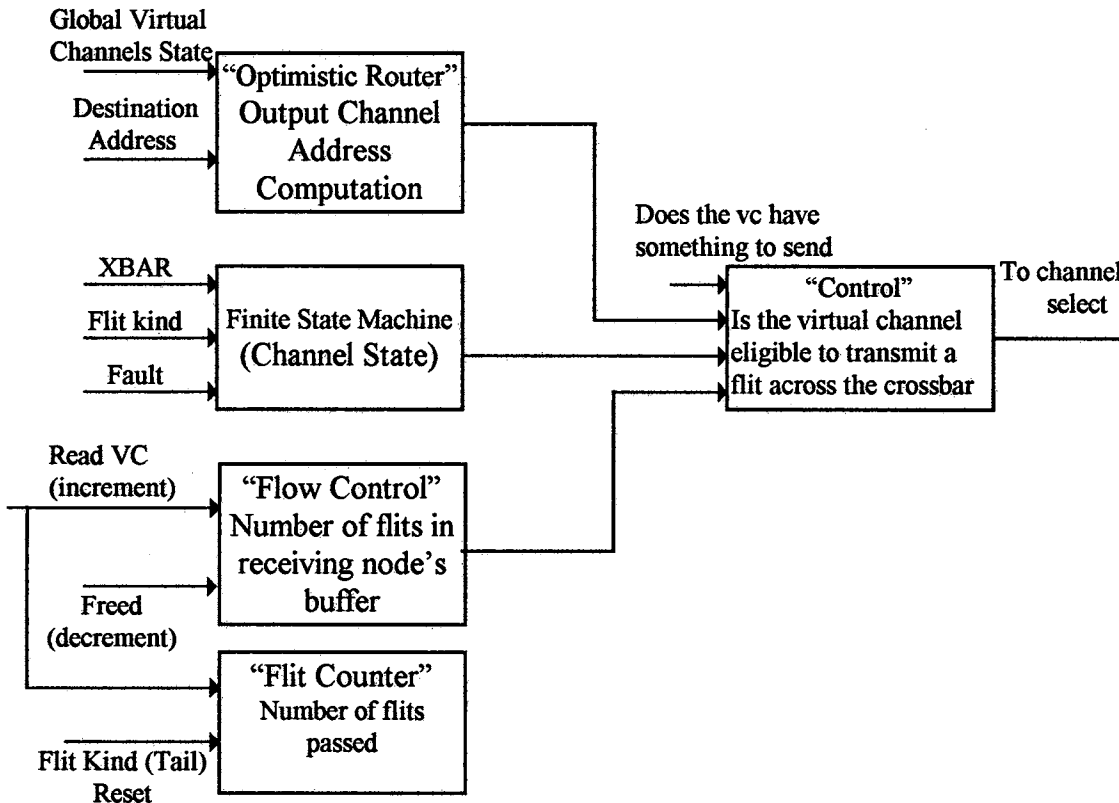


Figure 54 Virtual channel module for SBRP

In case node B goes down and corrupts a message pipeline, the actions taken by the nodes of Figure 51 are as follows:

Node A

Inserts a NFT flit at the end of the sub-message.

Node C

- Absorbs all non-head flits routed to the faulty node.
- Sends a retransmission request (containing message id, sub-message serial number and the sequential number of the last flit received) to the sender.

Sender node

After receiving retransmission request from node C, the sender performs the following tasks:

- Inserts a tail flit in the path of corrupted message in order to release the reserved channels.
- Changes the head by incrementing SSNF by one.
- Computes the FSN.
- Retransmits the remainder of the corrupted message.

FSN is computed as follows :

Assume that last flit passed the faulty node is m . Node C does not know the value of m and does not know how many flits were in the faulty node. Therefore, the sender must assume the maximum possible flits between node C and node A, $Max(C, A)$. Thus, the retransmission must start with a value less than or equal $m+1$. The sequential number of first flit, s , is obtained as

$$s = \text{Last flit passed node C} - \text{Max}(C, A).$$

It is clear that $s \leq m+1$. When $s < m+1$ duplicate flits will reach the receiver.

5.2.3 SBRP-2

In SBRP-1 and SBRP-0, the resend request is sent immediately after the fault occurrence. But in SBRP-2 the resend request is sent by the receiver when it gets the NFT. Once the receiver gets the NFT, it invokes the recover handler. The recover handler gets the message id and number of received flits and sends a recover message to the original sender. Unlike the SBRP-1 approach, SBRP-2 does not require any extra hardware cost.

In case node B goes down and corrupts a message pipeline, the actions taken by the nodes of Figure 51 are as follows:

Node A

Inserts a NFT at the end of the sub-message.

Node C

Absorbs all non-head flits routed to the faulty node.

Receiver node

After receiving a NFT, the receiver sends a retransmission request (containing message id, sub-message serial number and the sequential number of the last flit received) to the source.

Sender node

After receiving retransmission request from the receiver, the sender does the following:

- Inserts a tail flit in the path of corrupted message in order to release the reserved channels.
- Retransmits the remainder of the corrupted message after changing the head by incrementing the SSNF by one.

SBRP-2 does not handle the case when the message's head is corrupted. This special case can be handled as whole message lost in upper layers.

5.3 Software Overhead in SBRP

Recent advances in messaging implementation and improved network interface have reduced the software cost of messaging significantly[69]. Also, direct communication between the processor and the network interface saves instructions and bus transfers. For example, in the j-machine a single instruction, SEND, appends the contents of two processor registers to a message. Composing and injecting a four word message takes only 4 clock cycles in j-machine[9].

In SBRP, a retransmission request(message) is generated after a fault detection.

The overhead with generating a control message in SBRP includes the following :

- Getting the information about corrupted message from the router. This information includes VC flit counter and message's head. Router's instructions can be used to load these values from router registers to processor registers. Two load instructions are needed. The total cost of this step, L , equals $2 * \text{load's CPI}$. We assume that each load instruction cost one clock cycle.

- Fixed overhead for send (T_s).
- Injection overhead (T_w).
- Network latency (N).
- Fixed overhead for receive (T_r).
- Receive overhead (T_o).

Total overhead = $L + T_s + T_w + T_r + T_o + N$. Based on J-machine measurements[10], $T_s = 7$ cycles, $T_w = 1 * \text{Message Length}$, $T_r = 5$ cycles, $T_o = 7 * \lceil \text{Message Length} / 2 \rceil$. The length of control message is 4 words (two data words, head and tail).

$$\begin{aligned} \text{Total overhead} &= 2 + 7 + 4 + 5 + 14 + N \\ &= 32 + N \end{aligned}$$

The value of N will be provided by the simulator.

Based on the reported measured times for active message implementation [17], the software overhead was 100 simulator cycles which includes both the actual system software overhead and the network interface overhead[31].

5.4 Comparison between the three approaches of SBRP

Table 2 illustrates the different requirements of the three approaches of SBRP. Since SBRP-2 does not require any flit counter or head copy, it has the lowest requirement. Therefore, the current routers can be easily modified to implement SBRP-2. SBRP-0 does not require the flit counter but it requires the head copy and the sender must resend the whole message again. SBRP-1 requires the head copy and the flit counter but only part of the message need to be resend.

Requirements	SBRP-0	SBRP-1	SBRP-2
Flits counter	No	Yes	No
Head's copy in every node in the path	Yes	Yes	No
Resends whole message again	Yes	No	No
Resends only part of the message	No	Yes	Yes
Tolerates head loss	Yes	Yes	No

Table 2 Comparison between SPRP protocols

5.5 Comparison between SBRP and other recovery protocols

Table 3 summarize the differences between SBRP and current recovery protocols. The significant advantage of SBRP over the current recovery protocols is the recovery overhead is attached only with the corrupted messages. All the recovery protocols including SBRP require additional hardware. The compressionless protocol requires special control lines and requires also a special router design to enable the router to resend the messages again in case of message corruption. The Reliable Router (RR) , SBRP-0 and SPRP-1 requires a copy of the message head to be saved in all the nodes in the message path. Moreover, SPRP-1 requires a flit counter for every virtual channel.

Although the SBRP does not incur overhead in fault-free environment, it needs help from Network Interface. SBRP requires the Network Interface (NI) to invoke the Recovery handler once the fault happens. Also, SBRP assumes the NI has the ability to latch the values of the router's registers. Resending the message again requires also NI help.

Requirements	SBRP	Reliable Router	Compressionless
Attaches overhead with every message	No	Yes	Yes
Requires control messages	Yes	No	No
Requires control lines	No	Yes	Yes
Handles multiple faults	Yes	No	Yes
Requires software fault handler	Yes	No	No
Keeps the message in the sender's buffer until delivery is insured	Yes	No	Yes
Retransmits whole message	No	No	Yes

Table 3 Comparison between SBRP and other protocols

Chapter 6

SIMULATION AND PERFORMANCE ANALYSIS

We designed a simulator to implement wormhole interconnection networks. The purpose of the simulator is to study the performance of wormhole networks, analyze the effect of network traffic and the impact of design parameters, and to evaluate the performance of the proposed recovery protocol. Developing an analytical model for performance evaluation of wormhole routing is difficult because of the multiple and simultaneous resource possessions as well as the blocking during the pipelined routing. Moreover, adaptive routers have complex behavior which depends on network status. Due to the difficulty of accurate analytic modeling, router performance is evaluated via simulation [67][44].

The router model used in the simulation is shown in Figure 55. The design configuration and assumptions are described in the next sub section. Number of input inject buffers is equal to number of virtual channels per physical channel. Each virtual channel requires a counter to record the number of flits in the corresponding input buffer on the receiving node. When a positive (negative) acknowledgment arrives for a virtual channel, the counter that corresponds to it is incremented (decremented). In this thesis we consider hypercube and 2-d mesh topologies even though the proposed protocol is applicable to all other topologies.

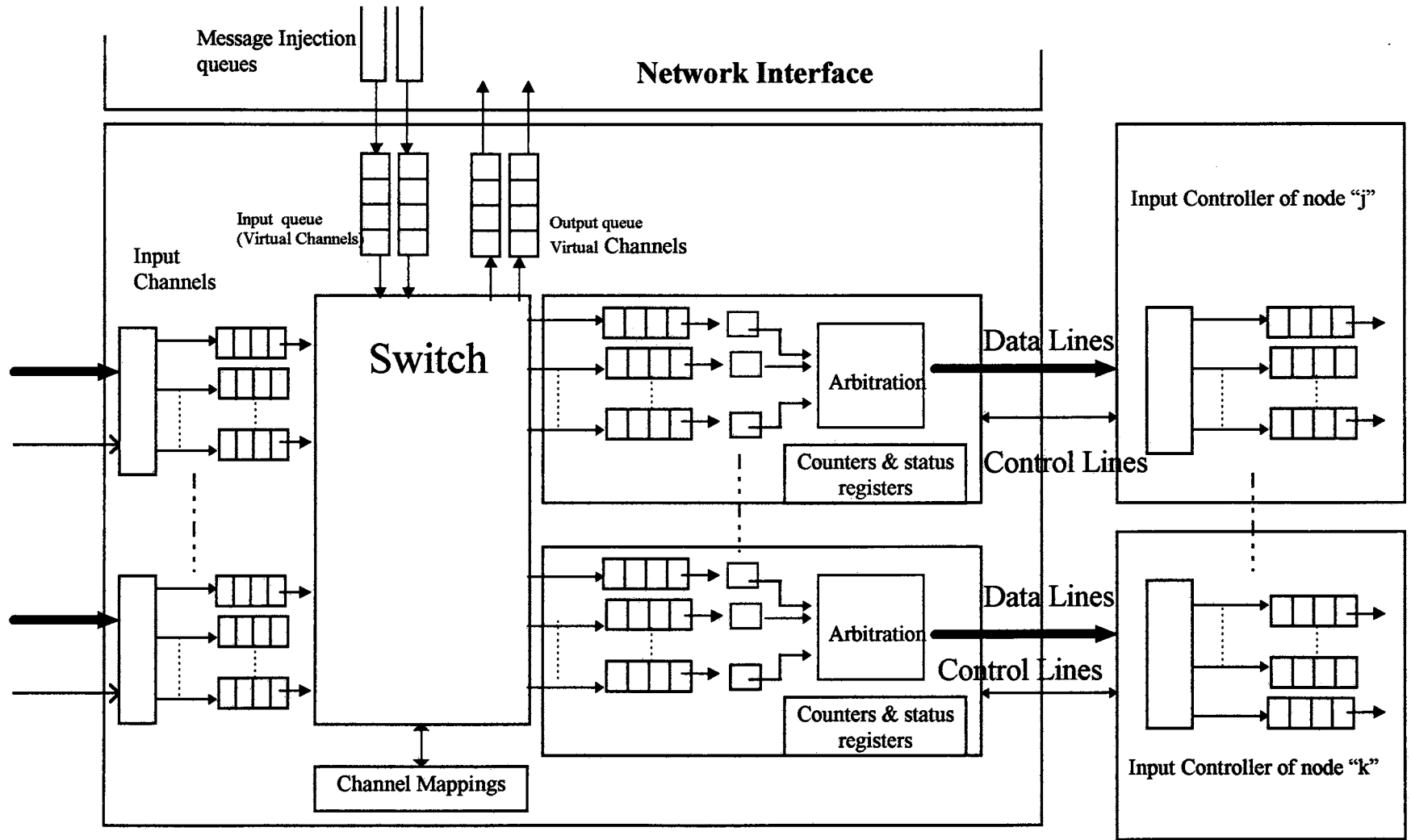


Figure 55 Simulated Router Model

6.1 Simulation Configuration

The following assumptions are made in the simulation:

- 1- A node can generate messages of fixed length destined for any other node. Traffic is uniformly-distributed unless otherwise specified.
- 2- A message arriving at its destination node is eventually consumed.
- 3- Wormhole routing is used. So, once a queue accepts the first flit of a message, it must accept the remainder of the message before accepting any flits from another message. Also, a message may occupy several channels simultaneously.
- 4- The crossbar switch in the router allows multiple messages to traverse a node simultaneously without interference. It takes one clock cycle to transfer a flit from an input buffer to an output buffer regardless of router's design parameters.
- 5- There are two separate unidirectional channels between any two adjacent nodes.
- 6- Each virtual channel uses one input buffer. Each buffer can store only a single flit.
- 7- Multiple messages can be sent simultaneously between a processor and the corresponding router.
- 8- All flits reaching a destination are consumed simultaneously.
- 9- Transferring a flit between two nodes via a physical channel takes one unit of time.
- 10- A round robin scheduling policy is used to arbitrate the multiple request in physical links. Virtual channels, who are ready to send a flit, are the only ones who compete to get access to the physical channel. Blocking messages and messages waiting to be routed do not consume any channel bandwidth.

11- The route taken by a message depends on its destination and the status of output channels. If all the alternative output channels are busy the message header is not required to wait on any predetermined channel. Instead, it waits on the input channel buffer. It is repeatedly routed until it is able to reserve a channel, thus getting the first channel that becomes free.

12- Each node knows its neighbors' state (i.e. faulty or not faulty). Also, there is no message from or to the faulty node.

13- If a node is a faulty node, all its input and output physical channels are marked as faulty. Each physical channel has a fault status bit.

6.2 Performance Metrics

The most important performance metrics of an interconnection network are latency and throughput. Latency is the time elapsed since the message transmission is initiated until the message is received at the destination node. Hardware latency is defined as the time elapsed since the message header is injected into the network at the source node until the last unit of information is received at the destination node. The queuing time at the source node is added to the latency. When the messaging layer is being considered, latency is defined as the time elapsed since the system call to send a message is initiated at the source node until the system call to receive the message returns control to the user program at the destination node. In this thesis, we consider the hardware latency for all messages except for recovery messages as will be explained in section 6.3. Latency is measured in time units. We will use the simulator clock cycle as the unit of measurement.

Throughput is the maximum amount of information delivered per time unit. The normalized throughput is equal to the number of messages received over the number of messages that can be transmitted at the maximum load. The maximum load is derived by considering the fact that 50% of uniform random traffic crosses the bisection plane of the network[14]. Thus, if a network has a *bisection* bandwidth B flits/cycle, it can inject a maximum of $2B$ flits/cycle. An optimal routing algorithm can handle such a load without *saturation*. The bisection width B is equal to 2^n and $2n$ for n -dimensional hypercube and $n \times n$ meshes, respectively.

The evaluation of interconnection networks requires the definition of representative work models. The work model is basically defined by three parameters: distribution of destination (traffic pattern), injection rate and message length.

The distribution of destination indicates the destination for the next message at each node. The most frequently used distribution is the uniform one. In this distribution, the probability of node k sending a message to node j is the same for all k and j . The uniform distribution makes no assumption about the type of computation generating the messages. In the study of interconnection networks, it is the most frequently used distribution. There are other kinds of destination distributions like bit-reversal, perfect-shuffle, butterfly, matrix transpose and complement [36].

For synthetic workloads, the injection rate is usually the same for all the nodes. In most cases, each node is chosen to generate messages according to uniform distribution within an interval. Other possible distributions include bursty traffic and trace from parallel applications. The *normalized load* is equal to the number of messages generated over the number of messages that can be transmitted at the maximum load.

Message length can also be modeled in different ways. In most simulation runs, message length is chosen to be fixed [36]. Also, message length can be computed according to a normal distribution or a uniform distribution within an interval. The selected message length distribution should be representative of intended applications, if there is any. Obviously, application traces should be used if available. In this thesis, we use fixed message length. However, message length varied from one run to another in order to study the effect of message length.

6.2 Fault - Free Performance

In this section, we analyze the performance of wormhole networks on hypercubes and two dimensional meshes under different design parameters and different traffic conditions. The effect of adding more virtual channels and using adaptive algorithms are evaluated. Also, We evaluate the performance under different traffic patterns. All the graphs in this chapter show the relation between the input load and latency and throughput. The input load can be measured as the number of messages or flits injected in a unit of time or can be measured as a fraction of maximum load.

The latency curve generally takes the shape shown in Figure 56. Latency increases slowly first and then more rapidly, approaching a vertical asymptote at network saturation. Beyond a saturation message rate, λ_{sat} no steady state solution for network latency exists. At higher message rates, latency grows without bound[19].

Beyond the saturation point, $\lambda > \lambda_{\text{sat}}$, network performance is described by the rate of accept traffic (throughput). Figure 57 shows a plot of throughput versus normalized

load. Below the saturation point, the curve is the straight line $\lambda_a = \lambda$. Beyond the saturation point, the curve has a constant value, $\lambda_a = \lambda_{sat}$.

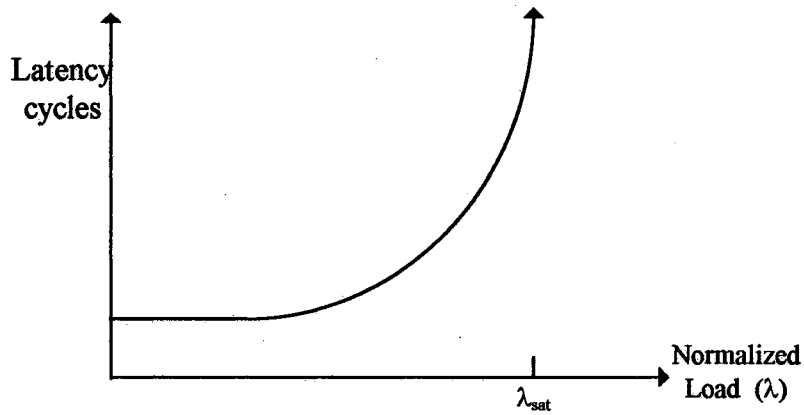


Figure 56 Latency versus normalized load (Adopted from [19])

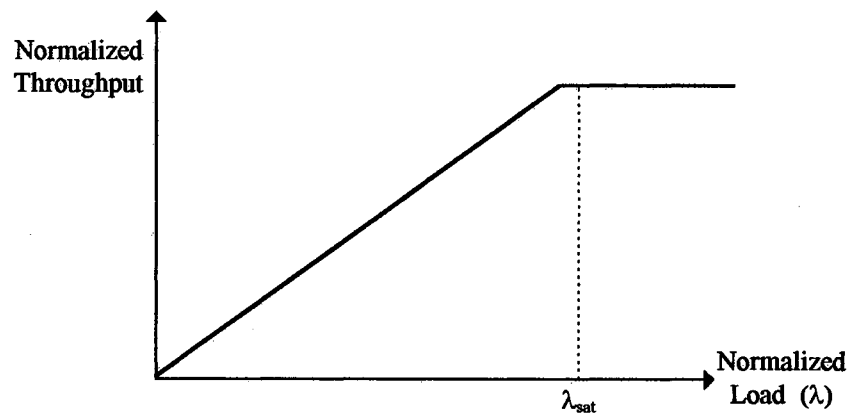


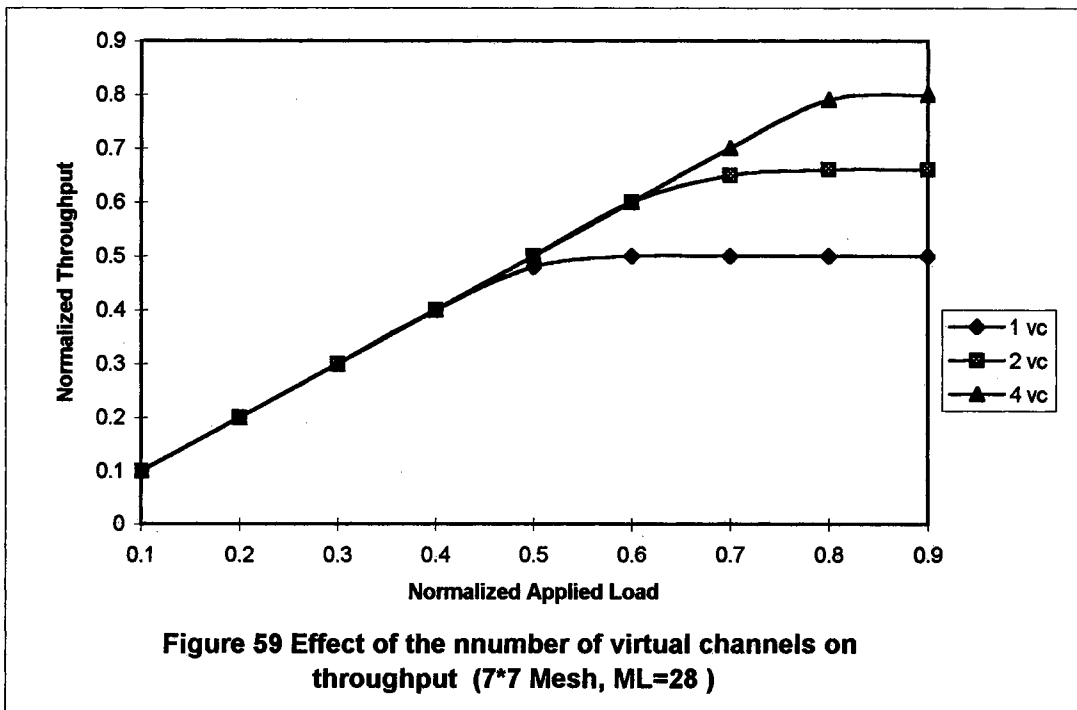
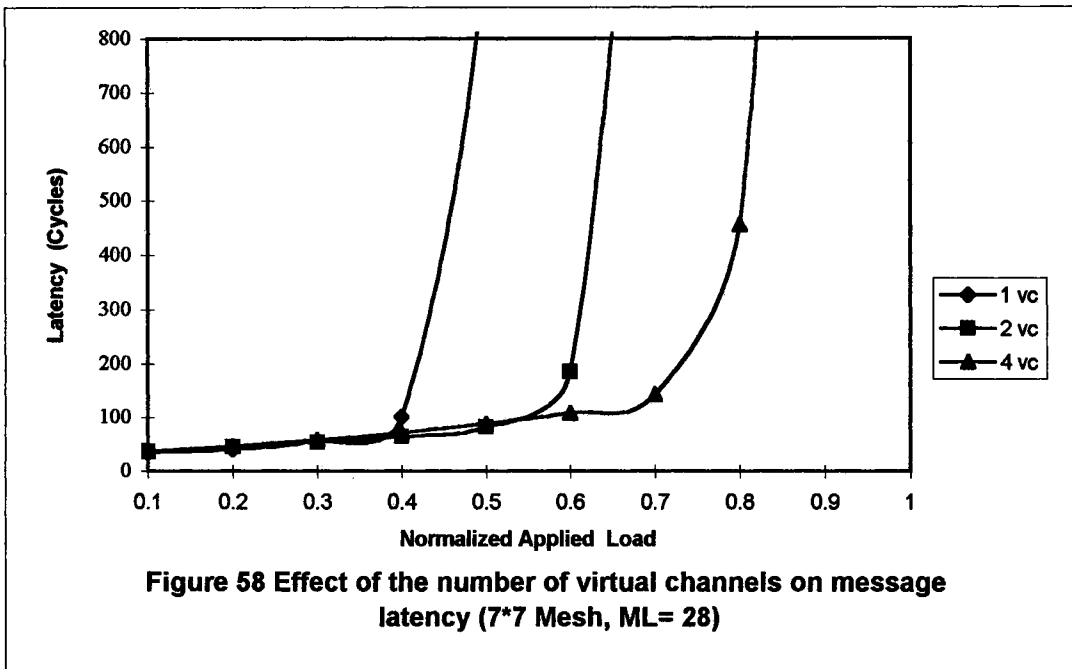
Figure 57 Normalized throughput versus normalized load (Adopted from [19])

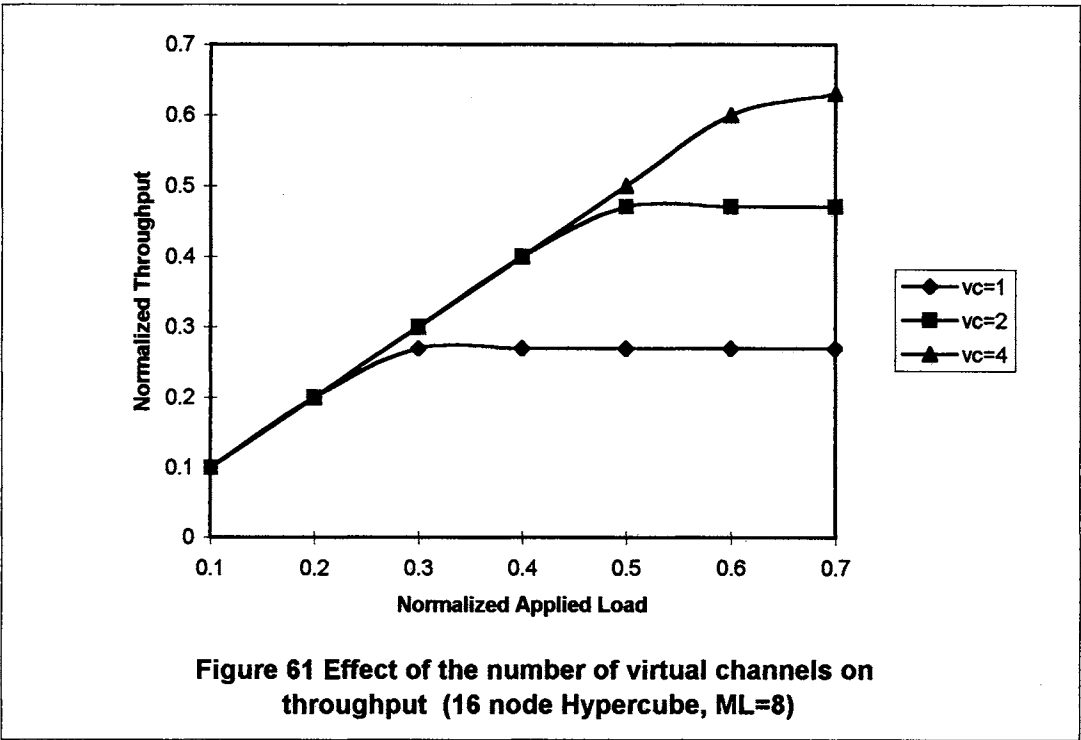
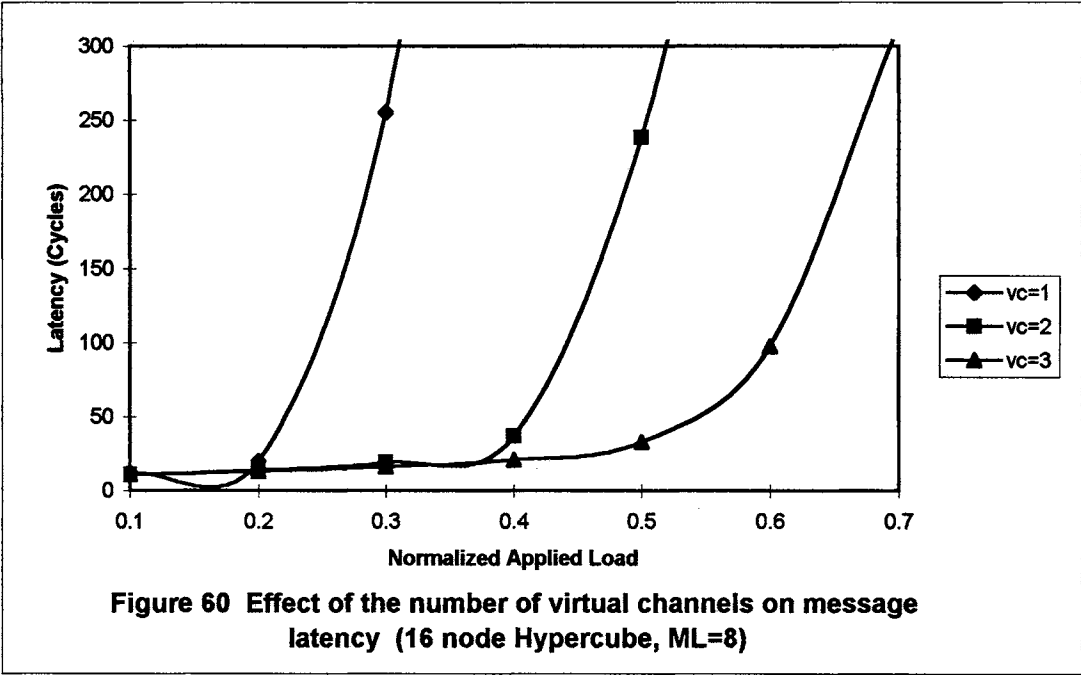
6.2.1 Virtual Channels

Figures 58, 59, 60 and 61 show the effect of the number of virtual channels in networks performance. Figures 58 and 59 show the relation between normalized input load and message latency and throughput, respectively, in a 7×7 mesh. Figures 60 and 61 show the same for 16 nodes hypercube. The plots show the performance enhanced as the number of virtual channels increase. As we have mentioned in chapter 2, virtual channels increase throughput by multiplexing more than one packet over one physical channel. However, studies in [72] show that adding virtual channels complicates router design. By adding one virtual channel, the time needed to route the flits through the router increases by 30%.

6.2.3 Routing Algorithms

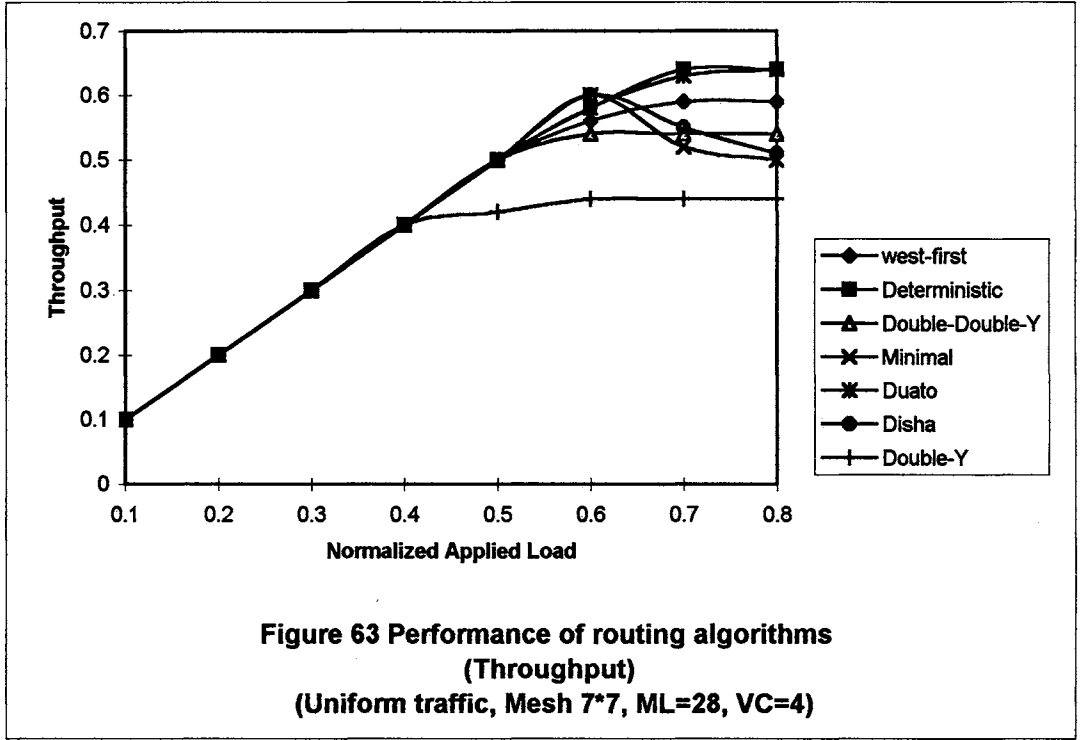
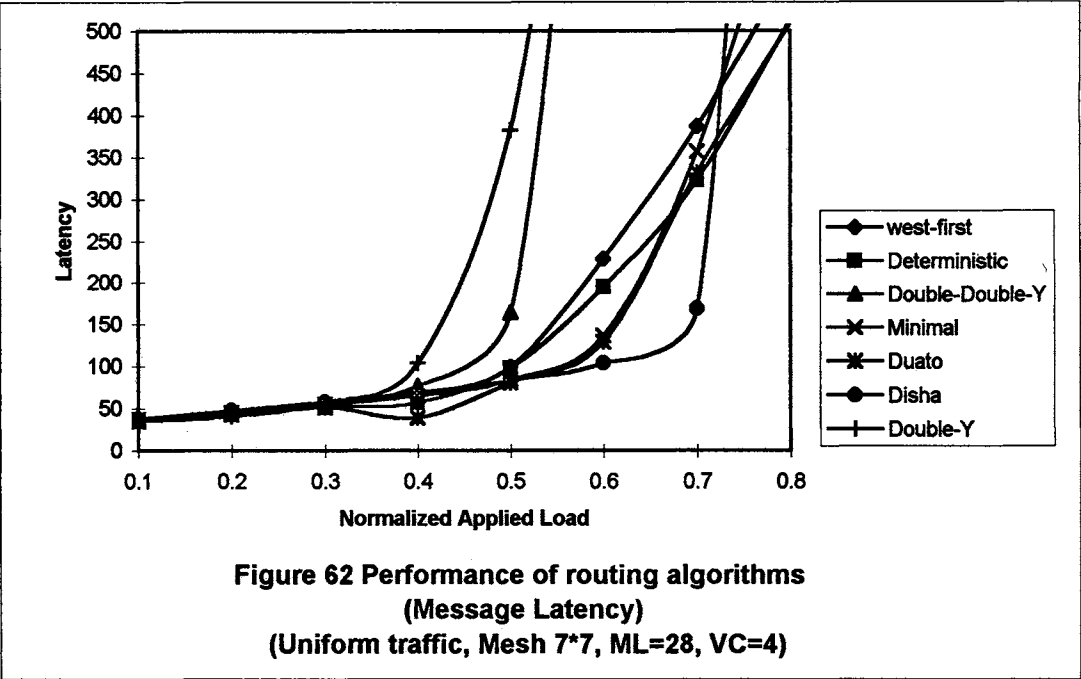
As we have discussed in chapter 3, adaptive routing algorithms utilize the network bandwidth more efficiently than deterministic routing algorithms and give more flexibility to avoid faulty or congested regions. In this section, we investigate the performance of different routing algorithms using our simulator. For two dimensional meshes we simulate several routing algorithms. For the deterministic routing, the XY algorithm is simulated. For the partial adaptive routing, the West-First routing algorithm is simulated. For the fully adaptive routing, Double-Y, Duato and Disha routing algorithms are implemented. We also simulated the true fully adaptive routing algorithm (TFAR) without any mechanism for deadlock avoidance or recovery. For all algorithms, we use two virtual channels.





As we mentioned in section 3.3.2 Double-Y uses only two virtual channels in y direction and one virtual channel in x direction. To utilize all the virtual channels, we simulated two independent subnetworks using three virtual channels. The first subnetwork uses two virtual channels in y direction and one virtual channel in x direction. The second subnetwork uses two virtual channels in x direction and one virtual channel in y direction. The packets are directed randomly to one of the two subnetworks. Once a packet is directed to a subnetwork it can not switch to the other subnetwork. We called this implementation Double-Double-Y. In [44], the same concept is used but for fault tolerance where the second subnetwork is used only to avoid faulty regions.

Figures 62 and 63 show the performance of the above algorithms under uniform traffic. From the plots we note that the deterministic algorithm gives equal performance to the partial adaptive algorithms' performance under uniform traffic in mesh. The reason is that with uniform traffic, the load is uniformly distributed. Also from the plots we note that the deterministic algorithm gives better performance than the partial adaptive algorithm's performance (West-First) under uniform traffic in mesh. That is because West-First causes more traffic congestion in one part of network and hence lead to early saturation[74]. Double-Y gives the worst performance because it does not utilize the second virtual channel in x direction. Moreover, Double-Y does not provide flexibility in virtual channel selection where each message is assigned a specific virtual channel depending on its direction. Although Double-Double-Y utilizes all virtual channels, it does not show good performance, because each packet must select a specific channel and must wait if it is busy even though all other virtual channels in same direction are free. TFAR gives good performance under light loads. However, under high loads the deadlocks

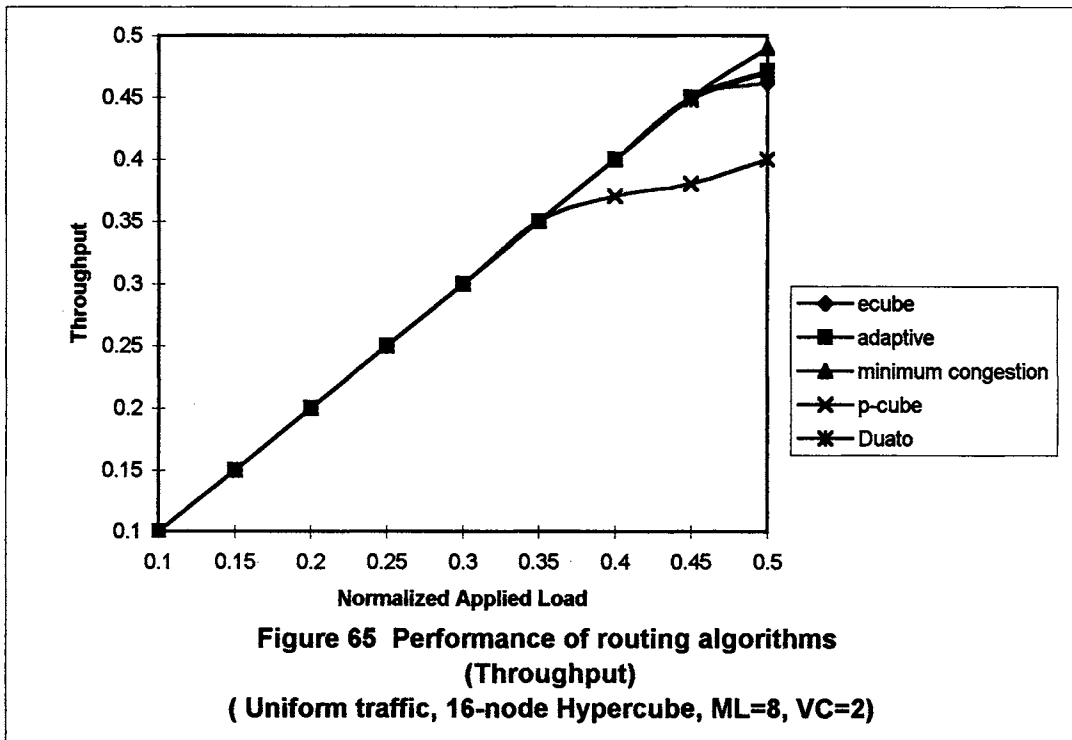
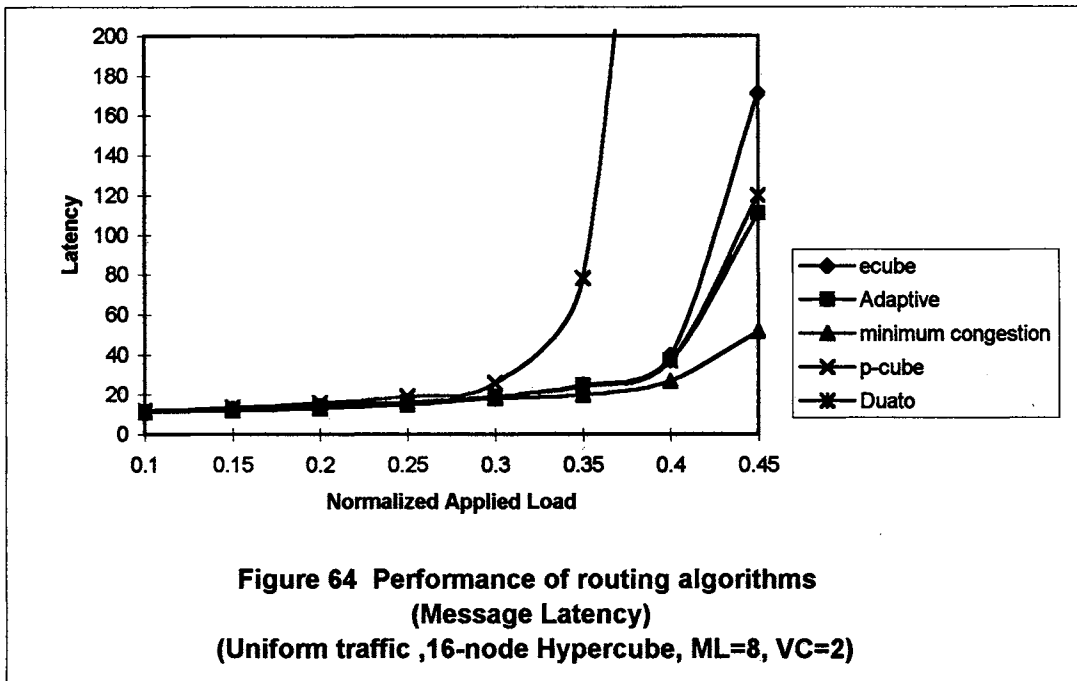


deteriorate the performance. Disha algorithm which uses deadlock recovery mechanism shows good performance. However, with high number of deadlocks in high load it could not handle all deadlocks and hence causes performance deterioration.

For hypercube, Figures 64 and 65 show the performance of a deterministic routing algorithm (ecube), the partial adaptive algorithm (p-cube), Duto's algorithm, the fully minimal adaptive algorithm and the Minimum-congested algorithm. In the fully minimal adaptive algorithm, the packet goes in a direction which might bring it closer to the destination. To take care of possible deadlocks in minimal adaptive algorithm, a deadlock recovery algorithm is used as in [12]. In Minimum-congestion algorithm, the routing algorithm selects the least loaded link between all possible minimal paths. From the graphs we note that the performances of deterministic and adaptive algorithms are almost the same under uniform traffic. However, when the network is near the saturation level the adaptive algorithm gives better performance because it avoids blocked links by selecting another physical link, if any. The Minimum congested algorithm shows the best performance in high load rate because it selects the least congested links which almost makes all links' load equal.

6.2.2 Traffic Patterns

In this section, we investigate the effect of traffic patterns (distribution of destination) in routing algorithms' performance. The traffic patterns take into account the transformation that are usually performed in parallel algorithms. In some parallel algorithms, there is no specific traffic pattern. However, other parallel algorithms (especially parallel numerical algorithms) show a specific pattern where the destination



node for the message generated by a given node is always the same. To evaluate both cases, we use two different traffic loads which are described below.

Random (Uniform): Each node sends messages with equal probability to all other nodes. The uniform distribution makes no assumption about the type of computation generating the messages. The uniform distribution provides an upper bound on the mean internode distance because most computations exhibit some degree of communication locality [36].

Dimension-reversal (Transpose): Each node sends messages to the node with address of reversed dimension index. In two dimensional networks, node (x,y) communicates with node (y, x) . In hypercube, the binary address of the node is split into halves, and these halves are swapped (if the dimension of the hypercube is odd, then the middle bit remains unchanged). For example, the node with binary address $(11100)_2$ communicates only with the node whose binary address is $(00111)_2$.

Figures 66 and 67 show the performance of three different routing algorithms in two-dimensional mesh wormhole network under the dimension-reversed (transpose) traffic pattern. Figures 68 and 69 show the performance of three routing algorithms in hypercube wormhole network under the transpose traffic pattern. In 2-d mesh, under the dimension-reversed (transpose) traffic pattern, the fully adaptive routing and Duato algorithms have the best performance. Dimension-reversal creates significant congestion under dimension-order (XY) routing in 2-d mesh. In hypercube, under transpose traffic pattern, the deterministic algorithm (ecube) shows very low performance. Also, dimension-reversal creates significant congestion under dimension-order (ecube) routing in hypercube.

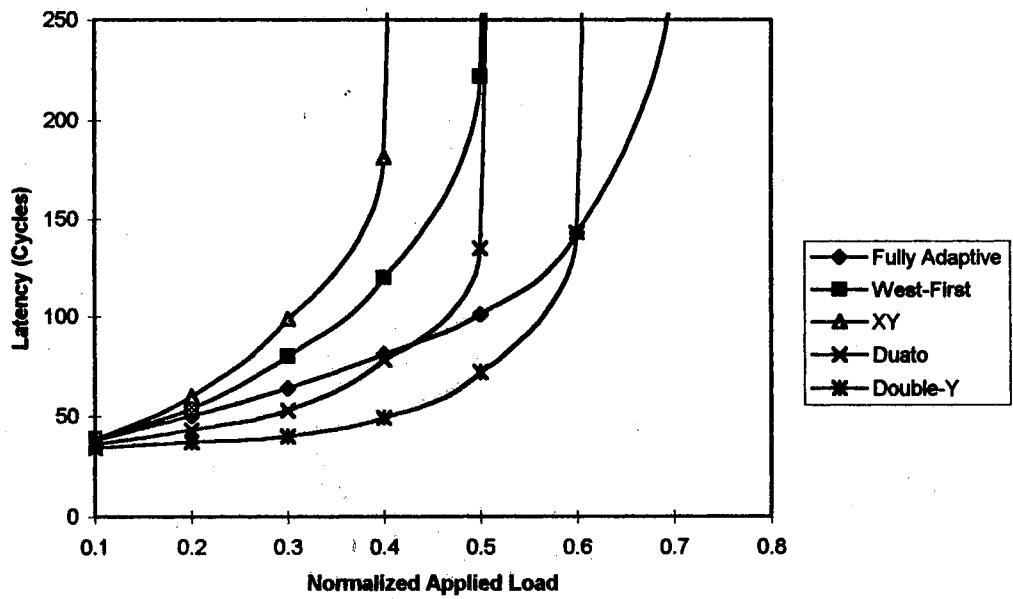


Figure 66 Effect of routing algorithms on message latency (7*7 mesh, ML=28, VC=2, Transpose)

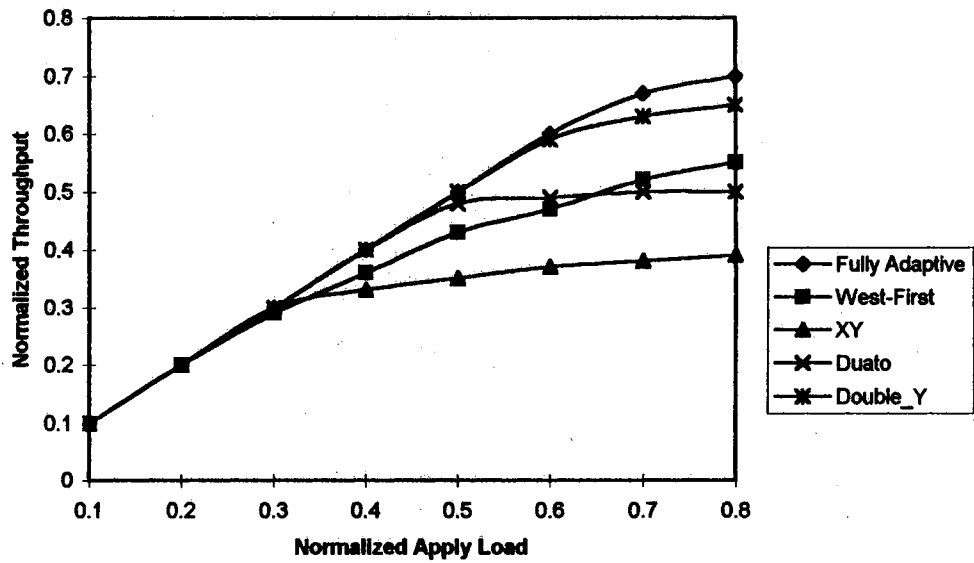
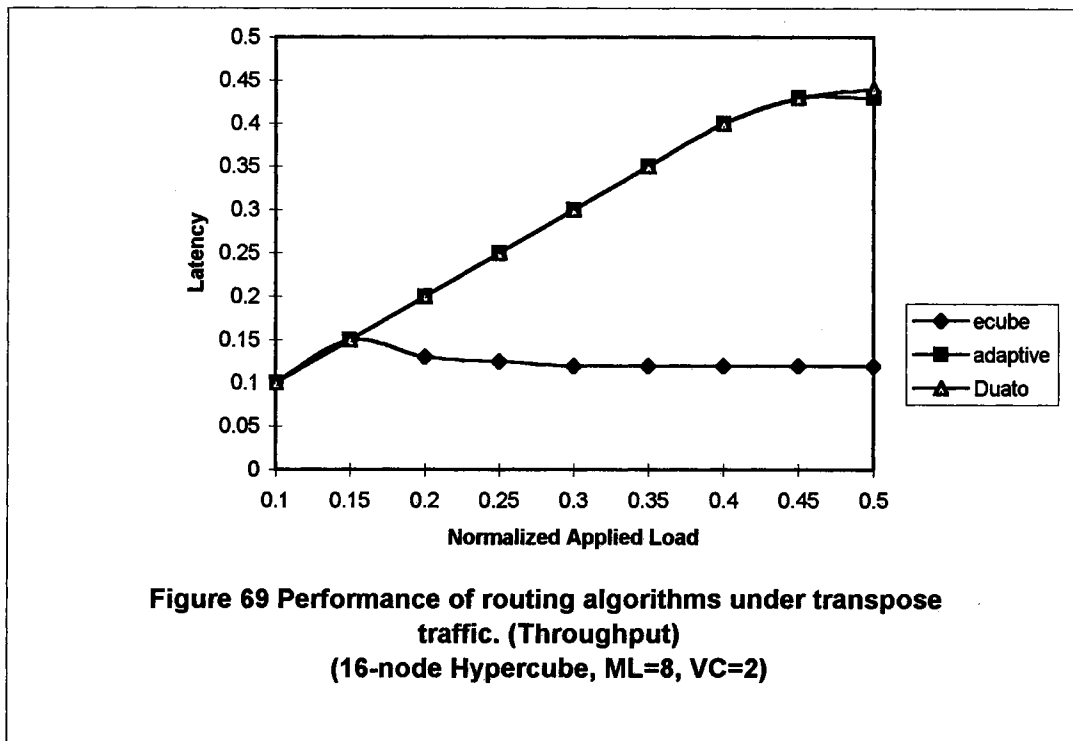
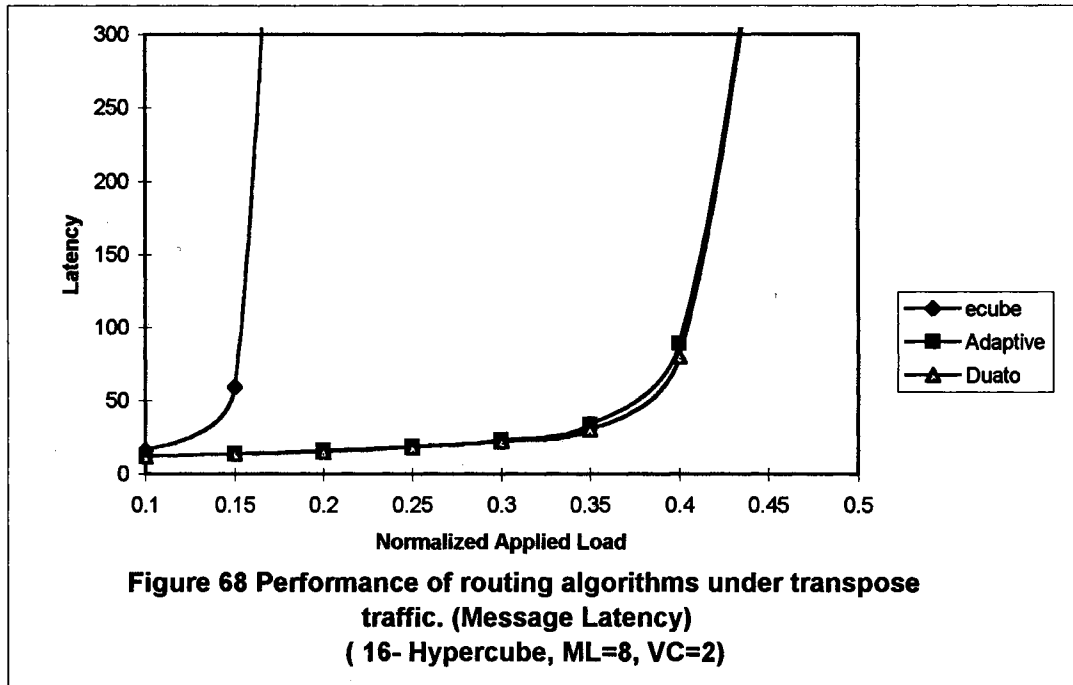
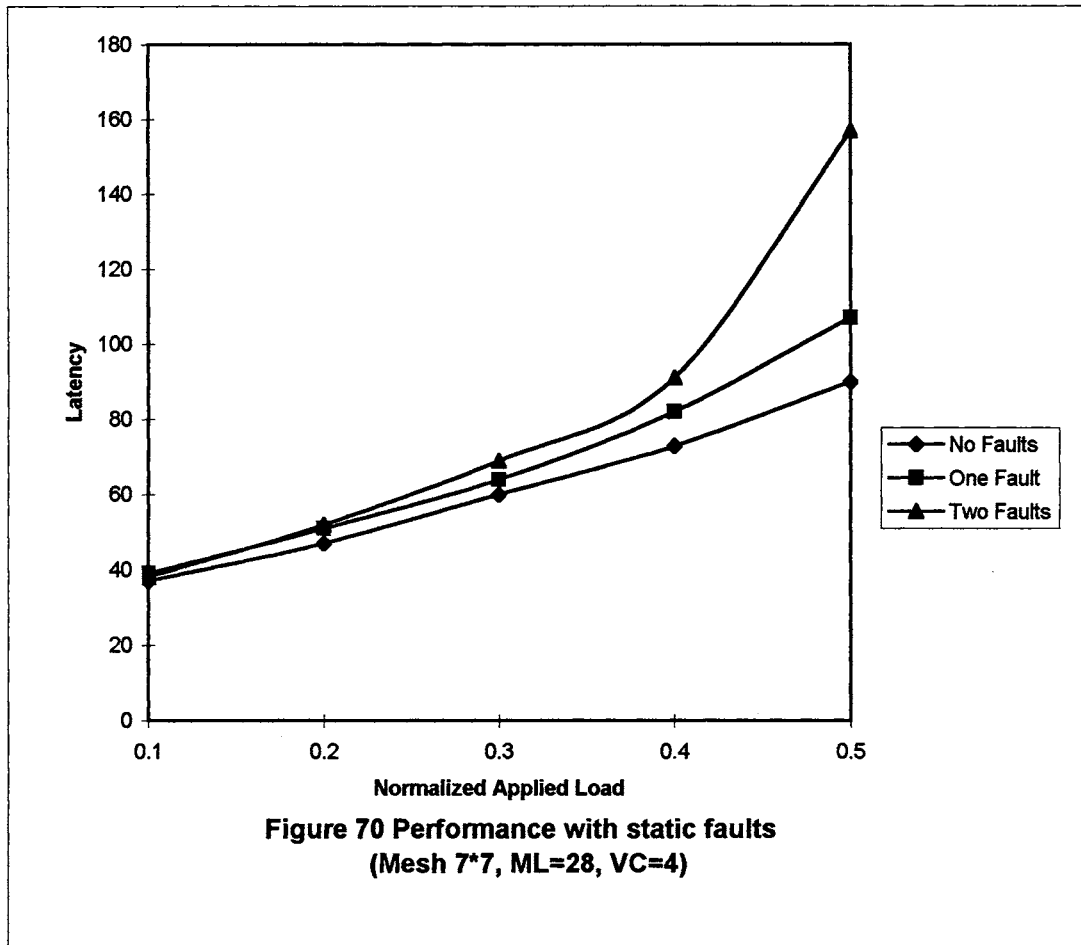


Figure 67 Effect of routing algorithms on throughput (7*7 mesh, ML=28, VC=2, Transpose)



6.3 Static Fault Performance

The existence of faults in network causes performance deterioration because it decreases routing freedom and makes some links congested. Figure 70 shows the performance of 49-nodes 2-d mesh with one fault and two faults. Fault free performance is shown as a reference.



6.4 Fault Recovery Performance

In this section, we compare the effect of different recovery mechanisms on network performance. we compare software based recovery protocol (SBRP-1) with compressionless protocol[2] and reliable router scheme[1]. As we have mentioned in chapter 4, the reliable router scheme doubles the message size to achieve fault recovery (see Figure 39). The compressionless protocol enlarges the original message length by padding extra flit at the end of every message. In compressionless protocol, the final message length is computed as follows:

$$\text{Message Length} = B_{\text{cap}} * D + L + 2 * M_{\text{mis}} * B_{\text{cap}}$$

where D: Distance to destination in hopes

L: Original message length.

B_{cap} : Depth of channel buffer ($B_{\text{cap}} = 1$ in our simulation)

M_{mis} : Maximum number of misrouted allowed.

Because we use the fault tolerant algorithm explained in section 3.4, we might need to misroute once to go around a faulty node. Therefore, M_{mis} equals 1 for hypercube and mesh. For a 12-flit message which needs five hopes, the new length is $5 + 12 + 2 = 19$ flits.

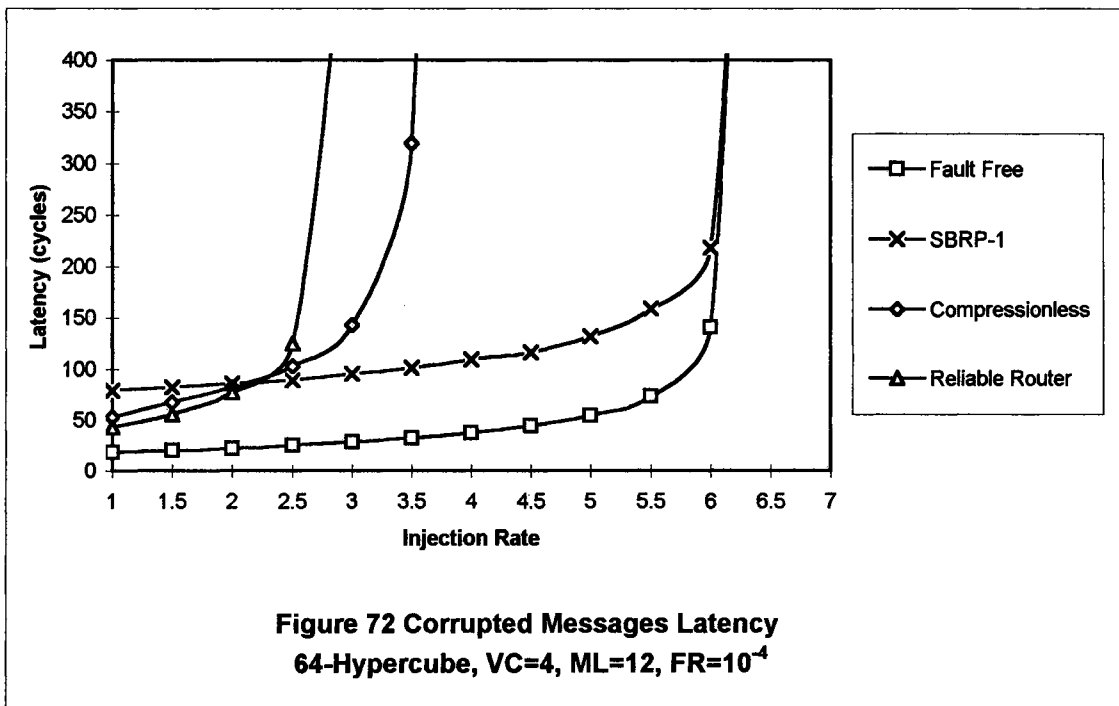
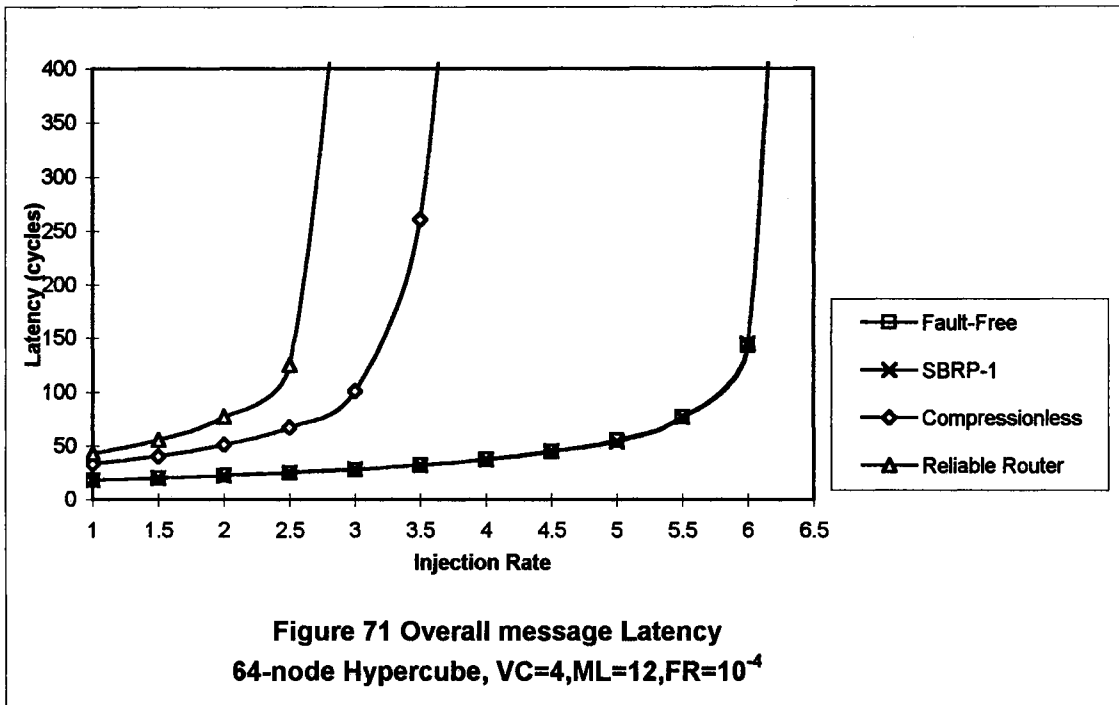
All simulations are run on 4 virtual channels(VC). We present our result with different message lengths (ML) to study the effect of message length on protocols' performance. We use transit fault model to study the performance of software based recovery protocol. Different fault rates (10^{-4} , 10^{-5} and 10^{-6} fault/node/cycle) have been

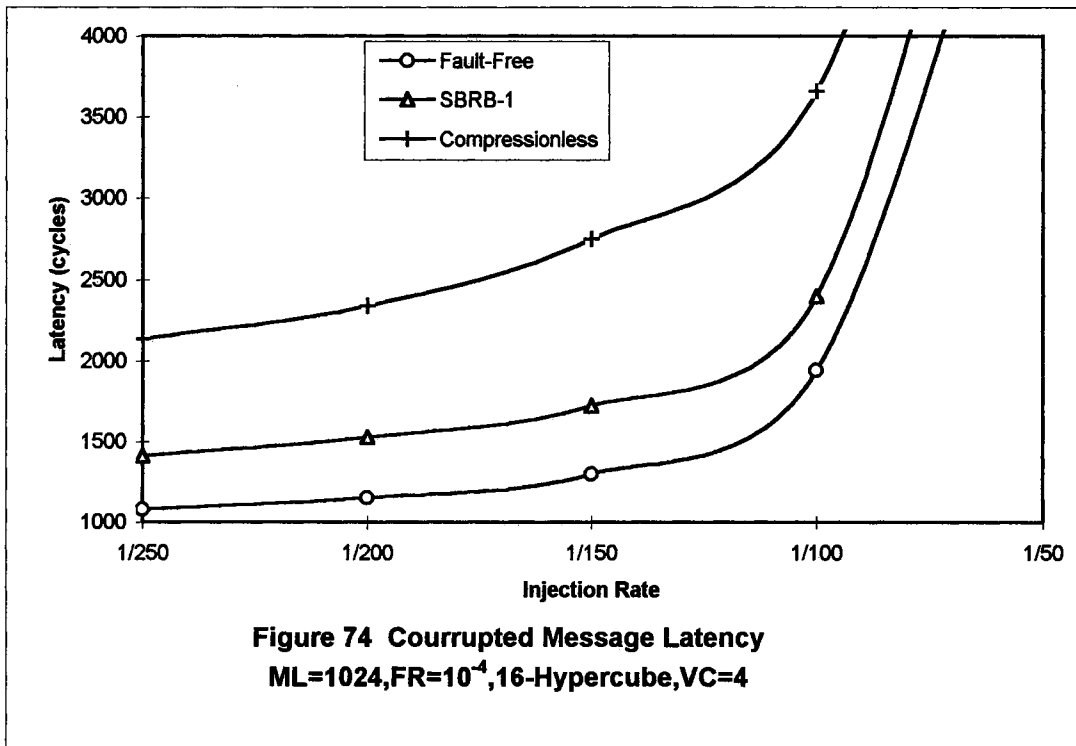
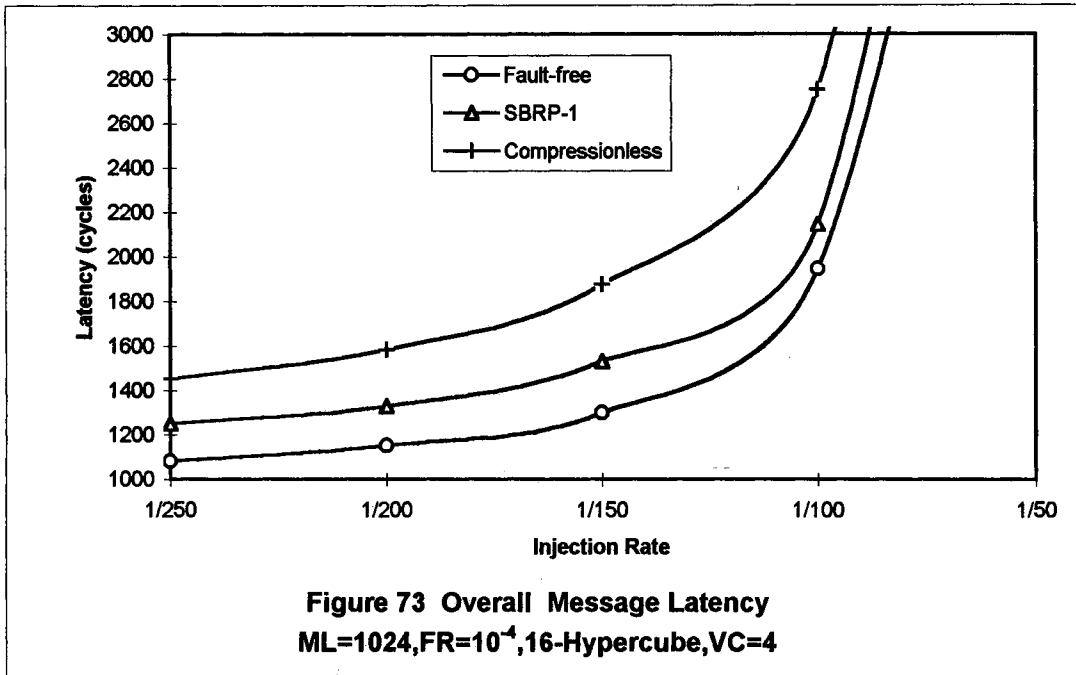
used during the simulation. The fault rate of 10^{-4} fault/node/cycle is considered very high because in 16 nodes multicomputer it means a fault happens every 625 clock cycles. For reference, the figures also show the performance of a fault-free network.

Figure 71 compares the message latency in SBRP with compressionless protocol[2] and reliable router scheme[1]. From Figures 71 and 72 , it is clear that the SBRP outperforms all other protocols. The overall message latency of SBRP is almost identical to the message latency of fault-free environment, because SBRP does not alter t

he latency of uncorrupted messages. Moreover, the latency of corrupted messages does not effect the overall latency because the number of corrupted messages is small compared to the total number of messages. The latency graphs of the corrupted messages only are shown in figure 72. The Reliable Router and Compressionless protocols give better performance than SBRP in low injection rate because of the high cost of software overhead. However, in high injection rate, SBRP shows better performance because the cost of enlarging message length in Reliable Router and compressionless protocols is larger than the cost of software overhead in SBRP. Results in Figures 71 and 72 are obtained in 64-node hypercube with 12 flits message length and 10^{-4} Fault Rate(FR).

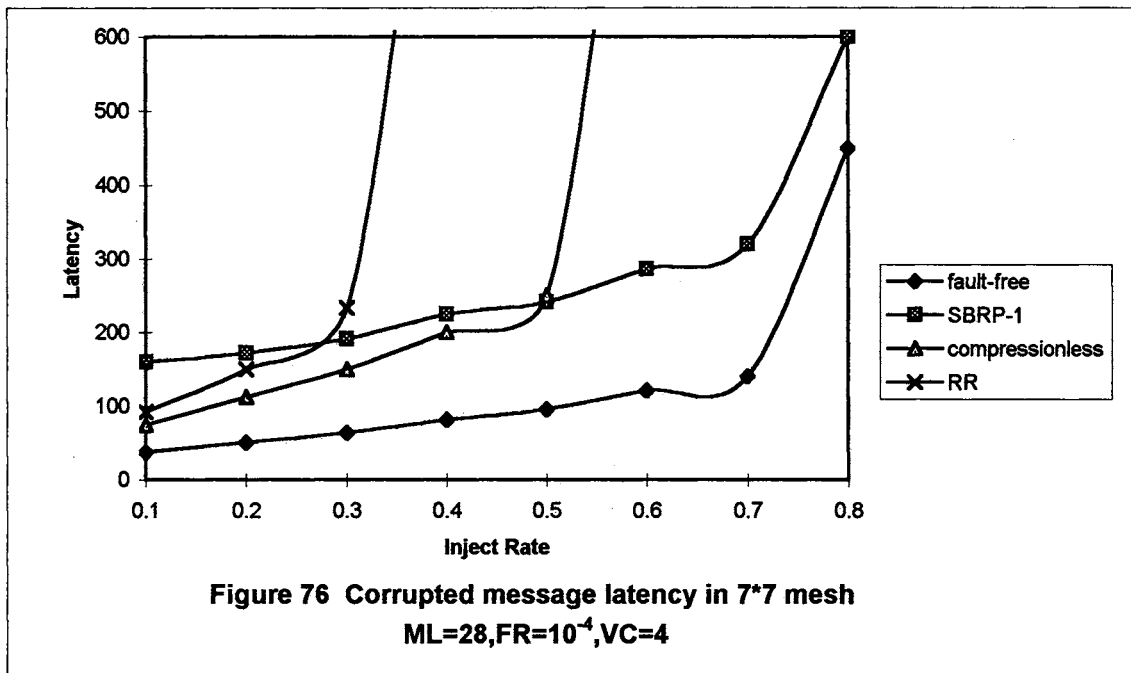
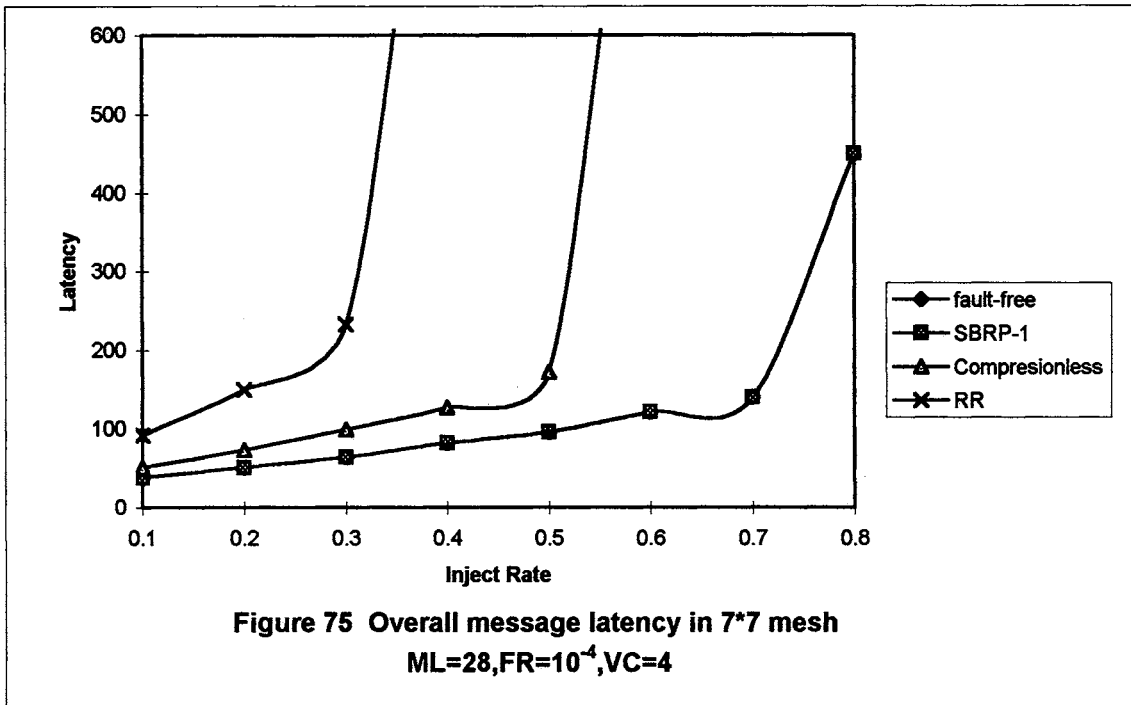
To show the performance of long messages, Figures 73 and 74 show the latency of overall messages and corrupted messages, respectively, for 1024 flit messages. SBRP gives better performance in long message environments too. However, the overhead caused by padding extra flits decreases in compressionless protocol. Figure 74 shows that the effect of not transmitting whole message is in long messages environment. Figures 75 and 76 show the latency of overall messages and corrupted messages for 28 flit messages in 49-node 2d-mesh.

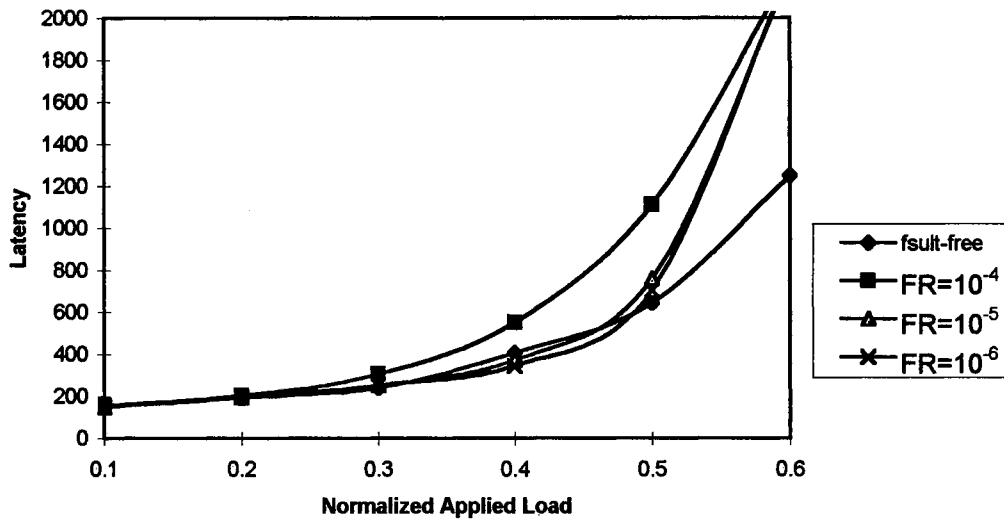




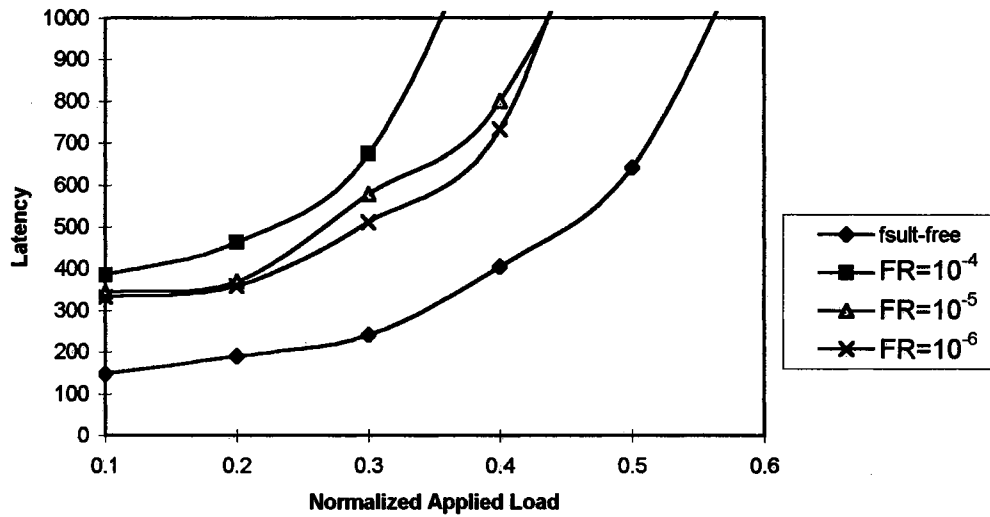
Also, we investigate the effect of different fault rates. When the fault rate increases, the number of corrupted messages is increased. The latency of corrupted messages is high due to the software overhead. Consequently, when we have more corrupted messages we expect to have worse performance. Figures 77 and 78 show the latency of overall messages and the latency of corrupted messages in SBRP-0 under three different fault rates 10^{-4} , 10^{-5} and 10^{-6} (fault/node/cycle). Fault-free latency is also shown as a reference. As the Figure 77 illustrates, messages in 10^{-4} fault environment have the highest latency.

Finally, we compare the performance of SBRP-0, SBRP-1 and SBRP-2. Figures 79 and 80 show the latency of overall message and the latency of corrupted messages, respectively, for short messages (8-flit messages). Figures 81 and 82 show the same for long messages (128-flit messages). SBRP-1 has the best performance because the resend request is initiated immediately after the fault occurs and the original sender resends only part of the corrupted message. SBRP-2 outperforms SBRP-0 in long message environment because it does not resend the whole message as SBRP-0 does. However, in short message environment, SBRP-0 gives better performance because the whole message resend overhead is small. As in SBRP-1, SBRP send the resend request directly after the fault happens. On the other hand, the resend request is sent by the receiver when it gets the (NFT).

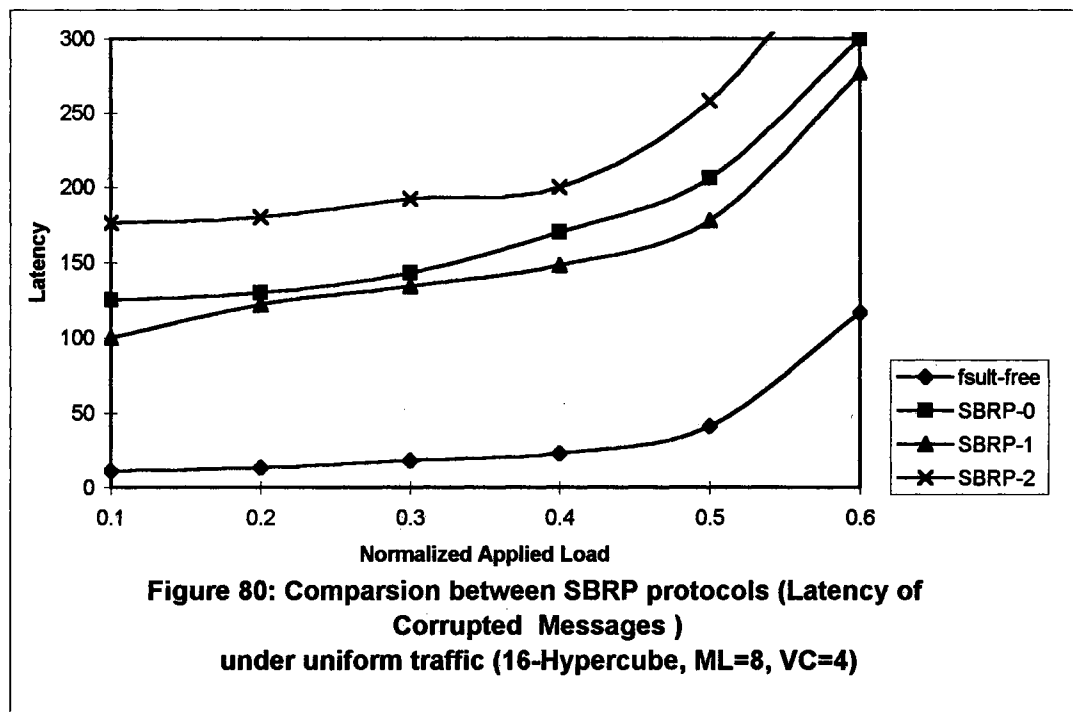
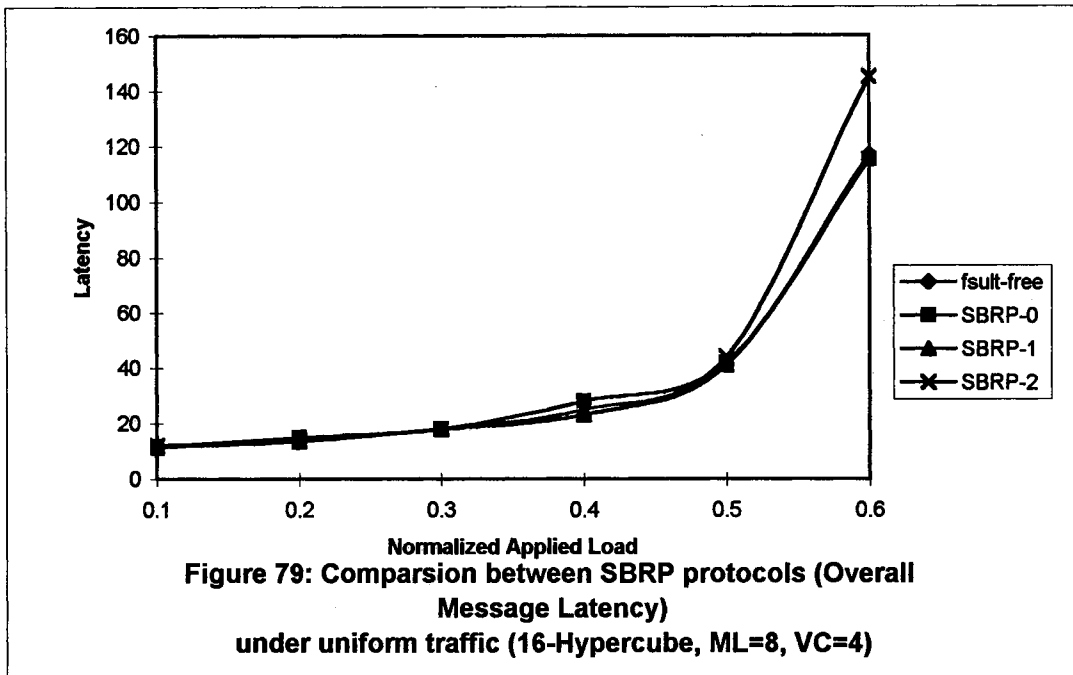


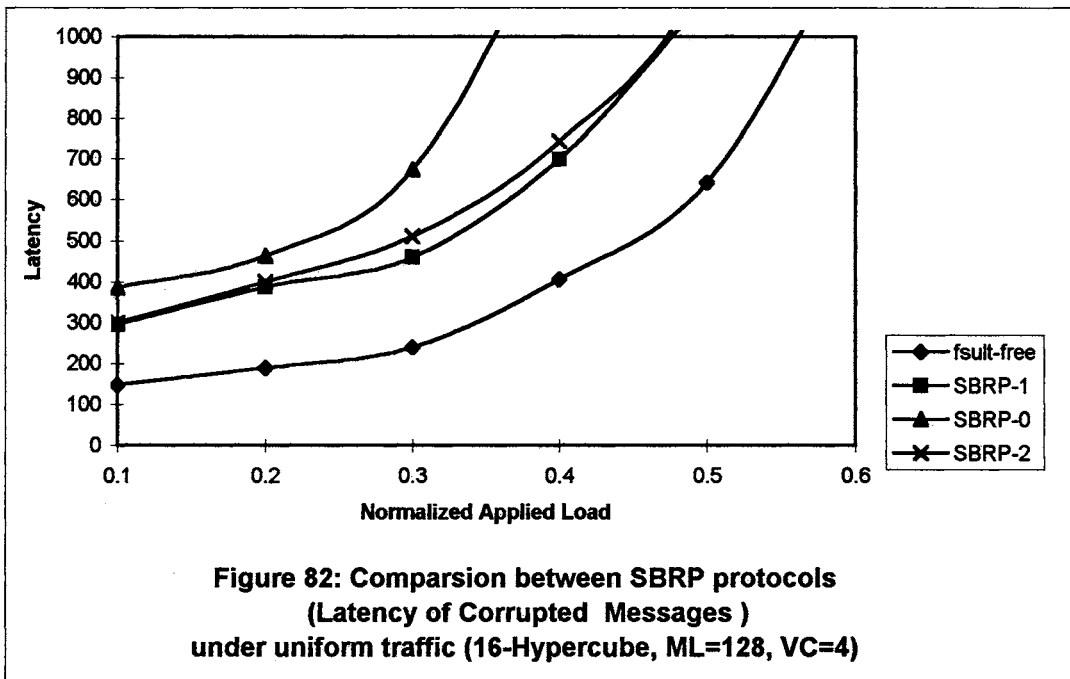
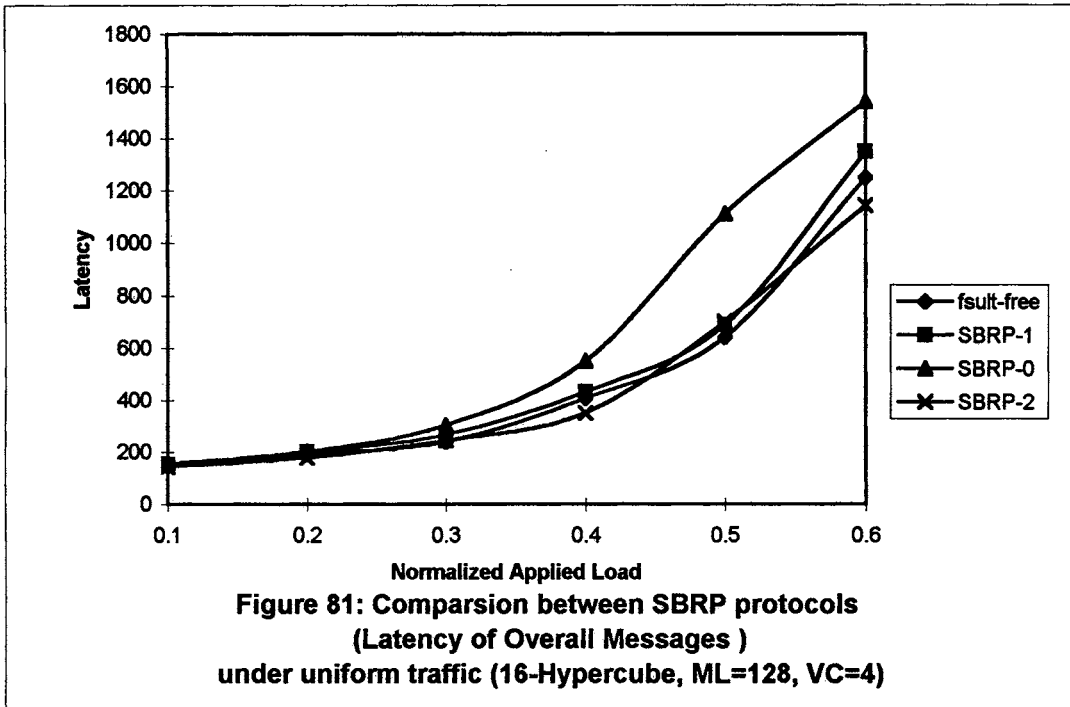


**Figure 77: Performance of SBRP-0 under different fault rates
(Latency of overall messages)
(Uniform traffic ,16-Hypercube, ML=128, VC=4)**



**Figure 78: Performance of SBRP-0 under different fault rates
(Latency of corrupted messages)
(Uniform traffic ,16-Hypercube, ML=128, VC=4)**





Chapter 7

CONCLUSIONS

This chapter summarizes the contributions of this dissertation, and present overall conclusion and recommendations. We conclude with some suggestion for future research.

7.1 Summary of Work

Our work in this dissertation can be divided to four main parts:

1- Developing a sophisticated wormhole network simulator to emulate the function of actual router and network interface in a parallel machine. Message initiation, formatting, injecting, buffering and queuing have been implemented. Message routing also implemented including path selection, virtual channel reserving and releasing, and input buffers and output buffers mapping. Also arbitrating more than one virtual channel on a physical channel is designed. Status registers and flit counters also have been considered.

Our simulator gives us an useful tool to study interconnection networks with different topologies and with different design parameters. For example, different traffic patterns can be used with any selected message injection rate. Also, number of virtual channels or message length can be selected. Finally, our simulator gives many statistics like average message latency, throughput, blocking times and number of generated flits.

2- Simulating and studying many types of routing algorithms. We have implemented deterministic, partially adaptive and adaptive routing algorithms for both mesh and hypercube topologies. Moreover, fault tolerant algorithms also were implemented.

3- Chapter 3 of the dissertation is dedicated to study and to evaluate the data recovery protocols presented in available literature in interconnection networks. We found that all

current recovery protocols incur substantial overhead with every single message to achieve data recovery. Taking into consideration that faults occurrence is rare, attaching fault recovery overhead with every message deteriorating fault-free performance. Moreover, all current recovery protocols need special hardware requirements.

4- To overcome the drawbacks of current recovery protocol, we propose a software based recovery protocol (SBRP) which does not affect the performance of fault-free performance. To recover from faults, SBRP uses control software messages instead of hardware acknowledgment. Recovery control messages are used only when a fault happens. Corrupted message is recovered by sending a control message to the original sender. Once the original sender gets the control message, it resends the corrupted message or part of it. We implement three versions of SBRP (SBRP-0, SBRP-1, SBRP-2). SBRP-2 has the minimum hardware requirements where SBRP-1 has the maximum requirements. They also differ in recovery process steps.

7.2 Conclusions

Since the message latency is the main obstacle to achieve high performance in the massive parallel processors (MPP) and even in the network of workstations (NOW), a lot of research have been conducted to reduce the required time to exchange messages between nodes. Traditional message passing systems incur huge overhead by calling operating system handlers and by using multilevel buffering. On the other hand, recent message passing systems achieve low latency messaging by eliminating unnecessary buffering.

Reliable delivery is one of the main requirements provided by the interconnection networks. Most of network systems use an end-to-end protocol to guarantee reliable delivery. In an end-to-end protocol, the sender waits until it gets an acknowledgment from the receiver. Traditional systems use software acknowledgment messages which increase the network traffic. Recent systems use control lines to send hardware acknowledgment. In general, end-to-end recovery protocols incur overhead with every message.

What current recovery protocols do by increasing every message latency makes the cost of data recovery very high. Since the probability of fault occurrence is low and the number of messages which corrupted by the fault is low, a data recovery protocol should not effect normal operations. Message latency of normal (fault-free) message should not be increased to solve a rare problem. The data recovery protocol must be invoked in fault occurrence only.

In this dissertation we propose a new recovery protocol (SBRP). SBRP is not an end-to-end protocol. Therefore, SBRP does not penalize every message. In case of fault occurrence only, SBRP invokes a special handler to resend the corrupted messages. Only the latencies of corrupted messages have been increased. Because SBRP depends on sending control messages, the recovery overhead is high. However, the high overhead is attached with corrupted messages only, if any. Our simulation studies show good performance of SBRP even with high fault rate environments.

Although SBRP shows good performance, it needs hardware and software requirements. SBRP may require additional hardware support like flit counters, special registers to save header information or some changes in control logic of router's finite state machine. Also, SBRP requires network interface help to send control messages and

to resend corrupted messages. However, the great gain on performance might justify all the requirements.

7.3 Open Topics

In this dissertation we explained the general hardware and software requirements of SBRP. However, integrating current recovery protocol including SBRP in current routers' design needs more studies. The communication between router and network interface needs to be more efficient. New designs should provide bi-directional information exchange between router and network interface.

All current recovery protocols are implemented to recover messages in one-to-one messages environment. Recovery process in collective communication [65] like one-to-all or all-to-all have not addressed yet.

Most of current recovery protocols do not address fault recovery in real time systems. A real time system is required to deliver the expected service in a timely manner even in the presence of failures. A fault-tolerance policy should be implemented to recover from faults within the time limit (deadline).

References

- 1- W. Dally, L. Dennison, D. Harris, and T. Xanthopoulos, "Architecture and Implementation of the Reliable Router," Proceedings of Hot Interconnects II, Palo Alto, CA, IEEE Comput. Soc., August 1994.
- 2- J. Kim, Z. Liu and A. Chien, "Compressionless Routing a Framework for Adaptive and Fault Tolerant Routing," in IEEE Transactions on Parallel and Distributed Systems, Vol. 8, no. 3, March 1997.
- 3- P. Gaughan, B. Dao, S. Yalamanchili and D. E. Schimmel. "Distributed Deadlock-Free Routing in Faulty, Pipeline k-ary n-cube," IEEE Transactions on Computers, Vol. 45, no. 6, June 1996.
- 4- J. Kim and A. Chien. "Network Performance Under Hybrid Traffic Loads," Journal of Parallel and Distributed Computing, Vol. 28, no. 1, July 1995.
- 5- J. Kim and A. Chien. "The Impact of Packetization in Wormhole-Routed Networking," In Proceeding of Parallel Architectures and Languages Europe 93, Munich, Germany, June 1993.
- 6- V. Karamcheti and A. Chien. "Do Faster Routers Imply Faster Communication?" In Proceedings of Parallel Computer Routing and Communication Workshop. Seattle, Washington, University of Washington, May 1994.
- 7- W. Dally. "Virtual-Channel Flow Control," IEEE Transactions on Parallel and Distributed Systems, Vol. 3, no. 2, March 1992.
- 8- Tze C. Lee and John P. Hayes. "A Fault-Tolerant Communication Scheme for Hypercube Computers," IEEE Transactions on Computers, Vol. 41, no. 10, October 1992.
- 9- V. Dao, J. Duato, and S. Yalamanchili, "Configurable flow control mechanism for fault tolerant routing," Proceedings of the 22nd International Symposium on Computer Architecture (ISCA), Santa Margherita Ligure, Italy, ACM SIGARCH, June 1995.

- 10- P. Gaughan and S. Yalamanchili " Adaptive Routing Protocols for Hypercube Interconnection networks," Computer, Vol. 26, no. 5 , May 1993 .
- 11- Ni and P. McKinley. "A Survey of Wormhole Routing Techniques in Direct Networks," Computer, Vol. 26, no. 2, February 1993.
- 12- Anjan K. V. and T. M. Pinkston. "An Efficient Faulty Adaptive Deadlock Recovery Scheme: DISHA," Proceedings of the 22nd International Symposium on Computer Architecture (ISCA), Santa Marqherita Ligure, Italy, ACM SIGARCH , June 1995.
- 13- W. 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, April 1993.
- 14- C. Su and K. Shin. "Adaptive Fault Tolerant Deadlock-Free Routing in Meshes and Hypercubes," IEEE Transactions on Computer, Vol. 45, no. 6, June 1996.
- 15- Y. Suh. B. Dao. J. Duato and S. Yalamanchili "Software Based Fault-Tolerant in Pipeline Network," In Proceedings of the 24th International Conference on Parallel Processing, Oconomowoc, Wisconsin, Pennsylvania State University, August 1995.
- 16- V. Karamcheti and A. Chien "Software Overhead in Messaging Layers: Where Does the time go?," In Proceedings of ASPLOS-VI, San Jose, CA, October 1994.
- 17- T. von Eicken, D. Culler, S. Goldsten and K. Schauster." Active Messages: A Mechanism for integrated Communication and Computation," Proceedings of the International Symposium on Computer Architecture, Gold Coast, Australia, ACM SIGARCH, June 1992.
- 18- J. Duato. "A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks," IEEE Transactions on Parallel and Distributed Systems, Vol. 6, no. 10, October 1995.
- 19- W. Dally. "Network and Processor Architecture for Message-Driven Computers," in VLSI and Parallel Computation. R. Suaya and G. Birtwhistle, Eds., Morgan Kaufmann, Los Altos, CA 1990.
- 20- M. Noakes, D. Wallach and W. Dally. " The J machine Multicomputer: an Architecture Evaluation," Proceedings of the 20th International Symposium on Computer Architecture , San Diego, CA, ACM SIGARCH, May 1993.

- 21- P. Nuth and W. Dally. "The J-Machine Network," Proceedings of the International Conference on Computer Design: VLSI in computer and Processors, Cambridge, MA, IEEE Comput. Soc., October 1992.
- 22- W. Dally et al. "The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanism," IEEE Micro, Vol. 12, no. 2, April 1992.
- 23- D. Talia. "Message-Routing systems for Transputer-Based Multicomputers" IEEE Micro, Vol. 13, no. 3 June 1993.
- 24- Y. Tsai and P. McKinley. "A Broadcast Algorithm for All-Port Wormhole-Routed Tours Networks" IEEE Transactions on Parallel and Distributed Systems, Vol. 7, no. 8, August, 1996.
- 25- Cray Research, Inc. CRAY T3D System Architecture Overview, 1993.
- 26- Intel Scaleable System Division. Intel Paragon Systems Manual. Intel Corporation, <http://www.ssd.intel.com/paragon.html>.
- 27- nCUBE3 Processor Overview. nCUBE, <http://www.ncube.com/products/cpu.html>.
- 28- S. Broker et al "Supporting systolic and memory communication in iWarp." Proceedings of the 17th Annual International Symposium on Computer Architecture, Seattle, WA, ACM SIGARCH, May 1990.
- 29- C. B. Stunkel et al. "The SP-1 High Performance Switch," Proceeding of Scaleable High Performance Computing Conference, Knoxville, TN, IEEE Comput. Soc., May 1994.
- 30- L. Choi and A. Chien. "Integrating Networks and Memory Hierarchies in a Multicomputers Node Architecture" Proceedings of the 8th International Parallel Processing Symposium , Cancun, Mexico, IEEE Comput. Soc., April 1994.
- 31- B. Dao, S. Yalamanchili and J. Duato. "Architecture Support for Reducing Communication Overhead in Multiprocessor Interconnection Networks" Proceedings of the Third International Symposium on High Performance Computer Architecture, San Antonio, TX, IEEE Comput. Soc., February 1997.
- 32- V. Karamcheti and A. Chien. "A comparison of Architectural Support for Messaging in the TMC CM-5 and the Cray T3D" Proceedings of the 22nd International Symposium on Computer Architecture (ISCA), Santa Marqherita Ligure, Italy, ACM SIGARCH, June 1995.

- 33- E. Spertus et al. "Evaluation of Mechanisms for Fine-Grained Parallel Programs in the J-machine and CM-5," Proceedings of the 20th Annual International Symposium on Computer Architecture, San Diego, CA, ACM SIGARCH, May 1993.
- 34- D. Culler, L. Liu, R. Martin and C. Yoshikawa. "Assessing Fast Network Interface," IEEE Macro, Vol. 16, no. 1, February 1996.
- 35- R. Boppana and S. Chalasani . "A comparison of adaptive wormhole routing algorithms," Proceedings of the 20th International Symposium on computer Architecture, San Diego, CA, ACM SIGARCH, May 1993.
- 36- J. Duato, S. Yalamanchili and L. Ni. "Interconnection Networks: An Engineering Approach," www.ee.gatech.edu/users/sudha
- 37- Stunkel et al., "The SP-2 High Performance Switch," IBM System Journal, Vol. 34, no. 2, 1995.
- 38- Gaughan, B. V. Dao and S. Yalamanchili. "A Family of fault-tolerant routing protocols for direct multiprocessor network," IEEE Transactions on Parallel and Distributed Systems, Vol. 6, no. 5, May 1995.
- 39- C. Glass and L. Ni, "The Turn Model for Adaptive Routing," Journal of ACM, Vol. 41, no. 5, September 1994.
- 40- J. Duato. "Deadlock free adaptive routing algorithm for multicomputers: Evaluation of a new algorithm," Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing, Dalas, TX, ACM, December 1991.
- 41- Z. Liu and A. Chien. "Hierarchical Adaptive Routing: A framework for Fully Adaptive and Deadlock-Free Wormhole Routing," In Proceedings of the 6th IEEE International Symposium on Parallel and Distributed Processing, Dallas, TX, IEEE Comput. Soc., October 1994.
- 42- J. Kim and K. Shin. "Deadlock-Free Fault-Tolerant Routing in Injured Hypercubes," IEEE Transactions on Computer, Vol. 42, no. 9, September 1993.
- 43- C. Glass and L. Ni. "Fault-Tolerant Wormhole Routing in Meshes," In Proceedings of 23rd International Symposium on Fault-tolerant Computing, San Diego, CA, ACM SIGARCH, June 1993.

- 44- A. Chien and J. Kim. "Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors" in Proceedings of the 19th International Symposium on Computer Architecture, Gold Coast, Australia, ACM SIGARCH, May 1992.
- 45- R. Boppana and S. Chalasani. "Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks," IEEE Transactions on Computer, Vol. 44, no. 7, July 1995.
- 46- W. Dally and C. Seitz. "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," IEEE Transactions on Computer, Vol. C-36, no. 5, May 1987.
- 47- T. Pinkston and S. Warnakulasuriya. "On Deadlock in Interconnection Networks," in Proceedings of International Symposium on Computer Architecture, Denver, CO, ACM SIGARCH, June 1997.
- 48- Q. Li. "Minimum Deadlock-Free Message Routing Restrictions in Binary Hypercube," Journal of Parallel and Distributed Computing, Vol. 15, no. 2, June 1992.
- 49- D. Linder and J. Harden. "Wormhole Routing for k-ary n-cubes". IEEE Transactions on Computers, Vol. 40, no. 1, January 1991.
- 50- C. Cunningham and D. Avresky. "Fault-Tolerant Adaptive Wormhole Routing in Two-Dimensional Meshes," In Proceedings of the First International Symposium on High Performance Computing Architecture, IEEE Comput. Soc., Raleigh, NC, January 1995.
- 51- Anjan V. , T. Pinkston and J. Duato. "Generalized Theory for Deadlock-Free Adaptive Wormhole Routing and its Application to Disha Concurrent," The 10th International Parallel Processing Symposium, Honolulu, HI, ACM SIGARCH, April 1996.
- 52- C. Su and K. Shin. "Adaptive Deadlock-Free Routing in Multicomputers Using Only One Extra Virtual Channel," International Conference on Parallel Processing, Syracuse, NY, CRC press, August 1993.
- 53- L. Schwieber and D. Jayasimha. " A Universal Proof Technique for Deadlock-Free in Interconnection Networks," In 7th Annual ACM Symposium on Parallel Algorithms and Architectures, Santa Barbara, CA, ACM SIGARCH July 1995.
- 54- L. Schwieber and D. Jayasimha. "Optimal Fully Adaptive Minimal Wormhole Routing for Meshes," Journal of Parallel and Distributed Computing, Vol. 27, no. 1 , May 1995.

- 55- D. Jayasimha et al. "A foundation for Designing Deadlock-Free Routing Algorithms in Wormhole Networks," In Symposium on Parallel and Distributed Processing, New Orleans, LA, IEEE Comput. Soc., October 1996.
- 56- G. Chiu and S. Wu. " A Fault-Tolerant Routing Strategy in Hypercube Multicomputers," IEEE Transactions on Computers, Vol. 45, no 2, February 1996.
- 57- J. Duato. "A Theory of Fault-Tolerant Routing in Wormhole Networks, " The International Conference on Parallel and Distributed Systems, Hsinchu, Taiwan, Nat. Chiao Tung Univ., December 1994.
- 58- J. Upadhyay, V. Varavithya, and P. Mohaparta. " An Efficient Fault-Tolerant Routing Scheme for Two Dimensional Meshes," The First International Symposium on High Performance Computer Architecture, Raleigh, NC, IEEE Comput. Soc., January 1995.
- 59- Y. Boura and C. Das. "Fault-tolerant Routing in Mesh Networks," The International Conference on Parallel Processing. August 1995.
- 60- B. Harry et al. "A Fault-Tolerant Communication System for the B-HIVE Generalized Hypercube Multiprocessor, " Third Conference on Hypercube Concurrent Computers, Pasadena, CA, ACM press, January 1988.
- 61- D. Grunwald and D. Reed. "Networks for Parallel Processors: Measurements and Prognostications," Hypercube Multiprocessor" Third Conference on Hypercube Concurrent Computers, ACM press, January 1988.
- 62- "nCUBE3 processor Overview". nCUBE corporation, <http://www.ncube.com/products/cpu.html>. accessed March 1997.
- 63- C. Chang, G. Czajkowski, C. Hawblitzel and T. Eicken. "Low-Latency Communication on the IBM RISC System/6000 SP," Proceedings of Supercomputing, Pittsburgh, PA, ACM SIGARCH, November 1996.
- 64- MPI report. [Http:// www.msc.anl.gov/mpi/mpi-report](http://www.msc.anl.gov/mpi/mpi-report).
- 65- P. McKinley and D. Robinson. "Collective Communication in Wormhole-Routed Massively Parallel Computers," Computer, Vol. 28, no. 12, December 1995.

- 66- W. Dally et al. "Architecture and Implementation of the Reliable Router," In the Proceedings of Hot Interconnects II, IEEE Comput. Soc., August 1994.
- 67- P. Mohapatra, "Wormhole Routing Techniques in Multicomputer Systems," Iowa State Univ. TR-ACAR-95-01.
- 68- News Track. Communication of the ACM. March 1997.
- 69- S. Pakin, V. Karamcheti and A. Chien. "Fast Messages: Efficient, Portable Communication for Workstation Clusters and MPPs," IEEE Concurrency, Vol. 5, no. 2, April-June 1997.
- 70- N. Boden et al. "Myrinet: A Gigabit-per-Second Local Area Network," IEEE Micro, Vol. 15, no. 1, February 1995.
- 71- R. Horst. "TNet: A Reliable System Area Network," IEEE Micro, Vol. 15, no. 1, February 1995.
- 72- K. Aoyama and A. Chien. "The Cost of Adaptivity and Virtual Lanes in a Wormhole Router," Technical Report, Department of Electrical Engineering, University of Illinois at Urbana Champaign, 1994.

2
VITA

Mohammad S. Alowayed

Candidate for the Degree of

Doctor of Philosophy

**Thesis: DATA RECOVERY IN WORMHOLE ROUTING
NETWORKS IN HYPERCUBES AND MESHES**

Major Field: Computer Science

Biographical:

Personal Data: Born in Arras, Saudi Arabia, on August 25, 1966, the son of Saleh Alowayed and Fatima Alhazza.

Education: Graduated from Alfisal secondary school, Riyadh, Saudi Arabia in 1983. Received Bachelor of Science degree in Computer Engineering from King Saud University, Riyadh, Saudi Arabia in 1989. Received Master of Science degree in Computer Science from Western Michigan University in July 1993. Completed the requirements for the Doctor of Philosophy degree with a major in Computer Science at Oklahoma State University in December 1997.

Experience: Employed by Alahssa Technology College as a teaching faculty (August 1989 - February 1991) and (August 1993 - Jan 1994)

Professional Memberships: IEEE computer society, and ACM.