

THESIS

3



This is to certify that the

dissertation entitled


MULTICASTING IN MULTISTAGE
INTERCONNECTION NETWORKS

presented by

Chi-Ming Chiang

has been accepted towards fulfillment
of the requirements for

PhD degree in Computer Science


Major professor

Date May 26, 1995

LIBRARY
Michigan State
University

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.

DATE DUE	DATE DUE	DATE DUE
1988	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

MSU is An Affirmative Action/Equal Opportunity Institution

c:\circ\datedue.pm3-p.1

MULTICASTING IN MULTISTAGE
INTERCONNECTION NETWORKS

By

Chi-Ming Chiang

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

1995

ABSTRACT

MULTICASTING IN MULTISTAGE INTERCONNECTION
NETWORKS

By

Chi-Ming Chiang

Multicast communication, also known as multi-point communication, refers to the delivery of a message from a single source node to a number of destination nodes. It is a frequently used communication pattern in distributed-memory parallel computers and computer networks. *Multistage interconnection networks* (MINs) have resurged as another popular class of interconnection architecture for constructing scalable parallel computers and high speed network switches. While efficient implementation of multicast communication is critical to the performance of message-based scalable parallel computers and switch-based high speed networks, little research has been devoted to supporting multicast in MINs.

Unlike unicast communication, the size of a header in a multicast message depends on the number of destinations, the distribution of destinations, and the multi-address encoding/decoding schemes. This research suggests and compares six different multi-address encoding/decoding schemes to shorten the header which is an overhead to

the system. Each of them has its own advantages and disadvantages. An appropriate choice of the multi-address encoding scheme depends on the destination pattern and is detailed in this research.

Several efficient multicast algorithms, both hardware and software implementations, for unidirectional wormhole-switched MINs are proposed in this research. The hardware implementation offers better performance than the software approach. Tree-based hardware approaches for wormhole-switched MINs, namely *multi-head worms*, require special mechanisms to avoid potential deadlocks when there are multiple multicasts. As shown in this research, the hardware approaches to support multicast should be considered in the design of high performance networks. In systems which do not support hardware multicast, multicast must be implemented atop existing unicast communications. This research proposes an efficient unicast-based multicast (or software multicast) algorithm for such systems.

While Banyan MINs are limited to a unique routing path between any source and destination pair, an extra stage MIN can provide extra routing paths. Extra routing paths can reduce the message transmission blocking probability and allow additional flexibility in selecting a routing path. An algorithm to find a traffic-optimal multicast tree in such networks within polynomial time is proposed.

Many new ideas and new algorithms are proposed to support efficient multicast communication in wormhole-switched MINs. Performance evaluation and comparison of different approaches are conducted through extensive simulation experiments. Research results obtained from this work will be extremely useful to parallel computer and network switch designers who wish to support multicast in their designs.

© Copyright 1995 by Chi-Ming Chiang
All Rights Reserved

To my parents

ACKNOWLEDGMENTS

I would like to take this opportunity to express my appreciation to several persons, without whom this dissertation could not have been completed. My achievements, great or little, were possible through their participation. I will always be indebted to my advisor, Lionel M. Ni. He has been my mentor, my colleague, and my friend. His very positive influence on my personal and technical development will carry forward into my future endeavors.

I am very grateful to the other members of my dissertation committee: Herman D. Hughes, Abdol Esfahanian, and Raoul D. LePage, for their valuable comments, help, encouragement, and friendship. I would also like to thank all my colleagues and friends who made my stay at Michigan State University enjoyable.

A person cannot accomplish anything without the help and understanding of family members. I thank my parents, brother, and sisters for their contiguous encouragement, support, patience, and love. I appreciate my host family, Ows, for their help, love, and friendship throughout the course of my master and doctorate work. I proudly share this accomplishment with them all.

Last, but not least, my very special thanks go to my wife Ya-Ping and her family for sustaining me with their everlasting love and understanding.

TABLE OF CONTENTS

LIST OF TABLES	x
LIST OF FIGURES	xi
1 Introduction	1
1.1 Wormhole Switching	4
1.2 Multistage Interconnection Networks	5
1.3 Motivation and Problem Definition	5
1.4 Performance Metrics	13
1.5 Objectives and Thesis Outline	14
2 Multi-Address Encoding	18
2.1 Header Encoding Design Considerations	19
2.2 Multi-Address Encoding Schemes	21
2.2.1 All-Destination Encoding	21
2.2.2 Bit String Encoding	22
2.2.3 Multiple Region Broadcast	23
2.2.4 Multiple Region Stride	25
2.2.5 Multiple Region Mask	27
2.2.6 Multiple Region Bit String	28
2.3 Multi-Address Decoding	29
2.3.1 All Destination Decoding	31
2.3.2 Bit String Decoding	33
2.3.3 Multiple Region Broadcast Decoding	34
2.3.4 Multiple Region Stride Decoding	35
2.3.5 Multiple Region Mask Decoding	37
2.3.6 Multiple Region Bit String Decoding	37
2.4 Summary	39
3 Multistage Interconnection Networks	40
3.1 Switches	41
3.2 Network Topology	42
3.3 Node Architecture	45
3.4 Decoding in Multistage Interconnection Networks	45
3.4.1 All Destination Decoding	49

3.4.2	Bit String Decoding	51
3.4.3	Multiple Region Broadcast Decoding	53
3.4.4	Multiple Region Stride Decoding	54
3.4.5	Multiple Region Mask Decoding	56
3.4.6	Multiple Region Bit String Decoding	57
3.5	Performance of Multi-Address Encoding and Decoding on Multistage Interconnection Networks	59
3.6	Summary	64
4	Hardware Multicast Wormhole-Switched	66
4.1	Implementation of Multi-head Worms	67
4.2	The Synchronous Multi-head Worm	73
4.3	Multi-address Encoding	78
4.4	Performance and Comparison	82
4.5	Summary	88
5	Network Partitionability and Traffic Localization	89
5.1	Influence of Traffic Localization	90
5.2	Definition of Different Cubes	94
5.3	Contention Free and Channel Balanced Partition	96
5.4	Non Contention-Free Partition	102
5.5	Modification of Baseline and Butterfly Networks	105
5.5.1	Modification of a Baseline Network	106
5.5.2	Modification of a Butterfly Network	108
5.6	Performance Evaluation	111
5.7	Summary	115
6	Extra Stage Multistage Interconnection Networks	117
6.1	MINs with Extra Stages	119
6.1.1	Interstage Connection Patterns	119
6.2	Structural Equivalence of different Delta Networks	121
6.2.1	Design of Extra-Stage MINs	121
6.2.2	Design Choices	124
6.3	Multicast in Extra-Stage MIN	126
6.3.1	Alternate Routing Styles in Extra-Stage MINs	127
6.3.2	Distributed Routing and Multicast Implementation	128
6.3.3	Number of Multicast Trees: Upper Bound	130
6.3.4	Multicast Tree Selection Criteria	131
6.3.5	Traffic Optimal Multicast Trees	132
6.4	Multicast Algorithm	134
6.4.1	Latest Branch Multicast Algorithm	135
6.4.2	Optimality Proof and Complexity	137
6.4.3	Other Sub-Optimal Multicast Heuristics	138
6.5	Performance	142
6.5.1	Simulation Description	142

6.5.2	Dimension Patterns and Output Parameters	143
6.5.3	Plots and Observations	144
6.6	Summary	147
7	Software-based Multicast	149
7.1	Issues in Multicast Communication	149
7.2	Multicast Algorithm	153
7.3	Non-Optimal Multicast in Baseline and Butterfly Networks	157
7.4	The C-min Algorithm	159
7.5	Performance Evaluation	163
7.6	Summary	167
8	Related Work	169
8.1	MIN-based ATM Switches	171
8.2	Hardware Multicast	177
8.2.1	Path-based Multicast	177
8.2.2	Trip-Based Multicast	179
8.2.3	A Multidestination Worm Conforming to Base Routing Schemes	180
8.2.4	Synchronous Receiver Initiated Multicast	181
8.3	Software Multicast	182
8.4	Summary	185
9	Conclusions and Future Work	187
9.1	Research Contributions	187
9.2	Directions for Future Research	190
	BIBLIOGRAPHY	192

LIST OF TABLES

3.1	Number of flits in a header based on various multi-address encoding schemes. A header consists of a counter and addresses.	61
6.1	Patterns of this study	143

LIST OF FIGURES

1.1	Broadcast 100 integers from process 0 to every process in the group. . . .	2
1.2	The framework	3
1.3	Path-based hardware multicast.	7
1.4	Tree-based hardware multicast.	8
1.5	An asynchronous multicast tree.	8
1.6	A synchronous multicast tree.	9
2.1	The message header format of six address encoding schemes.	21
2.2	Single region broadcast and single region stride, where the source node is 0 and the destination sets are {4, 5, 6} and {1, 3, 5}, respectively. . .	24
2.3	Multicast to the same row and column in (a) 2D mesh and (b) linear array architectures.	26
2.4	(a) A generic switch/router with four input/output ports, and the message is forwarded via both output ports 1 and 3. (b) The receiving of a duplicated message when the header is not handled properly and alternate routing paths are allowed.	29
2.5	All destination decoding algorithm.	32
2.6	An example of all-destination decoding.	32
2.7	Buffered bit string decoding algorithm.	33
2.8	An example of buffered bit string decoding.	34
2.9	An example of hierarchical bit string decoding.	34
2.10	Multiple region broadcast decoding algorithm.	35
2.11	An example of multiple region broadcast decoding.	36
2.12	An example of multiple region stride decoding.	36
2.13	An example of multiple region mask decoding.	37
2.14	Multiple region bit string decoding algorithm.	38
2.15	An example of multiple region bit string decoding.	38
3.1	A generic MIN structure with $N = k^n$ input/output ports and n stages. . . .	41
3.2	Four 16-node MINs built with 2×2 switches.	46
3.3	A 16×16 cube network built with 2×2 switches.	48
3.4	A duplicated receiving when there are alternative paths, in an extra stage MIN, between source and destination nodes.	48
3.5	All address decoding algorithm.	50
3.6	An example of all-destination decoding.	51
3.7	The algorithm of bit string decoding schemes for MIN.	52
3.8	An example of buffered bit string decoding.	52

3.9	An example of hierarchical bit string decoding.	53
3.10	Multiple region broadcast decoding algorithm.	54
3.11	An example of multiple region broadcast decoding.	55
3.12	An example of multiple region stride decoding.	55
3.13	Define a subcube in a MIN.	56
3.14	An example of multiple region mask decoding.	57
3.15	Multiple region bit string decoding algorithm for MIN.	58
3.16	An example of multiple region bit string decoding.	58
3.17	The length of header and reduction rate on different destination patterns vs different multi-address encoding/decoding schemes.	62
4.1	An example of deadlock for a path-based multicast on a 16-node multistage cube network.	68
4.2	An example to establish a wormhole-switched multicast tree, where the shadow box represents the header flit and white box stands for data flits.	70
4.3	Establishment of an asynchronous multi-head worm with network con- tention.	71
4.4	An example of a deadlock situation with two multi-head worms, where the bold line represents the multi-head worm T_0 and the dotted line stands for T_6	72
4.5	Establishment of a synchronous multi-head worm with network contention.	74
4.6	Two multi-head worms requesting common channels to achieve destination sets may cause deadlock or starvation.	74
4.7	A deadlock example with all-destination encoding/decoding scheme with 3 synchronous multi-head worms, where the destination information carried by a flit is numbered within the box.	81
4.8	Comparison among hardware and software implementations.	84
4.9	Latency comparison of different switch sizes and different number of des- tinations on a cube network.	85
4.10	Comparison on a butterfly network with different sized switches and a different number of destinations.	86
4.11	Latency comparison of the different number of nodes in each cluster.	86
4.12	Comparison between the region broadcast and pseudo all-destination en- coding schemes.	87
5.1	Average latency and associated system utilization of a blocking multicast message on a 64-node cube network.	92
5.2	Three different processor clusters, $0xxx$, $10xx$ or $2X$, and $11x1$	96
5.3	A 16-node cube network based on 4×4 switches is partitioned into four contention-free and channel-balanced binary cube clusters, $x0xx$, $01xx$, $11x0$ and $11x1$	100
5.4	An 8-node cube network and an 8-node omega network are partitioned into three contention-free and channel-balanced binary cube clusters.	101
5.5	A 16-node baseline network is partitioned into different binary clusters.	103
5.6	A 16-node butterfly network is partitioned into different binary clusters.	104

5.7	An example to partition a 16-node modified baseline network into contention-free and channel-balanced disjoint k -ary cubes.	108
5.8	An example to partition modified butterfly network into contention-free and channel-balanced disjoint k -ary cubes.	111
5.9	Average latency of a blocking multicast message with various intertask contention.	112
5.10	Average throughput in cube network with average 64-flit messages. . . .	113
5.11	Average latency in cube network of non-blocking message with average 64-flit length.	114
5.12	Average latency in cube network of non-blocking message with multiple sources.	115
6.1	A generic MIN structure with $N = 2^n$ input/output ports and h stages, where each 2×2 switch has four connectivity choices.	118
6.2	Structural equivalence between (a) PS-MIN(16) and CU-MIN(16) and (b) PS-ESMIN(16:5) and CU-ESMIN(16:1,3,2,1,0)	122
6.3	An example of the inadequacy of the traditional routing approach (source \oplus destination): A source node 1 to destination node 2 route does not lead to the right destination.	128
6.4	The four optimal multicast trees in an ESMIN(8:1,2,1,2,0)	134
6.5	The eight optimal multicast trees in an ESMIN(8:1,1,2,2,0)	134
6.6	Latest-Branch multicast algorithm illustration: a) CU-ESMIN(16:1,2,3,3,1,1,0), b) Construction of the CU-ESMIN ¹ by seeking the rightmost occurrence of each dimension and marking the remaining stages as 'horizontal.' Horizontal stages merely carry-forward the data. c) Example multicast in the CU-ESMIN ¹ . . .	136
6.7	An example of blocking in multicast tree construction by the RB algorithm.	141
6.8	The number of channels used in the first branch algorithm.	144
6.9	The number of channels used in the latest branch algorithm.	145
6.10	The number of channels used in the random mapping algorithm.	145
6.11	The probability of blocking in random branch algorithm.	147
6.12	The number of channels used in three non-blocking algorithms.	148
7.1	An example of deadlock in single region broadcast.	150
7.2	Unicast-based software multicast trees	152
7.3	The C-min algorithm	160
7.4	An example of software multicast tree implemented by C-min algorithm. . .	161
7.5	The latency in blocking multicasts.	164
7.6	The latency of C-min algorithm on various networks and various sized switches.	166
7.7	The latency of various algorithms on a cube network constructed by 4×4 switches.	167
7.8	The throughput of various algorithms on cube networks with 4×4 switches. .	168
8.1	Switch fabric.	172

8.2	A 16×16 copy network.	173
8.3	An example of synchronous region broadcast.	176
8.4	An example of a path-based multicast in a 2-D mesh. The lower pair of numbers is the absolute address of a node and the upper number is the relative address of the same node based on a Hamiltonian path. . . .	178
8.5	An example of a trip-based multicast.	180
8.6	Examples of a multidestination worm conforming to base routing schemes.	181
8.7	An example of a U-min multicast.	183
8.8	An example of two multicasts based on U-min and C-min algorithms. . .	185

CHAPTER 1

Introduction

Efficient data communication among processor nodes is critical to the performance of message-based *scalable parallel computers* (SPCs). Generally data communication can be classified into point-to-point communication and collective communication [1]. While point-to-point communication deals with the basic *send* and *receive* operations between two nodes, collective communication deals with communication that involves a group of processes which form a *process group*. Multiple applications may space-share an SPC in a way that processor nodes in the system can be partitioned into several disjoint subsets, namely *processor clusters* or *clusters*, each of which is dedicated to a distinct application. Multiple processes may time-share a processor node and multiple process groups may exist in a cluster. Each process group includes a subset of all processes in the cluster. As a result, a system-level multicast service, in which the same message is delivered from a source node to an arbitrary number of destination nodes, is fundamental in supporting collective communication primitives including the application-level broadcast, reduction, and barrier synchronization [2].

Multicast communication is also demanded in high speed networks, such as ATM switches, for various network applications [3, 4] including multimedia applications (*e.g.*, [5]).

For example, the format of the group broadcast in MPI is specified as follows:

`MPI_Bcast(buffer, count, datatype, root, comm)`

The parameters *buffer*, *count*, *datatype*, *root*, and *comm* represent the data buffer, number of data, data type, source process, and communicator, respectively. The communicator is used to distinguish different process groups since a process can join more than one process group. The message is delivered to every process which owns the specified communicator. Figure 1.1 gives an example that source process 0 forwards an array of 100 integers to all processes in the process group specified by *comm*. Note that this example only shows partial codes. Some of the variables, such as *comm*, must have been assigned appropriate values before these codes.

```
MPI_Comm comm;
int          array[100];
int          root = 0;
...
MPI_Bcast (array, 100, MPI_INT, root, comm);
```

Figure 1.1: Broadcast 100 integers from process 0 to every process in the group.

As indicated in [6], lots of applications require multicast communication. As the scale of application increases, communication becomes a bottleneck and degrades the overall performance [7].

The relationship between different applications, high level parallel languages, communication packages, ATM adaptor layer and networks is shown in Figure 1.2. The

message passing interface (MPI) explicitly supports application-level broadcast as well as multicast [1] while High Performance Fortran (HPF) supports these functions implicitly [8]. Both of them rely on the system to provide efficient multicast communication. The interconnection network for these applications can be either a special purpose network or an asynchronous transfer mode (ATM) network.

Teleconferencing, multimedia, and interactive television are some newly emerging and natural applications for multicast communication. As long as there are more than two parties communicating in a teleconference, multicast communication is required. Although the source may send multiple copies of the message individually (one for each party), the scalability of such an approach is limited by the bandwidth, and the latency among different parties may become intolerable. Efficient multicast support becomes a critical issue for the success of these applications.

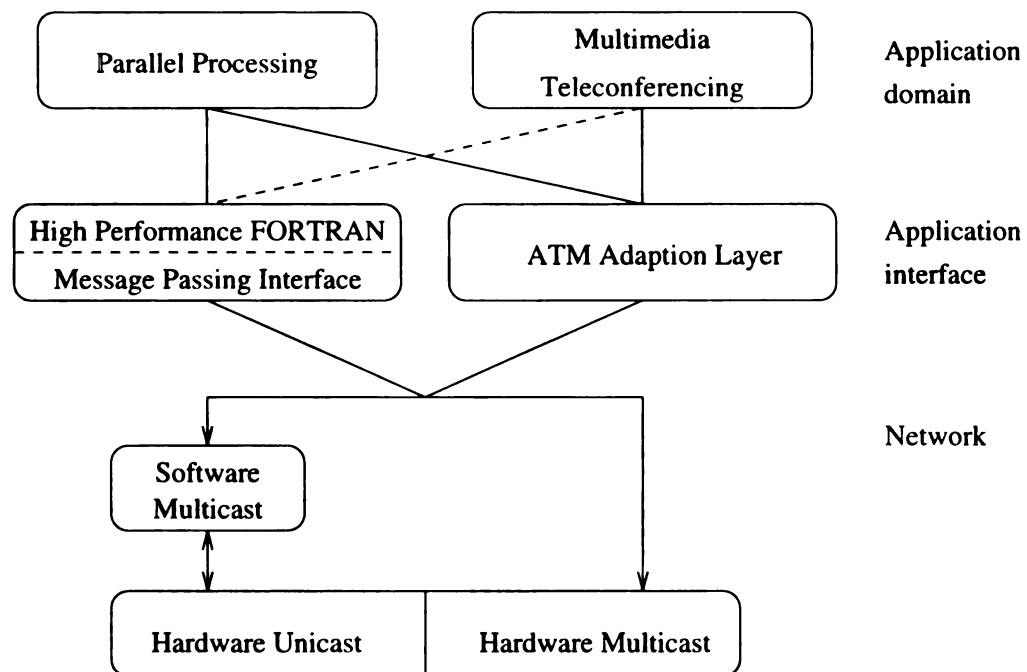


Figure 1.2: The framework

As shown in Figure 1.2, teleconferencing, multimedia and interactive TV are basically based on the ATM network. The current knowledge of multicast support of ATM switches is still limited. For example, both the Fore Systems ATM switch and the SynOptics ATM switch support multicast via high speed buses [9, 10].

The objective for this work is to support efficient multicast communication for all applications on top of a MIN. It is certainly network topology dependent. In this thesis, we will concentrate our effort in wormhole-switched unidirectional MINs.

1.1 Wormhole Switching

Switching methods greatly affect network latency. *Wormhole switching* [11], which has been adopted in almost all existing direct networks, is also being used in multistage networks. In wormhole switching, a message consists of a sequence of flow control digits, or *flits*. The header flit(s) of the message governs the route, and the remaining flits follow in a pipeline fashion. If the header flit(s) encounters a busy channel, other flits will hold and wait. This property makes wormhole switching susceptible to deadlock. As a result, deadlock avoidance is a critical issue in wormhole-switched networks. Because of its low network latency and the small amount of dedicated buffer space required at each node, wormhole switching has become the most promising switching technology and will be the only switching method considered in this thesis.

A message is divided into small packets, called *cell*, in the ATM network. The propagation of a cell within a switch is similar to wormhole switching [12]. Thus, the

proposed model could be applied to ATM switching with minor modifications.

1.2 Multistage Interconnection Networks

Multistage interconnection networks (MINs) are a popular class of interconnection architecture for constructing SPCs, such as the BBN GP-1000 [13] and TC-2000 [14], IBM SP-1[15] and SP-2, TMC CM-5 [16], Meiko CS-2 [17], and NEC Cenju-3 [18], as well as high speed networks, such as the SynOptics ATMX [19] and DEC GIGAswitch [20]. In such systems, processor nodes are interconnected through a MIN, a class of indirect networks. Each processor node has its own processor(s), local memory, and other supporting devices. As the number of nodes in the system increases, the total communication bandwidth, memory bandwidth, and processing capability of the system scales up as well. MINs can be further classified as unidirectional MINs and bidirectional MINs [21]. As indicated in [21], bidirectional MINs are essentially a fat tree, such as the TMC CM-5, IBM SP-1, and Meiko CS-2. This thesis concentrates on those unidirectional MINs, such as the NEC Cenju-3 [18] and the BBN TC-2000 [14].

1.3 Motivation and Problem Definition

Since the multicast communication service is the primitive basis for increasing performance of parallel processing applications and newly emerged applications, several research efforts and works have been done on direct networks, such as 2-D mesh

and hypercube, but very little in unidirectional wormhole-switched MINs. Basically, there are two different approaches for providing multicast services: the hardware implementation and the software implementation. No matter what topology a network employs, a multicast message header must carry all destination addresses so that routers can make appropriate routing decisions. The header information is an overhead to the system and should be minimized in order to reduce communication latency and to increase effective network bandwidth.

Multi-Address Encoding and Decoding

The *multiple address* (multi-address) encoding and decoding is overlooked or ignored by most researchers. The all-destination encoding scheme, which puts all destination addresses in the header, is the most intuitive approach and the one used by most researchers when they mention this issue [22]. Basically, such a scheme is fine when the number of destinations is small; however, when the number of destinations increases, the overhead of the header increases. Few other works have used the single region broadcast or single region mask which forces their multicast communication to be limited in a single contiguous region, *e.g.*, the NEC Cenju-3, or in a single sub-cube, *e.g.*, the nCUBE-2, respectively. To reduce the overhead of multi-address and eliminate such restrictions becomes the first challenge to us. A good multi-address encoding scheme should consider how to minimize the message header length overhead, how to reduce the header processing time (or routing decision making time), and how to support cut-through switching.

Hardware Approach

There are two different hardware approaches to support multicast: the path-based and the tree-based. As shown in Figure 1.3 and 1.4, a path-based multicast is basically a copy-and-forward technology, while the tree-based multicast is a replicate-and-forward technology. In a path-based approach, the router will replicate an incoming

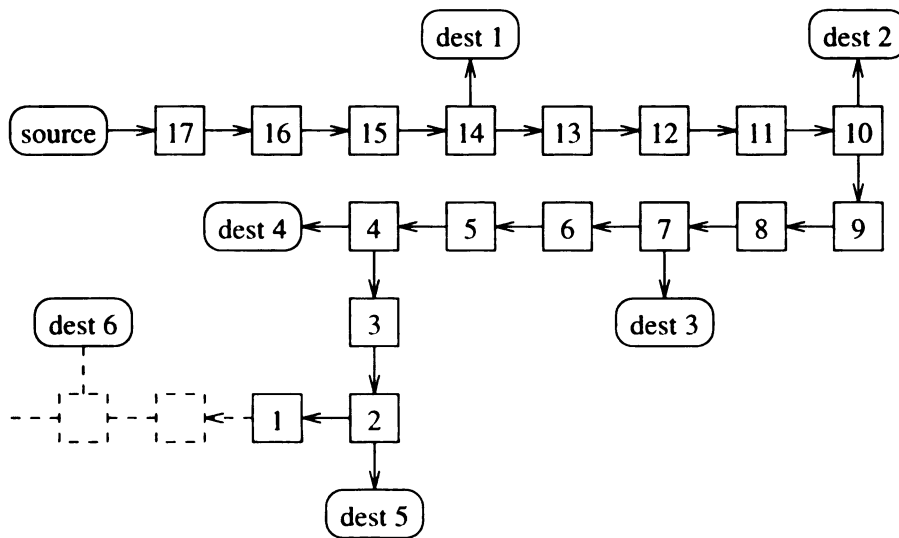


Figure 1.3: Path-based hardware multicast.

flit and forward one copy to the processor if the process is one of the destination while sending the other copy to the outgoing channel as shown in Figure 1.3. As shown in this figure, a path is established from the source node to all destinations in a certain order. Every destination is visited once and only once. The path-based approach is more suitable for a direct network since the router is connected to the processor directly. In a unidirectional MIN, the distance between any source and destination pair is constant. The most serious problem is that deadlocks become possible due to a self-blocking property.

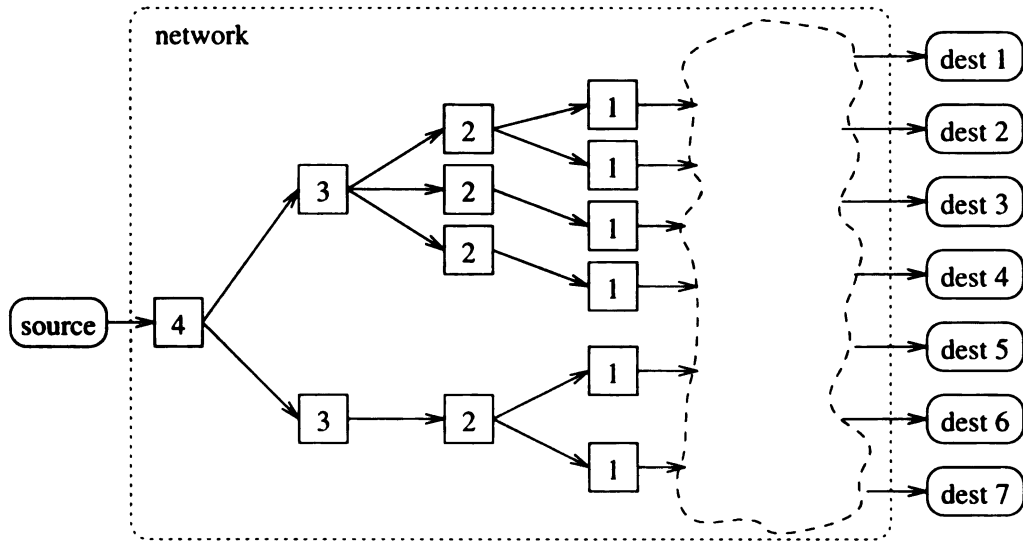


Figure 1.4: Tree-based hardware multicast.

The router in a tree-based network may replicate the incoming flits and then forwards them to multiple outgoing ports as shown in Figure 1.4. Intuitively, such an approach may cause deadlock if there are multiple multicast trees. Some restrictions must be applied to avoid deadlock when multiple multicast trees are allowed simultaneously. Such restrictions depend on the underlying network topology.

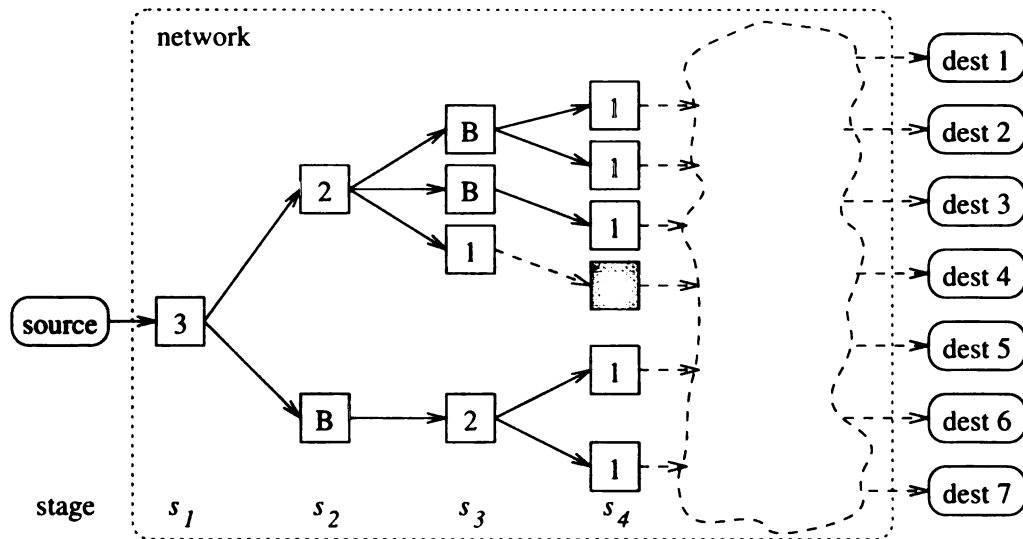


Figure 1.5: An asynchronous multicast tree.

Because multiple branches exist in a tree-based multicast, two different approaches

may be used when some branches are blocked. A switch in the *asynchronous* method forwards its flits independently as shown in Figure 1.5. As long as all associated switches in the next stage are ready to receive the next flit, the switch forwards the flit residing in its buffer. For example, flit 1 on the top switch at stage s_3 is forwarded to stage s_4 since those two associated switches in stage s_4 are ready. Flit 2 in the top switch at stage s_2 is blocked since not all three associated switches are ready. A buffer is left empty in a switch if the next flit is blocked and the previous flit is forwarded. Such an empty buffer is named a *wormhole bubble* and marked by B in Figure 1.5.

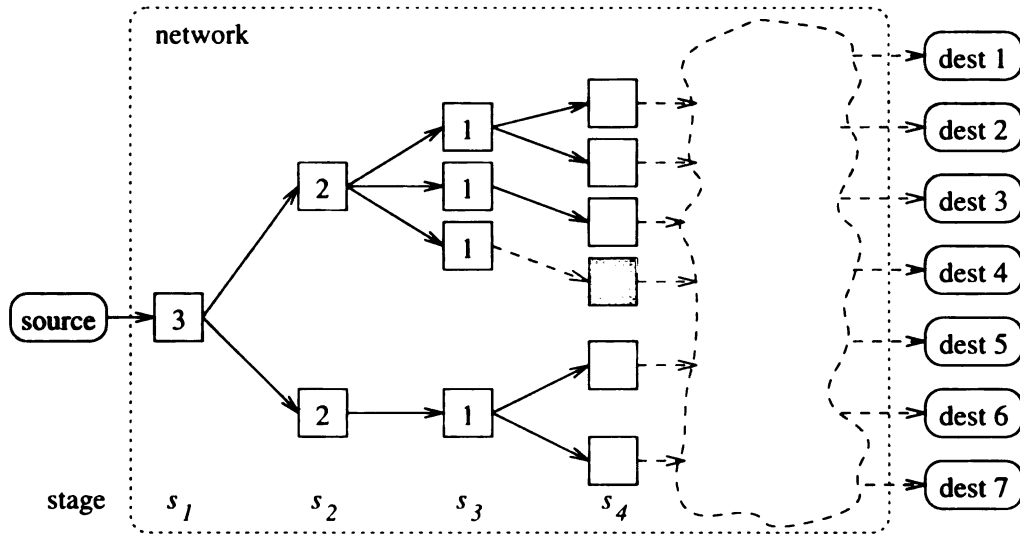


Figure 1.6: A synchronous multicast tree.

On the contrary, the synchronous multicast requires headers of all branches to be forwarded simultaneously. While any branch is blocked, all branches stay in current status as shown in Figure 1.6. In this figure, the fourth switch at stage s_4 is used by some other communication. The third switch at stage s_3 is unable to forward flit 1. Thus, it sends the condition back to the source node and the source node

forces every branch to stop forwarding. A wire-AND or wire-OR connection may be used to implement the control circuit to avoid long latency caused by the condition checking. Obviously, such a switch would be more complicated than the switch used in an asynchronous multi-head worm. It offers a potential solution for deadlock-free multicasts and is adapted by the nCUBE-2 [23].

Both a synchronous multi-head worm and an asynchronous multi-head worm have their own problems. To avoid deadlock in an asynchronous multi-head worm is very difficult, if not impossible. To restrict one multicast at a time is a possible solution but a high speed bus may offer the same performance. To enlarge the buffer within each switch is another solution if the buffer is large enough. Such a solution makes the network become a virtual cut-through network instead of a wormhole-switched network. Avoiding deadlock for multiple synchronous multi-head worms seems to be easier than for the asynchronous one, but it is not as easy as we think. For example, the nCUBE-2 has a synchronous multi-head worm but is forced to disable the hardware multicast due to potential deadlock. The NEC Cenju-3 is another system which supports the hardware multicast. Nonetheless, deadlock is still possible when multiple multicasts are allowed due to its asynchronous multi-head worm. Consequently, to provide hardware multicast capability without potential deadlock and/or starvation is one of the objectives of this work.

Software Approach

In systems which do not support hardware multicast, we have to resort to software approaches to support efficient multicast, referred to as *software multicast* or *unicast-based multicast*. Traditional approaches use *separate addressing* in which the source node sends the message to one destination at a time. As the number of destinations increases, separate addressing may require excessive time because many systems allow a local processor to send only one or a few messages at a time. An alternative approach is a *multicast tree*¹ in which the source sends the message to a subset of the destinations. Each recipient of the message forwards it to some subset of the destination that has not yet received it. Which type of multicast trees to use depends on the switching strategy and the unicast routing algorithm. An efficient multicast tree involves no local processors other than the source and destination processors, exploits the distance-insensitivity of wormhole switching, and is of minimum height, specifically, height $\lceil \log_2(m+1) \rceil$ for m destination nodes. Another key requirement is that there be no channel contention among the constituent messages of the multicast. That is, the unicast messages involved should not simultaneously require the same channel.

¹The software multicast tree is different from hardware multicast trees which are based on cut-through switching. In a software multicast tree, each intermediate node has to receive the complete message before forwarding to other nodes.

Multicast Communication in MINs

Multicast communication has been extensively studied for distributed-memory multi-computers based in direct network architectures, but little research has been done in indirect networks, especially MIN-class topologies. There are several characteristics which distinguish a MIN from other network architecture such as unique length between any (source, destination) pair, and $O(N \log N)$ hardware complexity, *etc.* Since those multicast approaches proposed for direct networks are not suitable for MINs, this thesis concentrates on multicast communication in MIN-based networks.

A regular MIN with N inputs and $\log_k N$ stages, where k is the number of input/output of a switch, is a *unique-path* network. With such property, deadlock becomes possible if multicast trees share more than two channels. The multicast capability in current systems is provided by either the unicast-based (software-based) approach or the hardware approach with excessive hardware and some limitations in order to avoid deadlock. Both approaches suffer long latency, inefficiency, or cost. Even worse, some of them are not deadlock-free. A general multicast algorithm to avoid these drawbacks and guarantee deadlock-free system is the major focus of this thesis.

A regular MIN is a unique-path network; hence, the multicast tree is also unique unless some variants of the MIN topology are considered. One such design extension, among a few others, is to consider a MIN with extra stages. The idea is to attach a few extra stages to the regular n -stage MIN. The extra stages bring forth the flexibilities in multicast paths and the difficulties. To find all traffic-optimal multicast trees in

an ESMIN becomes an NP problem.

1.4 Performance Metrics

An important metric used to evaluate a network is its communication latency, which is the sum of three component values: *start-up latency*, *network latency*, and *blocking time* [24]. The *start-up latency* refers to the time required for message framing/unframing, memory/buffer copying, validation, and so on, at both source and destination nodes. The startup latency is mainly dependent on the design of system software within the nodes and the interface between nodes and routers. Start-up latency can be further classified into *sending latency*, the software latency at source node, and *receiving latency*, the software latency at destination node. The *network latency* equals the elapsed time after the head of a message has entered the network at the source until the tail of the packet emerges from the network at the destination. Given a source and destination node, the startup and network latencies are static values, frequently used to characterize contention-free networks. The *blocking time* includes all possible delays encountered during the lifetime of a message. These delays are mainly due to conflicts over the use of shared resources, such as busy channels and filled buffers. Blocking time reflects the dynamic behavior of the network due to the passing of multiple messages and may be high if the network traffic is heavy or unevenly distributed. In this dissertation, *multicast latency* is used to measure multicast performance. The multicast latency refers to the time interval from when the source processor begins to send the first copy of the message until the last destination

processor has received the message.

1.5 Objectives and Thesis Outline

The main objective of this research is to establish efficient multicast communications in unidirectional wormhole-switched MINs. To support efficient multicast communication, the header overhead of the multicast message should be minimized and external network contention among different applications should be eliminated. These objectives and the thesis outline are discussed in the following.

In Chapter 2, we discuss the issue to shorten the header overhead by investigating different multi-address encoding and decoding schemes. Unlike a unicast message, the header of a multicast message must carry multiple destination information. Because of that, the length of a header varies with the number of destinations, the distribution of destinations, and the multi-address encoding/decoding schemes. The consideration of such a header as well as six multi-address encoding and decoding schemes are addressed in this chapter for general network topology. As the scale of networks increases and the demand of multicast communication increases, the overhead of message header becomes critical, which implies that multi-address encoding becomes critical. Although the proposed multi-address encoding and decoding schemes can be applied to networks with different switching techniques, such as circuit switching, store-and-forward switching, and cell relay, the emphasis of this thesis will be in the wormhole-switched technique. Depending on characteristics of a network topology, these multi-address encoding and decoding schemes may be further optimized.

In Chapter 3, we brief review the different interconnection patterns as well as associated network topologies of MINs. Furthermore, the optimization of the multi-address encoding schemes and the associated decoding algorithm in a switch for each encoding scheme in MINs are also addressed in this chapter. Performance evaluation of different multi-address encoding and decoding schemes is given based on simulations.

In Chapter 4, two different tree-based hardware multi-head worm implementations are studied — the asynchronous and synchronous. The asynchronous multi-head worm allows that each branch forwards independently while the synchronous multi-head worm insists that all branches forward synchronously. Unfortunately, both implementations are not deadlock-free unless certain rules are applied. Hence, current hardware implementations exhibit either undesirable properties or are restricted in their use. A deadlock-free and starvation-free hardware implementation of multiple synchronous multi-head worms is presented. The difficulty to the implementation of deadlock-free multiple asynchronous multi-head worms is also shown in this chapter.

Unlike the unicast message whose header has a fixed size, the size of a multicast message header depends on the number of destinations, the distribution of destinations and the encoding scheme. Since a single flit may not be able to carry all destination information, a multi-head worm may have different distances of branches toward different destinations. Such latency may cause deadlock of synchronous multi-head worms. We address this issue and propose a pseudo multi-address encoding scheme to eliminate the potential deadlock.

In Chapter 5, the network partitionability of different delta class networks is ex-

amined and performance results of various allocation schemes is given. By exploiting the locality of processor allocation, we show that the known topologically equivalent delta-class MINs have different capabilities in supporting hardware and software multicasts. Since the internal contention is unavoidable unless doing one multicast at a time, we concentrate on eliminating the external contention. A binary cube network partitioning scheme is studied and shown to be external contention-free in both cube and omega MINs. On the contrary, we also show that the baseline and butterfly MINs may not be partitioned into contention-free and channel-balanced subcubes. Two connection patterns are studied to modify baseline and butterfly MINs. Both modified baseline and butterfly MINs have the same partitionability as cube and omega MINs.

In Chapter 6, we consider the multicast problem for various MIN classes of topologies. Since the multicast problem for a regular MIN has a unique solution and does not offer any flexibility, we consider a design extension, namely *extra-stage MIN* (ESMIN), as our focus. The ESMIN multicast problem is formulated and an optimality criterion is defined. A lower bound on the number of multicast trees is estimated, and we show that the total number of traffic-optimal multicast trees may itself be exponential. However, a traffic-optimal multicast tree can be generated in polynomial time. We propose an algorithm towards this. The performance of this algorithm, with respect to three other proposed heuristics, is shown using simulation.

In Chapter 7, an efficient software-based multicast algorithm is presented and analyzed for those existing message-based SPCs which do not have hardware multicast support. One way to implement multicast in such systems is *separate addressing*. An

alternative approach is *multicast tree*. The focus here is on such multicast tree implementation, also known as *unicast-based* multicast implementation. By exploiting the locality of processor allocation, a minimum-time unicast-based multicast algorithm is proposed for some classes of MINs which support one-port communication.

In Chapter 8, we give an overview of related work. Since lots of research has devoted to multicast on different network topologies and different switching techniques, we concentrate on the work related to the multicast in MIN-based ATM switches as well as the hardware multicast and the software multicast in wormhole-switched networks. The concluding remarks as well as some possible future research directions are discussed in Chapter 9.

CHAPTER 2

Multi-Address Encoding

No matter what kind of topology a network is, a message header for multicast communication must carry the destination set information needed for routers/switches to make appropriate routing decisions. In some networks, such a header is needed in a sender-initiated control message in order to establish a multicast circuit; while in some other networks, such a header is used in all data messages. Nevertheless, the header information is an overhead to the system and should be minimized in order to reduce communication latency and to increase effective network bandwidth.

Basically, a multicast message header carries multiple destination addresses (multi-address) information. A destination address can be either a physical address or a relative address. A relative address is usually used to represent the relative location to the source address. Each address may be further represented by a number of dimensions. For example, in a 2D mesh network, each address may have two dimensions — one for each dimension. In this work, we don't further distinguish different types of addresses. Unless otherwise specified, an address refers to all details of the

address in our work.

2.1 Header Encoding Design Considerations

Each multicast message header must have a number of flits to carry the necessary routing information, where each flit is a flow control unit. It could be an address or a region (to be discussed later) depending on the encoding scheme. The total number of flits or header length is usually not only dependent on the number of destinations but also on the selected multi-address encoding scheme. The function of switches is taking a message from an input channel, making a routing decision, and forwarding (with possibly replicating) the message to one or more output channels. A good multi-address encoding scheme should not only shorten the message header length, but minimize communication latency and ease routing decisions. The following header design issues are considered.

The length of a message header can be either fixed or variable in terms of the number of destinations. The length of a variable length header is dynamically adjusted by switches as the message header is processed. For the fixed one, the length may be a function of the location of switches in the network.

In cut through switching, should each switch buffer the whole message header before making the routing decision? For a large message header, it implies a large buffer to hold the whole message header and a longer communication latency. In wormhole routing, a message is divided into a number of flits, and the minimum capacity of each buffer is one flit. Since a message header may be composed of many

flits, it is desirable that the routing decision in each router can be made as soon as possible to minimize the buffer requirement. Ideally, a message header can be processed on the fly on the flit basis.

For a variable length header, the number of destination addresses (or regions to be discussed later) is another design parameter. It may be impractical and inefficient to use a counter to indicate the number of destinations (or regions) because the counter flit is usually placed at the beginning of a message header. Since the value of counters may be changed by switches if the destination address set is split into different subsets, it will prohibit the processing of message headers on the fly by switches. An alternative approach is to have an *end-of-header* (EOH) flit to indicate the end of an address header. Some known hardware and software techniques, such as code violation and address stuffing, may be used to implement the EOH flit.

Both tree-based multicast, in which a router is able to replicate an input message through multiple output channels, and path-based multicast, in which a message traverses along a certain path picked by the destination nodes along the path, may be considered to implement multicast communication [25]. Usually, path-based multicast is used in direct networks, such as multi-dimensional meshes, and tree-based multicast is used in indirect (switch-based) networks, such as MINs. In both approaches, the message header may have to be modified by those intermediate routers/switches.

2.2 Multi-Address Encoding Schemes

Six multi-address encoding schemes are described in this section. The message header format of these schemes is shown in Figure 2.1.

(a) All Destination Encoding

$addr_1$	$addr_2$	$\dots\dots\dots$	$addr_m$	EOH
----------	----------	-------------------	----------	-------

(b) Bit String Encoding

b			$\dots\dots\dots$	e
-----	--	--	-------------------	-----

(c) Multiple Region Broadcast Encoding

$b_1:e_1$	$b_2:e_2$	$\dots\dots\dots$	$b_k:e_k$	EOH
-----------	-----------	-------------------	-----------	-------

(d) Multiple Region Stride Encoding

$b_1:e_1:s_1$	$b_2:e_2:s_2$	$\dots\dots\dots$	$b_k:e_k:s_k$	EOH
---------------	---------------	-------------------	---------------	-------

(e) Multiple Region Mask Encoding

$b_1:e_1:m_1$	$b_2:e_2:m_2$	$\dots\dots\dots$	$b_k:e_k:m_k$	EOH
---------------	---------------	-------------------	---------------	-------

(f) Multiple Region Bit String Encoding

$b_1 : e_1 : T_1$	$b_2 : e_2 : T_2$	$\dots\dots\dots$	$b_k : e_k : T_k$	EOH
-------------------	-------------------	-------------------	-------------------	-------

Figure 2.1: The message header format of six address encoding schemes.

2.2.1 All-Destination Encoding

This is an intuitive method used in [26], in which all destination addresses are carried by the header as shown in Figure 2.1(a). Assuming that all addresses are sorted in ascending order, the EOH flit can be all 0's. If the only destination address is all 0's, the second all 0's indicates the EOH. If the routing requires that the addresses

be arranged in a certain order, such as path-based multicast [25], the EOH flit may be represented by replicating the last address. However, for tree-based multicast, it is easier to generate the same EOH flit for all replicated outgoing messages. In this case, the EOH flit could be an unused address or use other methods. Note that if an EOH flit takes many addresses, the flit buffer must be large enough to hold the EOH flit. Clearly, all-destination encoding is good for a small number of irregular addresses as its header length is proportional to the number of addresses.

2.2.2 Bit String Encoding

The major drawback of the all-destination encoding scheme is its significant header overhead when the number of destinations is large. One way to limit the size of a header is to have a bit string to indicate destinations, where each bit corresponds to a destination ranged between node b and node e as shown in Figure 2.1(b). Since the number of nodes in a system is predefined, there is no need of an EOH field. In some network topologies, such as MINs, the bit string length can be a function of the number of reachable nodes from a given switch, which is still independent of the number of destinations. Apparently, the bit string encoding scheme is inefficient when the system is large and the number of destinations is small. However, it is flexible in handling a large number of irregular destination addresses.

Usually, it is extremely difficult for a switch to make the routing decision on the fly based on the incoming bit string information and to produce the bit string information for each output port. Thus, a switch usually has to buffer the entire bit

string in order to make the routing decision and to generate output bit strings. This is named *buffered bit string*. Although the buffered bit string encoding scheme allows distributed routing, each switch requires a large flit buffer and the communication latency is also increased.

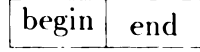
To eliminate a large flit buffer, a *hierarchical bit string* may be used. This is a source routing method in which the source node has to determine the complete multicast tree information. In this scheme, the input message header has $1 + k$ flits for a switch with k output ports. The first flit has k bits corresponding to k output ports. A “1” in a bit position indicates that the corresponding output port should forward a copy of the message. The remaining k even sized flits carry bit string information for each output port. The bit string information is recursively defined because each bit string becomes an input to the next switch. Note that if these bit string flits are not even sized or have less than k flits, it will be very difficult to determine the delimiter between two adjacent flits. Obviously, the hierarchical bit string encoding scheme can perform routing on the fly and requires a small flit buffer in each switch. However, the header length is much longer than the buffered bit string method.

2.2.3 Multiple Region Broadcast

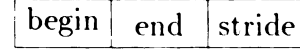
In order to enforce communication locality and minimize communication interference among processors from different process groups, processors belonging to the same process group are usually allocated in a contiguous region, if possible. Thus the

multicast addresses can be confined within a region, and each region is specified by two fields: $(b:e)$, the beginning and ending addresses of the region, respectively. This is referred to as *region broadcast* and is used in some ATM switches and the NEC Cenju-3 [18].

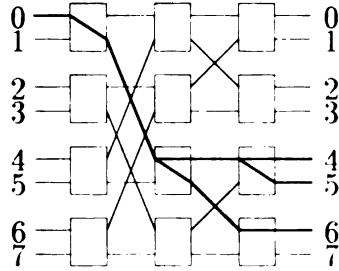
(a) Single region broadcast



(b) Single region stride



(c) An example of single region broadcast



(d) An example of single region stride

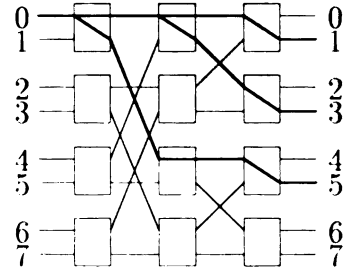


Figure 2.2: Single region broadcast and single region stride, where the source node is 0 and the destination sets are $\{4, 5, 6\}$ and $\{1, 3, 5\}$, respectively.

Figure 2.2(a) shows the specification of a single region broadcast, where the beginning address must be no greater than the ending address. An example is given in Figure 2.2(b). When a message enters a switch, it buffers the complete region flit and then directs the message to corresponding output port(s). The header is revised at every switch where the message is replicated. For example, the incoming header at the shadow switch is $(4:6)$, and the outgoing headers for upper and lower ports are $(4:5)$ and $(6:6)$, respectively.

However, depending on the processor allocation scheme and application program characteristics, not all process groups can have all their nodes in a contiguous region.

The single region broadcast thus cannot achieve the multicast in a single multicast communication. Two approaches may be used. One method is to send multiple single region broadcast messages in sequence to different disjoint regions. Another method is to add extra hardware (e.g., the copy network used in [27, 28]) and to introduce the concept of dummy addresses. Here, we generalize this scheme to multiple regions by allowing multiple region specification in the header.

The multiple region broadcast, see Figure 2.1(c) for its header format, forwards the message to every node covered by each region. The EOH flit can be specified as a region containing an address with all 1's followed by an address with all 0's (i.e., address violation). In fact, any pair of addresses can be an EOH as long as the second address is smaller than the first address.

Figure 2.3(a) illustrates an example of multiple region broadcast. Consider 64 nodes organized as a 2D array. For some 2D matrix applications, it may require the source node (3,2) to send a message to all nodes in the same column and in the same row. If the source node can also be a destination, the header for node (3,2) is (3:3,0:7; 0:7,2:2). Otherwise, the header will be (3:3,0:1; 3:3,3:7; 0:2,2:2; 4:7,2:2).

2.2.4 Multiple Region Stride

In some applications, a source node may wish to send a message to all odd-numbered destinations or to all even-numbered destinations. In other words, the destination addresses have a constant distance between two adjacent addresses. Thus, the addresses (or flit) can be specified by three parameters: ($b:e:s$), the beginning address,

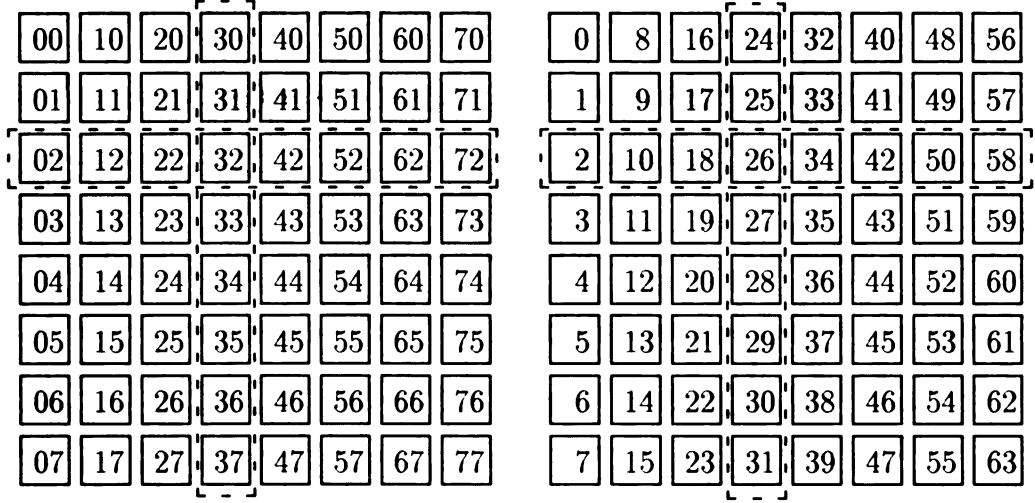


Figure 2.3: Multicast to the same row and column in (a) 2D mesh and (b) linear array architectures.

the ending address, and the stride value. Note that for a multi-dimensional address, each dimension may have its own stride value. This encoding scheme is referred to as *region stride multicast*.

Figure 2.2(c) shows the specification of a single stride-region. Consider a stride-region, $(b:e:s)$, where $(b \leq e)$ and s is the stride value. The message is forwarded to node $\{d | d = b + i \times s, i = 0, 1, \dots, \lfloor (e - b)/s \rfloor\}$. Figure 2.2(d) shows an example in which destinations are $\{1, 3, 5\}$ and the source node is 0. The header can be either $(1:5:2)$ or $(1:6:2)$. The header is also revised when the message is replicated. For example, the incoming header is revised to $(1:3:2)$ and $(5:5:2)$ for the outgoing messages to upper and lower ports of the shadow switch, respectively.

Similarly, this encoding scheme can be generalized to *multiple region stride*. The header format is shown in Figure 2.1(d). To be consistent with the stride-region flit specification, the EOH flit may contain three fields. Like multiple region broadcast, an address with all 1's followed by an address with all 0's may be used to indicate

the EOH, where the third stride field is irrelevant.

Consider the same multicast example illustrated in Figure 2.3(b). If the processors are organized in a linear order, such as the NEC Cenju-3 and the IBM SP-1, all nodes in the same row are not in a contiguous region. The multiple region stride encodes these destinations as (2:58:8; 24:31:1) if the source node is also a destination; otherwise, the header is (2:18:8; 34:58:8; 24:25:1; 27:31:1).

2.2.5 Multiple Region Mask

Another regular pattern of destination addresses that typically occurs in k -ary n -cube networks is subcube. Let $d_{n-1} \dots d_1 d_0$ represent a node in a k -ary n -cube, where $0 \leq d_i \leq k - 1$ (note that this definition can be easily extended to a more general cube network in which the radix of each dimension may be different). Any subcube can be described by a binary mask, m , with an address, b . The binary mask m defines the size and dimensions of the subcube, i.e., the number of ones in m and the location of each dimension with $m_i = 1$. The address b defines which subcube among those subcubes, i.e., b_i 's where $m_i = 0$. This approach is used in the nCUBE-2 [23]. In a more general case, we define a subset of a subcube with three fields: $(b:e:m)$, where $b_i = e_i$ when $m_i = 0$ and for those i 's with $m_i = 1$, b_i and e_i specify the lower and upper bounds of the subcube, respectively. The multiple region mask shown in Fig. 2.1(e) extends this approach to allow that a message be destined for multiple subsets of subcubes.

Consider a binary 4-cube with eight destinations {0100, 0101, 0110, 0111, 1100,

1101, 1110, 1111}. The multiple region mask encodes the header to (0100: 1111:1011), which is a complete 3-cube. When the destinations are {0110, 0111, 1100, 1101, 1110}, a subset of the 3-cube is specified as (0110:1110:1011).

2.2.6 Multiple Region Bit String

Both multiple region broadcast and multiple region stride are suitable for destinations that can be divided into a number of clusters and with a regular address pattern within each cluster. If the destinations are not in any regular shape, the header encoded by previous encoding schemes may be too large (e.g., many disjoint regions). The multiple region bit string encoding scheme shown in Figure 2.1(f) is proposed to reduce the header overhead under such a situation. In general, a region bit-string flit is specified as (b, e, T) , where b and e ($b \leq e$) indicate the beginning and ending addresses, and T is a binary bit string. Each bit in T corresponds to a node within b and e . A node is a destination if the corresponding bit is 1. Both b and e are destinations and T has $(e - b + 1)$ bits¹.

Consider a system with 16 nodes and the destinations are {0,1,3,4,6,12,13,15}. With multiple region broadcast, it requires five regions. With multiple region bit string, one possible specification is (0:6:1101101:12:15:1101). Apparently, in multiple region bit string, the length of each region is not fixed and is determined by the values of b and e . Depending on the flit buffer capacity, the system has to limit the maximum value of $e - b$. The EOH flit can contain two fields — a field with all 1's

¹Since the first and last bits are always 1 in T , these two bits may be removed from T .

followed by one with all 0's. Although the multiple region bit string is more flexible and can handle addresses with irregular patterns, it may be difficult to determine the most efficient number of regions, and the switch design may be more complicated.

2.3 Multi-Address Decoding

Consider a generic network switch/router with k input ports and k output ports as shown in Figure 2.4(a) with $k = 4$. The interconnection of routers defines the network topology. Each router may or may not have a local processor depending on the system architecture. When a multicast message arrives at a router via an input port, the router examines the header and may enable a number of output ports depending on the routing strategy. The message is replicated and forwarded to all enabled output ports. A critical issue is how to define new header information for each enabled output port. If not handled properly, the same message may be received repeatedly by the same processor as shown in Fig. 2.4(b).

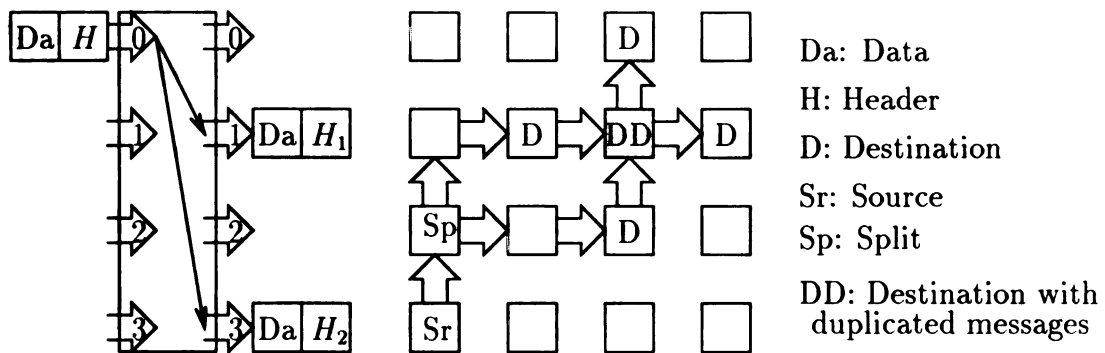


Figure 2.4: (a) A generic switch/router with four input/output ports, and the message is forwarded via both output ports 1 and 3. (b) The receiving of a duplicated message when the header is not handled properly and alternate routing paths are allowed.

In distributed routing, if a router is able to process the input message header

on the fly, it will buffer one header flit, make the routing decision, and deposit new message header flit information to a selected output port. When an address or a region is forwarded to its associated output port, the other enabled ports will be idle since there is no address or region to be forwarded to these output ports. Such an idle on timing is referred to as *wormhole bubble*. Note that a wormhole bubble does not necessarily imply an empty or a null flit. With self-timed design, a wormhole bubble implies a timing delay to the next flit. However, for ease of explanation, the wormhole bubble is represented as an individual flit in the following figures as B.

When a router replicates a message, the message header will be revised and an output message header is generated for each enabled output port (or each replicated message). Figure 2.4(a) shows an example replicating a message into two messages in which the headers $H1$ and $H2$ are different. Usually, the destination address set is divided into a number of disjoint address subsets, one for each replicated message. This approach can avoid multiple receptions of the same message, which is especially important if there are many alternate paths to a destination. The drawback of having disjoint subsets is its decreased flexibility in forming the header format. However, if the routing path is unique, those destination subsets may overlap. Since a router usually makes its routing decision based on the first field of a region flit, it may ignore any destination which is not reachable from its output port. This approach is more flexible in forming the header format.

For a single region which contains a number of addresses, the region may be split into several regions by a router. Each region is directed to an enabled output port. It is possible that two or more disjoint regions are directed to the same output port.

Thus, the number of regions in the new header may be larger than the incoming one. Therefore, placing the number of regions in the header may not be feasible.

For each of the six multi-address encoding schemes, this section will describe some generic address decoding algorithms that may be performed by a router. Given a header flit, say F , the routing function $Route(F)$ will determine the output port number to be selected to forward the message. Furthermore, we assume that if the processor, if any, associated with a router is one of the destinations, a copy of the message will be sent to the processor. An output port is enabled if a copy of the message is to be forwarded through the port. At the end of message transmission, all output ports will be disabled. These behaviors will not be further described in the following algorithms.

2.3.1 All Destination Decoding

For each incoming flit, d , Figure 2.5 shows the corresponding decoding algorithm. When sending information through an enabled port, it implies that the corresponding output channel is available; otherwise, the message transmission will be pending until the port is available. Avoiding deadlock is another critical design issue. The solution is dependent on the network topology and routing strategy, which is beyond the scope of this chapter.

For example, the addresses of a message into a router are $\{2, 3, 5, 7, 15, B, B\}$ as shown in Figure 2.6, where B indicates wormhole bubble and E represents EOH. Let the reachable nodes from port i be node $i \times 4$ to node $i \times 4 + 3$, where $0 \leq i \leq 3$.

Algorithm: all-destination decoding

Input: An address d .

Output: append d to the header of the selected output port.

Procedure:

begin

if $d = EOH$ then send EOH to all enabled ports; exit;

$j := \text{Route}(d)$;

send d to port j ;

end

Figure 2.5: All destination decoding algorithm.

We further assume that there are 16 nodes in the system. The router enables port 0 and forwards address 2 when address 2 is decoded. Address 3 is forwarded to port 0. While address 5 is decoded, port 1 is enabled and address 5 is forwarded to port 1. There is a wormhole bubble on port 0. The process is repeated until the header is split completely as shown in the figure. Port 2 is not enabled since there is no destination via this port.

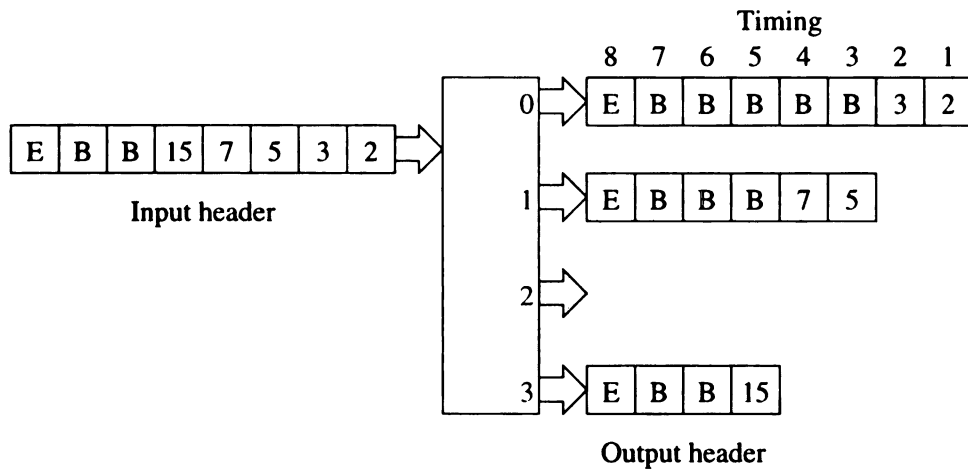


Figure 2.6: An example of all-destination decoding.

2.3.2 Bit String Decoding

Buffered Bit String.

A router stores the entire bit string and then detects each bit which has a 1 to enable the corresponding output port. If there is more than one output port that can reach the destination, the router selects exactly one port. The length of each bit string is dependent on the corresponding port and the number of nodes reachable from that port. The router also knows the reachable node for a given bit position. The buffered bit string decoding algorithm is given in Figure 2.7 while an example based on the previous example is shown in Figure 2.8.

<p>Algorithm: buffered bit string decoding</p> <p>Input: Binary bit string, T.</p> <p>Output: Bit string, D_j, for each enabled output port j.</p> <p>Procedure:</p> <p>begin</p> <p> for every bit, t_i, in bit string T do</p> <p> $(j, k) := Route(i)$;</p> <p> (* route through port j, bit position k *)</p> <p> $d_{j,k} := t_i$;</p> <p> endfor</p> <p> enable port j with $D_j \neq 0$ for all j;</p> <p>end</p>

Figure 2.7: Buffered bit string decoding algorithm.

Hierarchical Bit String Decoding.

In hierarchical bit string, the first field (k bits) of an incoming bit string indicates the enabling or disabling of the k output ports of a k -port router. The following bits are divided into k fields, one for each output port. The i^{th} field is forwarded to the

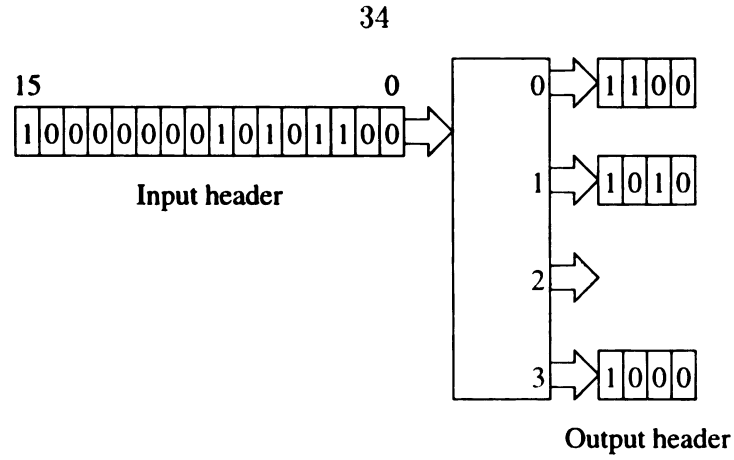


Figure 2.8: An example of buffered bit string decoding.

associated router in the next stage if port i is enabled, for $0 \leq i \leq k$. Otherwise, it is eliminated at the router. The decoding algorithm is too simple to be described here, and the corresponding example is shown in Figure 2.9.

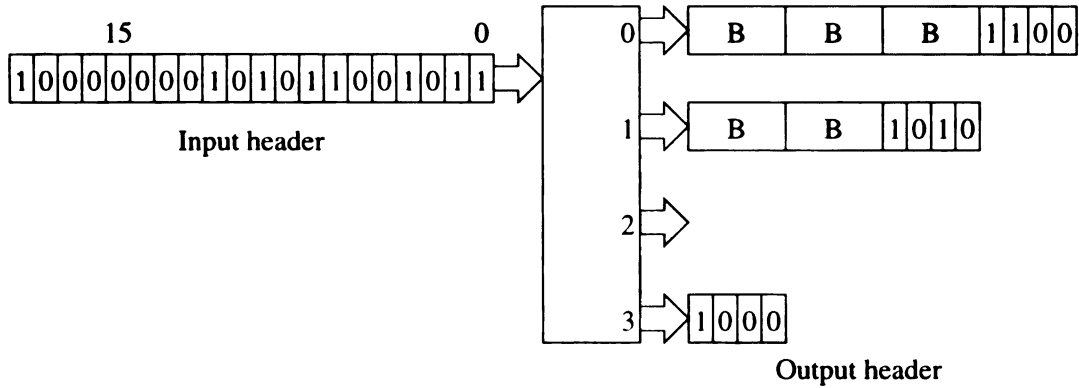


Figure 2.9: An example of hierarchical bit string decoding.

2.3.3 Multiple Region Broadcast Decoding

As shown in Figure 2.1(c), the header in multiple region broadcast contains several regions. All nodes covered by all regions should receive a copy of the message. Thus, the router may have to divide a region into several sub-regions, and each sub-region is directed through an appropriate output port. The algorithm for decoding each region

is given in Figure 2.10.

Algorithm: multiple region broadcast
Input: A region (b, e) .
Output: Send each sub-region to an output port.
Procedure:
begin
 if $(b, e) = \text{EOH}$, send EOH to all enabled ports; exit;
 while $(b \leq e)$ do
 $(j, e') := \text{Route}(b, e)$;
 send region (b, e') to port j ;
 $b = e' + 1$;
 end while;
end;

Figure 2.10: Multiple region broadcast decoding algorithm.

In this decoding algorithm, the router checks the beginning address of a region and searches for the first address, e' , which cannot be reached by the same output port. A sub-region is then identified for that output port. The process repeats until all sub-regions have been identified. For example, consider an incoming header of $(2:5; 7:8)$ shown in Figure 2.11. The router splits the first region to $(2:3)$ and $(4:5)$ for ports 0 and 1, and the second region to $(7:7)$ and $(8:8)$ for ports 1 and 2, respectively.

2.3.4 Multiple Region Stride Decoding

The method to handle multiple region stride is similar to that of multiple region broadcast. A router divides the region into several sub-regions, where each sub-region is directed to a single output port. As indicated in Figure 2.1(d), there is an additional field, stride, in each region to indicate the distance between two adjacent nodes. The decoding algorithm is similar to Fig. 2.10, except replacing (b, e) and (b, e')

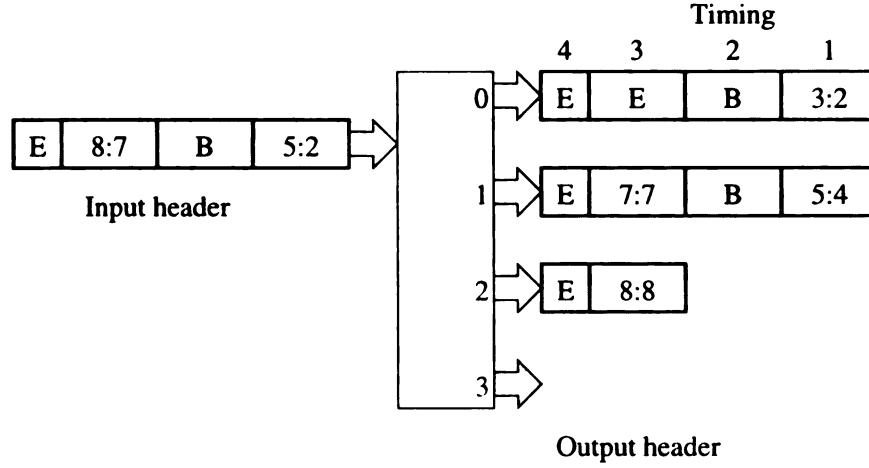


Figure 2.11: An example of multiple region broadcast decoding.

by (b, e, s) and (b, e', s) , respectively. Note that the stride value is never changed.

Consider the example in Figure 2.12. The incoming header is $\{(1:5:2), (6:10:1)\}$, which indicates that nodes 1, 3, 5, 6, 7, 8, 9, and 10 are destinations. The router splits the first region to $(1:3:2)$ and $(5:5:2)$ for ports 0 and port 1 and the second region to $(6:7:1)$ and $(8:10:1)$ for ports 1 and 2, respectively.

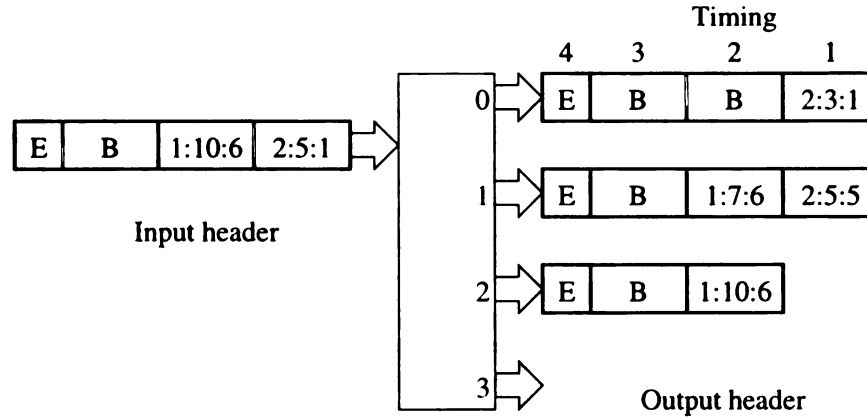


Figure 2.12: An example of multiple region stride decoding.

2.3.5 Multiple Region Mask Decoding

The method to handle multiple region mask is similar to that of multiple region broadcast and multiple region stride. A region is divided by the router into several sub-regions, where each sub-region is directed to an associated output port. Not only the beginning and ending address but also the mask will be changed by the router to avoid duplicated receiving.

An example of a binary 4-cube is given in Figure 2.13. The input header specifies six destinations {0000, 0001, 0010, 0011, 0100, 0101}. The router forwards a copy of the message to port 0, which defines a 2-cube with header (0000:0011:0011), and another copy to port 1, which defines a 1-cube with header (0100:0101:0001).

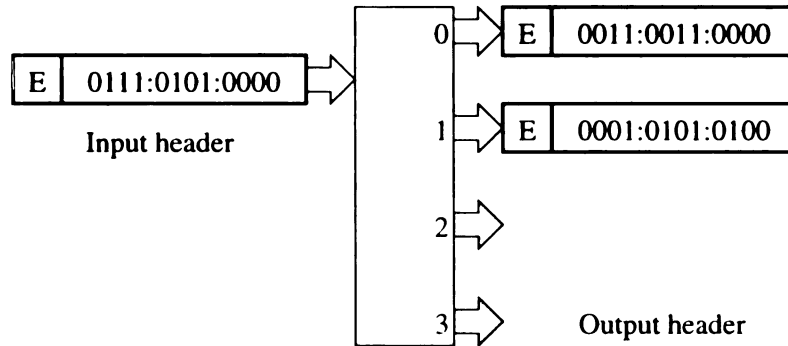


Figure 2.13: An example of multiple region mask decoding.

2.3.6 Multiple Region Bit String Decoding

As shown in Figure 2.1(f), there are three fields in a region. The decoding algorithm is also similar to that of multiple region broadcast except for the representation of regions. The tricky part is in the partitioning of a region into sub-regions. When a single output port can reach several non-contiguous regions, the router can mark

th

Th

are

lin

those bits that correspond to unreachable nodes to 0 instead of splitting the region.

The corresponding algorithm is shown in Figure 2.14.

Algorithm: multiple region bit string
Input: A region (b, e, T) .
Output: Send each sub-region to an output port.
Procedure:
 begin
 if $(b, e, T) = \text{EOH}$, send EOH to all enabled ports; exit;
 while $(b \leq e)$ or $(T = 0)$ do
 $(j, e', T') := \text{Route}(b, e, T)$;
 send region (b, e', T') to port j ;
 $t_i := 0$ if t_i is covered by T' ;
 $b :=$ corresponding node of the first non-zero t_i ;
 end while;
end;

Figure 2.14: Multiple region bit string decoding algorithm.

Let an incoming header be $(2:7:110101)$ which indicates that nodes 2, 3, 5, and 7 are destinations, as shown in Figure 2.15. The router enables port 0 since bits 0 and 1 in the input bit string are 1's; it enables port 1 since bits 3 and 5 are 1's.

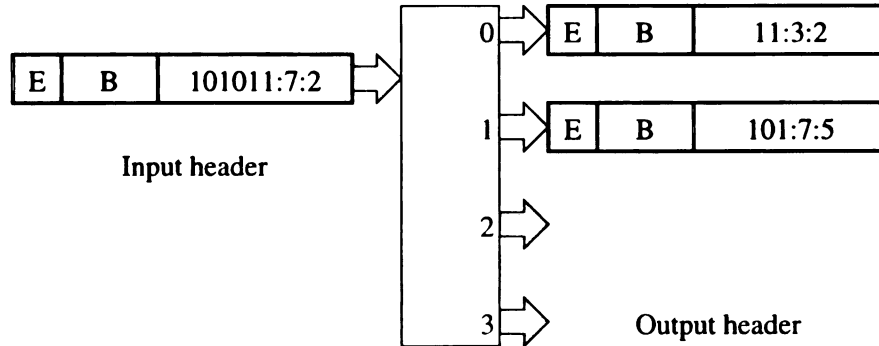


Figure 2.15: An example of multiple region bit string decoding.

2.4 Summary

As the scale of networks gets larger and the demand of multicast communication gets higher, the overhead of message header is becoming critical, which implies that multi-address encoding is becoming critical. Several multi-address encoding and decoding schemes have been investigated and explored in this chapter. Although the emphasis is on wormhole routing networks, the proposed schemes can be applied to networks with different switching techniques. Some other network topology dependent encoding schemes are possible.

As indicated earlier, different encoding schemes have their own advantages and disadvantages. The choice of an appropriate encoding scheme is dependent on many factors, such as network topology, network size, routing strategy, processor allocation strategy, switching technique, and frequent multicast communication patterns. A complicated system may be able to simultaneously support different encoding schemes. In this case, another indication flit is needed to indicate which encoding scheme is used in the associated multicast message.

In this chapter, we only address the basic concept of various multi-address encoding and decoding schemes. When a scheme is implemented in a network, additional level of header optimization is possible, depending on the corresponding network characteristics. In the next chapter, we illustrate how to implement each encoding/decoding scheme in a MIN and how to further optimize the header overhead.

CHAPTER 3

Multistage Interconnection Networks

Multistage interconnection networks (MINs) are a popular class of interconnection architecture for constructing *scalable parallel computers* (SPCs), such as the BBN TC-2000, IBM SP-1, TMC CM-5, Meiko CS-2, and NEC Cenju-3 as well as high speed networks, such as SynOptics ATMx and DEC GIGAswitch. In such systems, processor nodes are interconnected through a MIN, a class of indirect networks. Each processor node has its own processor(s), local memory, and other supporting devices. As the number of nodes in the system increases, the total communication bandwidth, memory bandwidth, and processing capability of the system scale up as well. MINs can be further classified as unidirectional MINs and bidirectional MINs [21]. As indicated in [21], bidirectional MINs are essentially a fat tree, such as the TMC CM-5, IBM SP-1, and Meiko CS-2. We concentrate our work on those unidirectional MINs, such as the NEC Cenju-3 [18] and BBN TC-2000 [14].

The general topological structure of a MIN can be employed in a number of varying configurations. For the sake of consistency and clarification, Figure 3.1 shows a generic MIN structure that is of interest to us. The N -node MIN with $N = k^n$ input ports and N output ports has n stages of $k \times k$ switches. The n stages are placed horizontally apart, while within each stage there are $\frac{N}{k}$ switches vertically stacked.

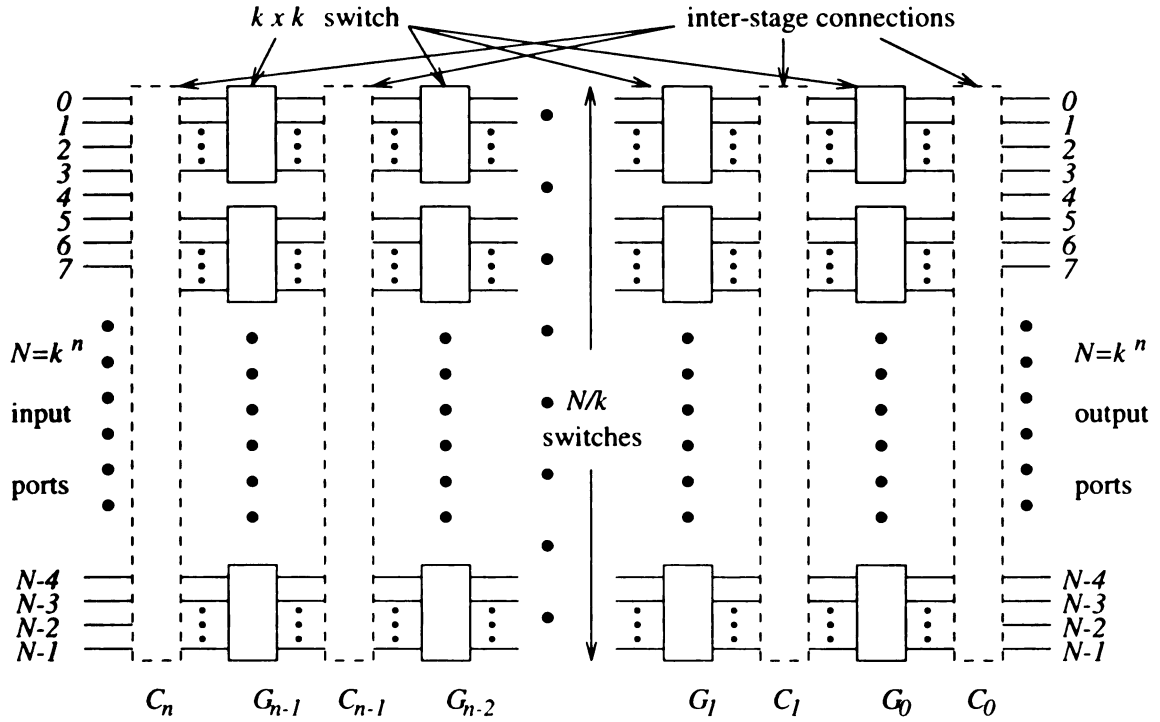


Figure 3.1: A generic MIN structure with $N = k^n$ input/output ports and n stages.

3.1 Switches

Switches are the basic building blocks of MINs. A $k \times k$ switch is a crossbar network with k inputs and k outputs. If each input port is allowed to connect to more than one output port, the switch is able to support a more complicated function called *one-to-many* or *multicast* communication. Unfortunately, such design faces a critical problem

— the deadlock. As long as two multicast trees share two common channels, deadlock is possible. Limitation or extra hardware is needed to avoid potential deadlock. The software approach provides an alternative for deadlock-free multicast. To achieve efficient unicast-based multicast on unidirectional MINs is the target of this study. Several network topologies are studied and specified in the following sections.

3.2 Network Topology

An N -port MIN built with $k \times k$ switches can be represented as

$$C_n(N)G_{n-1}(N/k)C_{n-1}(N) \dots C_1(N)G_0(N/k)C_0(N)$$

where G_i refers to the i^{th} stage, C_i refers to the i^{th} connection, and $N = k^n$. There are n stages. Each stage G_i consists of N/k identical $k \times k$ switches and thus is denoted as $G_i(N/k)$. Each connection C_i connects N right-hand side ports at stage G_i to N left-hand side ports at stage G_{i-1} and thus is denoted as $C_i(N)$. A connection pattern C_i defines the topology of the one-to-one correspondence between G_{i-1} ports and G_i ports, also known as *permutation*.

There are many ways to interconnect adjacent stages. Banyan networks are a class of MINs with the property that there is a unique path between any pair of source and destination [29]. An N -node ($N = k^n$) Delta network is a subclass of banyan networks, which is constructed from identical $k \times k$ switches in n stages, where each stage contains (N/k) switches. A unique property of the Delta network is its self-

routing property [30]. Many of the known MINs, such as Omega, flip, cube, butterfly, and baseline, belong to the class of Delta Networks [30] and have been shown to be topologically and functionally equivalent [31]. A good survey of those MINs can be found in [32].

This work considers three major interconnection patterns between adjacent stages: baseline, butterfly, and perfect shuffle, which are formally defined below.

Definition 1 *The i^{th} k -ary baseline permutation δ_i^k , for $0 \leq i \leq n-1$, is defined by*

$$\delta_i^k(x_{n-1} \cdots x_{i+1} x_i x_{i-1} \cdots x_1 x_0) = x_{n-1} \cdots x_{i+1} x_0 x_i x_{i-1} \cdots x_1 \quad \text{where } 0 \leq x_i \leq k-1.$$

Definition 2 *The i^{th} k -ary butterfly permutation β_i^k , for $0 \leq i \leq n-1$, is defined by*

$$\beta_i^k(x_{n-1} \cdots x_{i+1} x_i x_{i-1} \cdots x_1 x_0) = x_{n-1} \cdots x_{i+1} x_0 x_{i-1} \cdots x_1 x_i \quad \text{where } 0 \leq x_i \leq k-1.$$

Definition 3 *The perfect k -shuffle connection σ is defined by*

$$\sigma_i^k(x_{n-1} x_{n-2} \cdots x_1 x_0) = x_{n-2} x_{n-3} \cdots x_1 x_0 x_{n-1} \quad \text{where } 0 \leq x_i \leq k-1.$$

Four topologically equivalent MINs are considered in this thesis: baseline, butterfly, cube, and omega. Since all of them are a class of Delta network, the self-routing property allows the routing decision to be determined by the destination address. For a $k \times k$ switch, there are k output ports. If the value of the corresponding routing tag is i ($0 \leq i \leq k-1$), the corresponding packet will be forwarded via port i . For

an n -stage MIN, the routing tag is $T = t_{n-1}t_{n-2} \cdots t_1t_0$, where t_i controls the switch at stage G_i .

Baseline MINs. In a baseline MIN [31], connection pattern C_i is described by the i^{th} baseline permutation δ_i^k which is defined in Definition 1. The i^{th} baseline permutation puts the 0^{th} digit to i^{th} position and shifts every digit after and including the i^{th} digit one digit right. The perfect shuffle connection, σ , is selected to be connection pattern C_n . For a given destination $d_{n-1}d_{n-2} \cdots d_0$, the routing tag is formed by having $t_i = d_i$ for $0 \leq i \leq n-1$.

Butterfly MINs. In a butterfly MIN, connection pattern C_{n-i} is described by the i^{th} butterfly permutation β_i^k , for $0 \leq i \leq n-1$. As indicated in Definition 2, the i^{th} butterfly permutation interchanges the 0^{th} digit and the i^{th} digit of the index. β_0^k is selected to be connection pattern C_0 . For a given destination $d_{n-1}d_{n-2} \cdots d_1d_0$, the routing tag is formed by having $t_i = d_{n-i}$ for $1 \leq i \leq n-1$ and $t_0 = d_0$.

Cube MINs. In a cube MIN (or multistage cube network [32]), connection pattern C_i is described by the i^{th} butterfly permutation β_i^k for $0 \leq i \leq n-1$. C_n is selected to be σ . For a given destination $d_{n-1}d_{n-2} \cdots d_0$, the routing tag is formed by having $t_i = d_i$ for $0 \leq i \leq n-1$.

Omega Network. An omega network [33] is defined by $C_i = \sigma$, for $1 \leq i \leq n$, and C_0 is identity connection. For a given destination $d_{n-1}d_{n-2} \cdots d_0$, the routing tag is formed by having $t_i = d_i$ for $0 \leq i \leq n-1$.

Figure 3.2 shows the connection patterns of these MINs, which have been extensively studied in the past and have been adopted in many research prototype parallel computers, such as the Illinois Cedar [34], the Purdue PASM [35], the IBM RP3 [36],

and the NYU Ultracomputer [37]. Some commercial parallel computers have also adopted such networks, such as the BBN GP-1000 ($k = 4$), TC-2000 ($k = 8$) [14], Monarch ($k = 8$) [38], and the NEC Cenju-3 ($k = 4$) [18]. Both the GP-1000 and TC-2000 use circuit switching¹. The NEC Cenju-3 adopts wormhole switching.

3.3 Node Architecture

All nodes in the system are interconnected via a unidirectional MIN. In this paper, it is assumed that there is exactly one pair of input channel and output channel connecting a node to the network, resulting in so-called “one-port communication architecture”. This assumption, which is consistent with many existing multistage network systems, implies that the local processor must transmit (receive) messages in sequence. For ease of explanation, all output channels of the nodes are connected to the left-hand side of the network, and all input channels of the nodes are connected to the right-hand side of the network (i.e., there is a wraparound connection on the network).

3.4 Decoding in Multistage Interconnection Networks

This section considers multi-address decoding on multistage cube networks of which the interconnection pattern is the butterfly permutation. An example of a 16×16

¹With circuit switching, reply messages, such as acknowledgment, can be sent back via the same path.

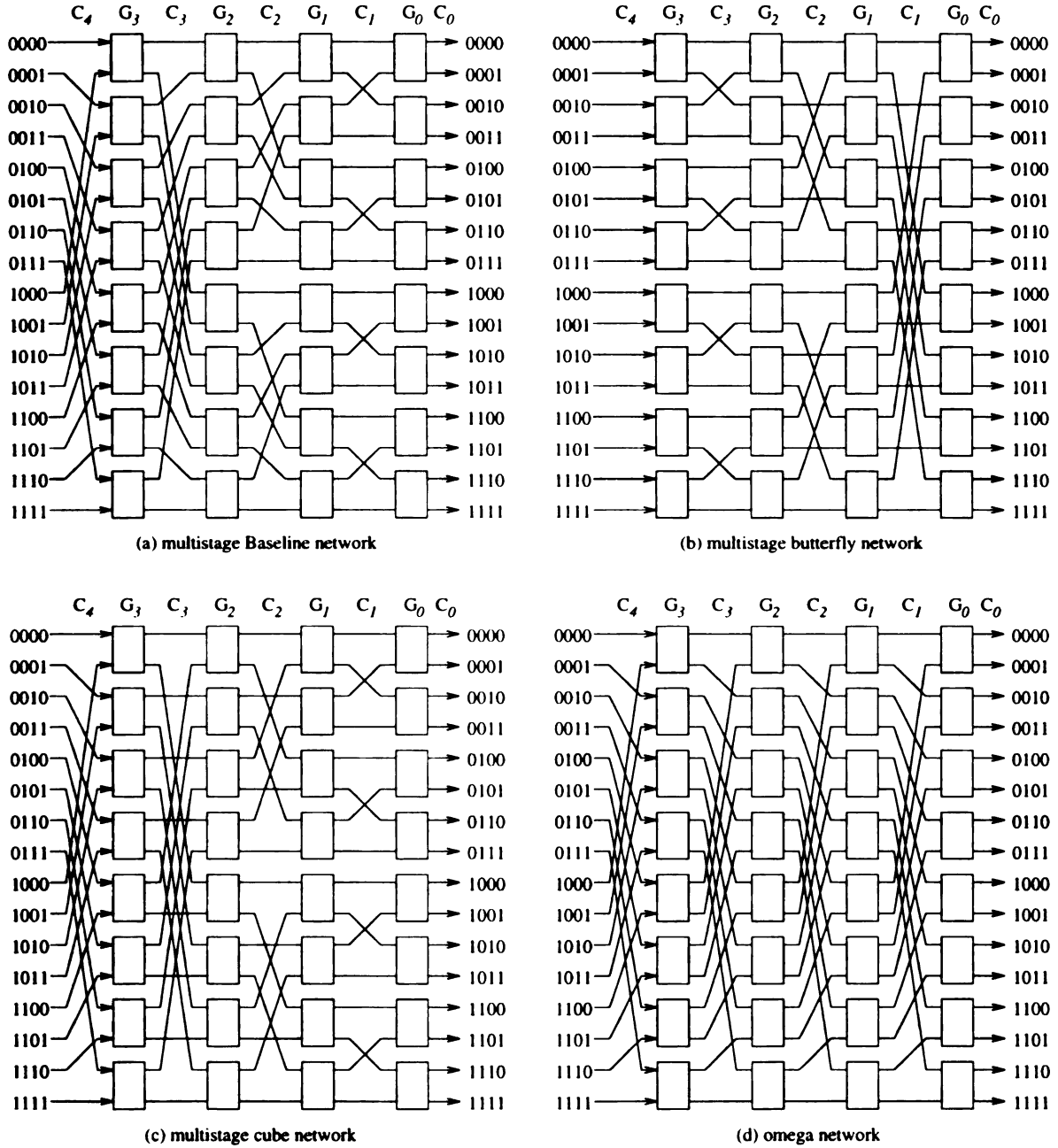


Figure 3.2: Four 16-node MINs built with 2×2 switches.

h

o

r

f

i

st

w

t

b

as

re

co

de

st

ea

of

I

($k = 2$) cube network is shown in Fig. 3.3, where G_i represents the i^{th} stage and C_i stands the i^{th} interconnection pattern. In order to avoid drawing wrap-around connections, nodes are shown on both sides of the network. However, the dotted circle on the right side represents a shadow node of the associated node on the left side. The cube MIN has been demonstrated to perform reasonably well in practice and is the basis of the k -ary n -cube network architecture. Examples of such commercial parallel computers include the BBN GP-1000 ($k = 4$) [13] and the BBN TC-2000 ($k = 8$) [14], among many other research prototypes. However, if a link becomes congested or fails, the unique path property can easily disrupt the communication between some input and output pairs. The reachable nodes from a switch are dependent on the stage and the number of output ports. In general, a switch, S_{ij} , can reach k^{i+1} nodes, where k is the switch size. For example, switches S_{10} and S_{20} in Figure 3.3 can reach 4 and 8 nodes, respectively. Because of space limitation for figures, k is assumed to be 2 in the following study unless otherwise specified.

When there are alternative paths between the source and destination nodes, such as extra stage MINs, the switch must handle the header carefully to avoid *duplicated receiving*. Figure 3.4 gives an example of such a situation where source node 0 encodes its destination by multiple region stride (2:10:2; 1:13:3), which indicates the destinations are nodes 1, 2, 4, 6, 7, 8, 10, and 13. The shadow switch routes the first stride to upper port and the second stride to lower port. The following switches route each stride to its associated destinations. Hence, nodes 4 and 10 receive two copies of the message. Note that this situation only happens when alternative paths exist.

In Delta networks, the switch may forward the same address to all enabled output

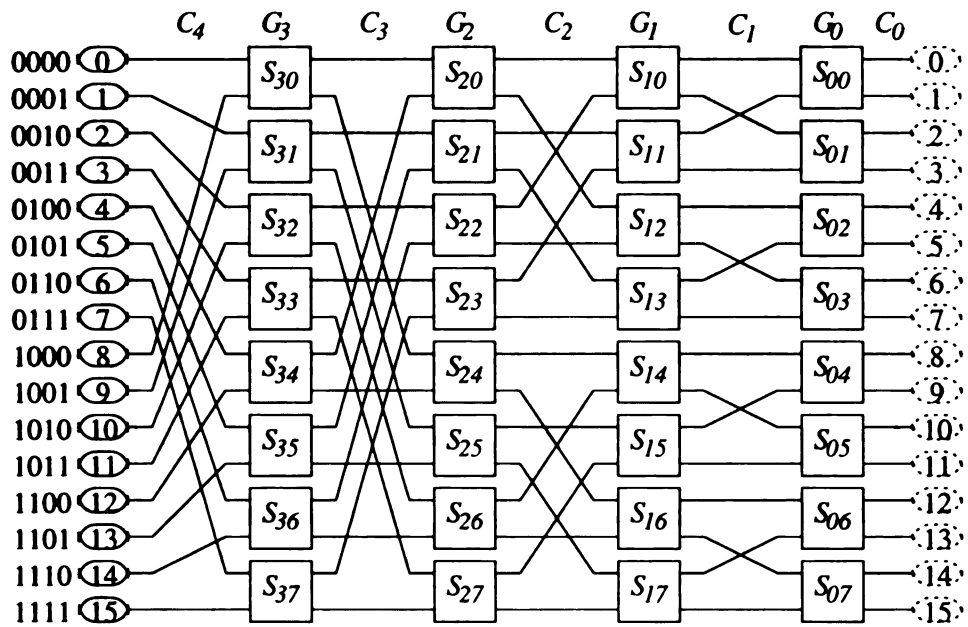


Figure 3.3: A 16×16 cube network built with 2×2 switches.

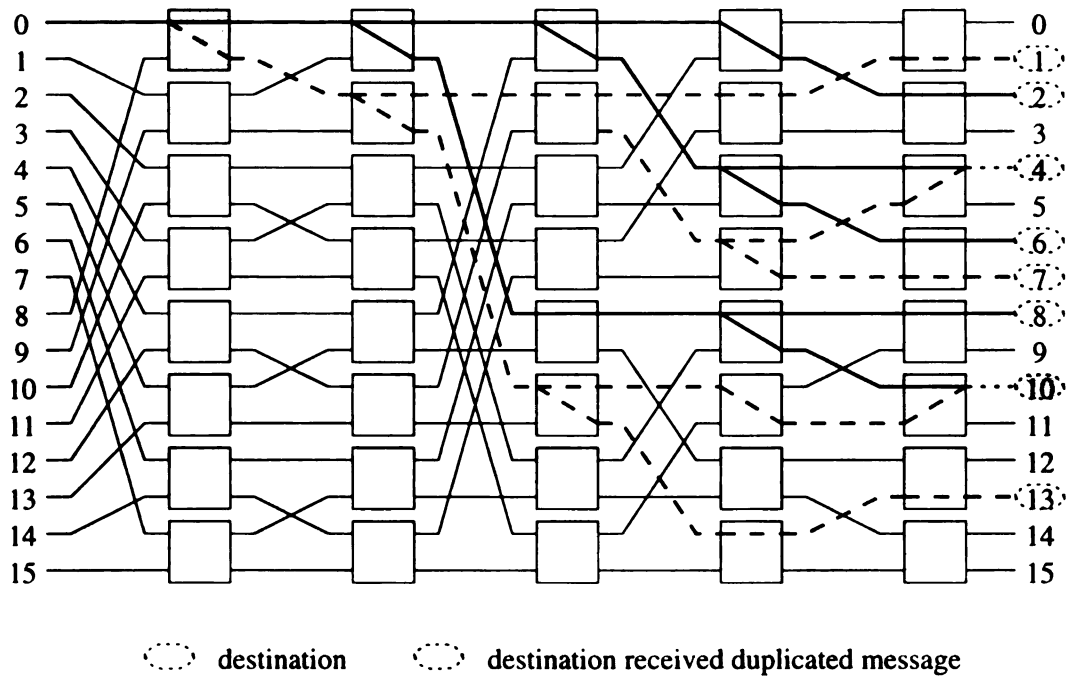


Figure 3.4: A duplicated receiving when there are alternative paths, in an extra stage MIN, between source and destination nodes.

ports since this address is only reachable by one of these output ports. An unreachable address is used to indicate a wormhole bubble and is represented as a **shadow square** in the following figures. The destination subsets may be overlapped because the routing path is unique. Such an approach provides more flexibility in forming the header. The reachable nodes of a switch are in ascending order according to its output ports. In other words, a single region will not be split to multiple regions for any single output port. Hence, a counter, *counter*, is used to specify the number of regions in a multicast message rather than *EOH* to reduce the length of header for such a network.

For each of the six multi-address encoding schemes, this section will describe some address decoding algorithms that may be performed at a switch in a MIN. Given a header flit, say F , the routing function $Route(F)$ will determine the output port number to be selected to forward the message. If F is not reachable by the current switch, this function returns -1. Furthermore, we assume that if the processor, if any, associated with a switch is one of the destinations, a copy of the message will be sent to the processor. An output port is enabled if a copy of the message is to be forwarded through the port. At the end of message transmission, all output ports will be disabled. These behaviors will not be further described in the following algorithms.

3.4.1 All Destination Decoding

Figure 3.5 gives the decoding algorithm for each incoming flit, d , which is an address. The associated output channel must be available when sending information

Algorithm all-destination decoding

Input: An address d

Output: Append d to the header of all enabled output port.

Procedure:

```

begin
  if  $d = EOH$  then send EOH to all enabled ports; exit;
  /* EOH is used since counter will not decrease the length of header
    in this algorithm.*/
   $j := Route(d)$ ;
  if ( $j \geq 0$ ) then enable port  $j$ ;
  send  $d$  to all enabled port; /*  $d$  is a useful address to the message
    directed to port  $j$  and is a useless address to other enabled port. */
end

```

Figure 3.5: All address decoding algorithm.

through an enabled port; otherwise, the message transmission will be pending until the port is available. Avoiding deadlock is another critical design issue. The solution is dependent on the routing strategy, and is beyond the scope of this paper.

For example, let the source node be 0 and destinations be $\{2, 3, 5\}$ in an 8×8 MIN. The modification of header in each switch is shown in Figure 3.6, where E indicates EOH. Switch S_{20} receives the header $(2,3,5,E)$. It enables upper output port when it detects address 2, and it enables the lower output port when it detects address 5. The new headers for upper port and lower port are $(2,3,5,E)$ and $(5,E)$, respectively. Address 5 in the first header is a wormhole bubble and is marked as a shadow square in the figure to indicate that it is not reachable by the first message.

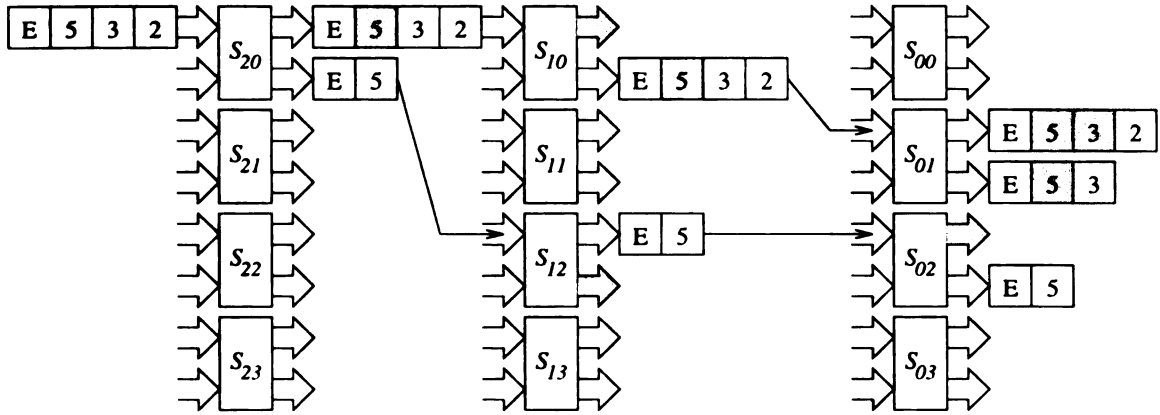


Figure 3.6: An example of all-destination decoding.

3.4.2 Bit String Decoding

Buffered Bit String Decoding

In this scheme, a switch stores the entire bit string and then detects each bit which has a 1 to enable the corresponding output port. The length of each bit string is dependent on the location of the switch. A switch at stage G_i can reach only k^{i+1} destinations. Therefore, the bit string length is k^{i+1} , where the first k^i bits correspond to the nodes reachable from the first output port, the second k^i bits indicate the second output port, and so on. Let $T = t_0 t_1 \dots t_{(k^{i+1}-1)}$, where t_j is a single bit to indicate the j^{th} reachable node of the switch, be the input bit string. The buffered bit string decoding algorithm is given in Figure 3.7. Figure 3.8 gives an example where the source node is 0 and destinations are $\{2, 3, 5\}$ in an 8×8 MIN.

Hierarchical Bit String Decoding

The first field (k bits) of an incoming bit string indicates the enabling or disabling of the k output ports of a k -port switch in the hierarchical bit string encoding/decoding

Algorithm buffered bit string decoding for a switch in stage G_i

Input: Binary bit string, T .

Output: Bit string, D_j , for each enabled output port j .

Procedure:

```

begin
  for  $\ell = 0$  to  $k - 1$  do  $(E_\ell, D_\ell) = \text{Route}(\text{next } k^i \text{ bits});$ 
    /* Here  $\text{Route}$  returns 1 to  $E_\ell$  if the OR operation of these  $k^i$  bits is 1;
       otherwise  $E_\ell = 0$ .  $D$  is a string contains these  $k^i$  bits only. */
  for all  $E_\ell = 1$ , where  $0 \leq \ell \leq k - 1$ , do
    enable port  $\ell$ ;
    send  $D_\ell$  to port  $\ell$ ;
  endfor;
end

```

Figure 3.7: The algorithm of bit string decoding schemes for MIN.

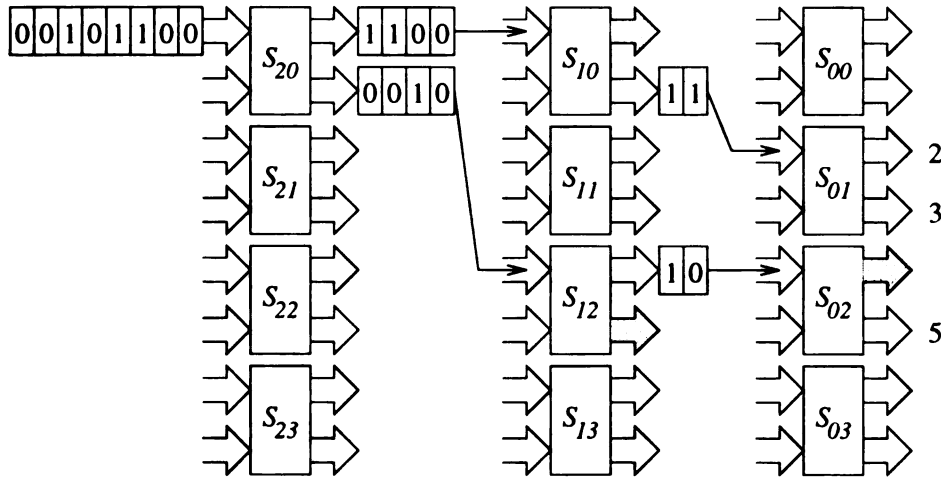


Figure 3.8: An example of buffered bit string decoding.

scheme. The following bits are divided into k fields, one for each output port. A field is forwarded to the next switch if the associated port is enabled. The decoding algorithm is too simple to be described here, and an example where the source node is 0 and destinations are $\{2, 3, 5\}$ is shown in Figure 3.9.

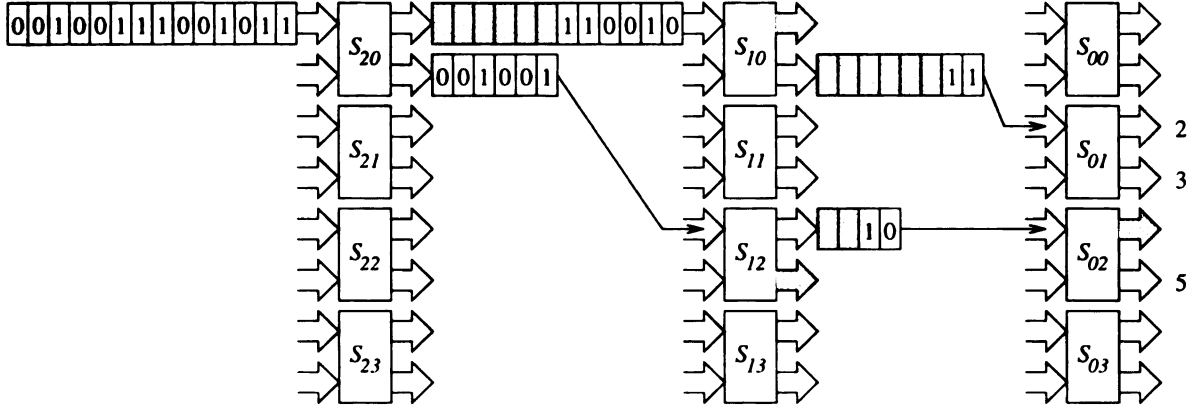


Figure 3.9: An example of hierarchical bit string decoding.

3.4.3 Multiple Region Broadcast Decoding

The header in multiple region broadcast contains several regions as given in Figure 2.1(c). Any nodes covered by all regions should receive a copy of the message. Thus, the switch may have to divide a region into several sub-regions, and each sub-region is directed through an appropriate output port. A region counter, *counter*, to indicate the number of regions in the header is used in this scheme to reduce the length of header. A switch saves the counter when it is enabled by the incoming header. It decreases the counter by one when it processes one region, puts the counter in an outgoing header when the corresponding port is newly enabled, and disables the decoding procedure when the counter becomes 0. The algorithm to decode a region in multiple region broadcast header is given in Figure 3.10, where the process of *counter*

is omitted.

Algorithm multiple region broadcast

Input: A region with the beginning address, b , and the ending address, e

Output: Enable associated output ports and revised header

Procedure:

```

begin
  while ( $b \leq e$ ) do
    ( $j, e'$ ) :=  $Route(b, e)$ 
    send region ( $b, e'$ ) to port  $j$ ;
     $b = e' + 1$ ;
  end while;
end

```

Figure 3.10: Multiple region broadcast decoding algorithm.

As given in the algorithm, the switch checks the beginning address of a region and searches for the first address, e' , which cannot be reached by the same output port. A sub-region is then identified for that output port. The process repeats until all sub-regions have been identified. For example, consider that source node 0 sends a message to nodes 1, 3, 4, 5, and 6 as shown in Figure 3.11. Switch S_{20} directs the first region to port 0 and splits the second regions (3:6) to (3:3) and (4:6) for ports 0 and 1, respectively. The region counter to port 1 is updated to 1. This process is repeated in each switch, and the message is directed to all destinations as shown in this figure.

3.4.4 Multiple Region Stride Decoding

The decoding scheme to handle multiple region stride is similar to that of multiple region broadcast. A switch divides the region into several sub-regions, where each

s

a

n

a

c

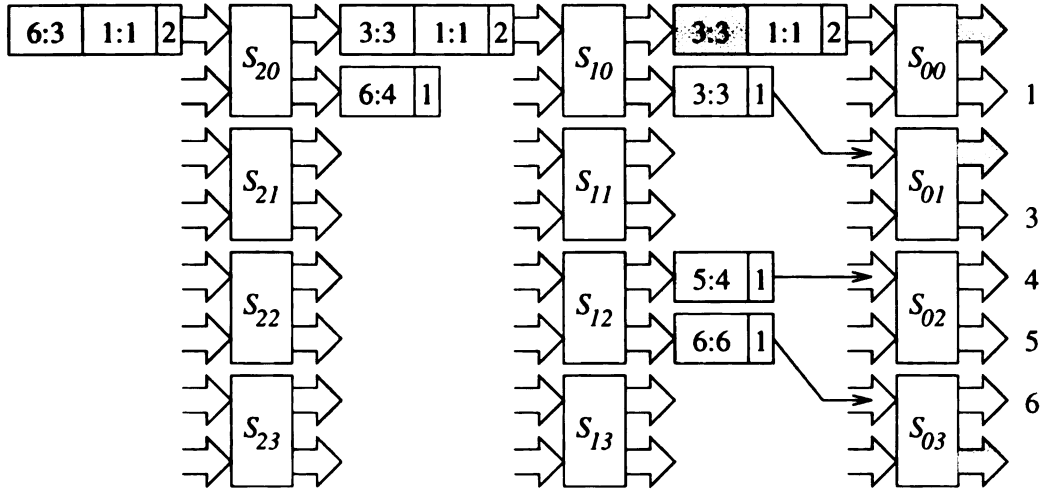


Figure 3.11: An example of multiple region broadcast decoding.

sub-region is directed to a single output port. As indicated in Figure 2.1(d), there is an additional field, *stride*, in each region to indicate the distance between two adjacent nodes. The decoding algorithm is similar to Fig. 3.10, except that it replaces (b, e) and (b, e') by (b, e, s) and (b, e', s) , respectively. Note that the stride value is never changed.

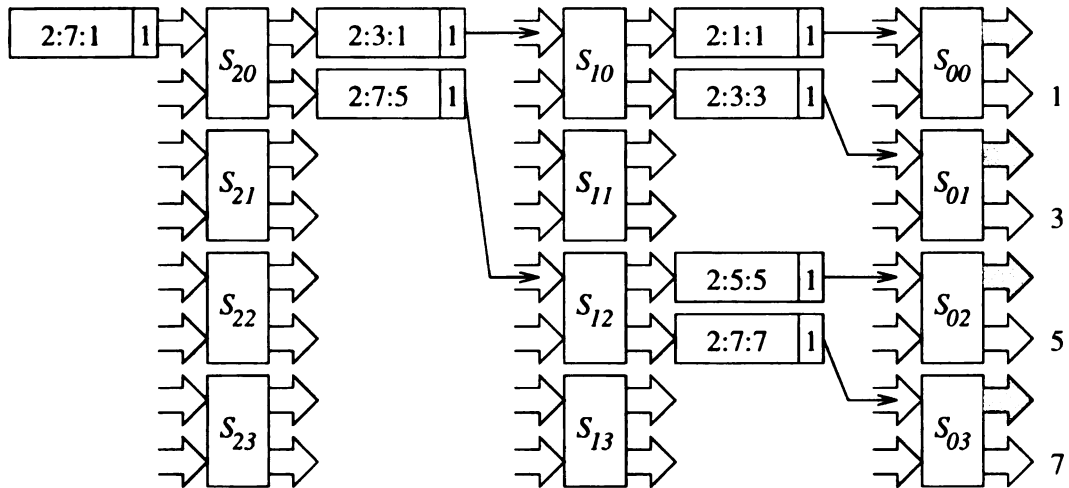


Figure 3.12: An example of multiple region stride decoding.

Consider the example in Figure 3.12 where source node 0 sends a message to all odd nodes. Switch S_{20} splits the region $(1:7:2)$ to $(1:3:2)$ and $(5:7:2)$ for ports 0 and

1, respectively. This process is repeated until the message reaches all destinations as shown in the figure.

3.4.5 Multiple Region Mask Decoding

Since the cube MIN is a cube network, the multiple region mask may be applied to it, where each stage represents a dimension. A region mask (b, m) can define a complete subcube and a region mask (b, e, m) can specify a subset of a subcube in such a network. Figure 3.13 shows an example of defining a subcube in a MIN. The subcube contains nodes $\{4, 5, 12, 13\}$ (or $\{0100, 0101, 1100, 1101\}$), which can be specified by either $(x10x:101)$ or $(0100:1101:1001)$. If node 1101 is not a destination, the first scheme requires two regions, but the second one can specify the destinations by $(0100:1100:1001)$.

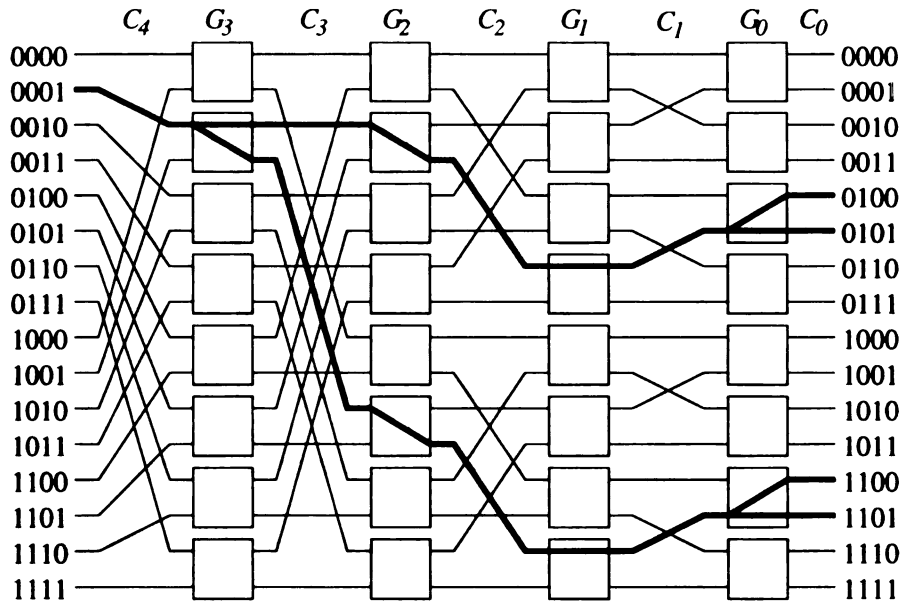


Figure 3.13: Define a subcube in a MIN.

The method to handle region mask is similar to that of multiple region broadcast

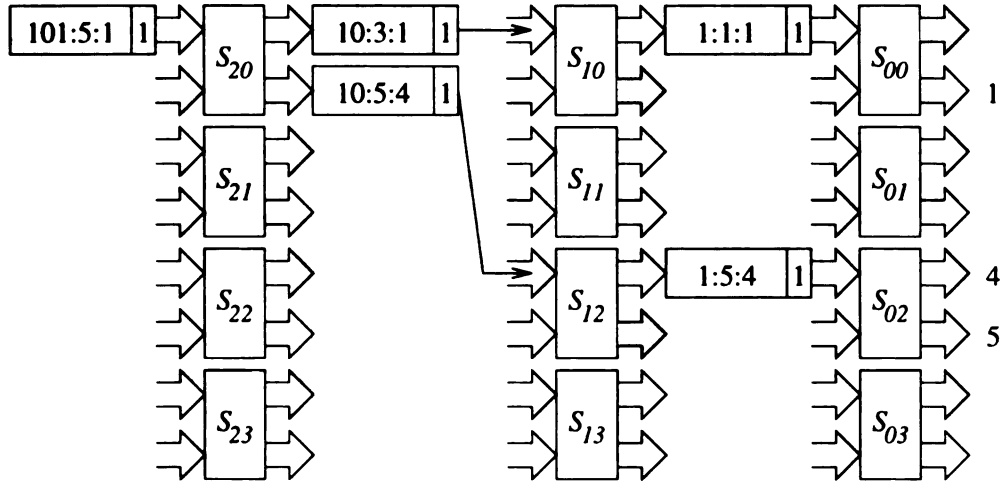


Figure 3.14: An example of multiple region mask decoding.

and multiple region stride. A region is divided by the switch into several sub-regions, where each sub-region is directed to an associated output port. Not only the beginning and ending address but also the mask will be changed by the switch to avoid duplicated receiving. An example to handle region mask (001:101:101) is given in Figure 3.14.

3.4.6 Multiple Region Bit String Decoding

There are three fields in a region as shown in Figure 2.1(f). The decoding algorithm is also similar to that of multiple region broadcast except for the representation of regions. Unlike the multiple region bit string specified in the previous chapter, it is not necessary to mark off an unreachable bit to avoid duplicated receiving since each destination is reachable by a unique path. The corresponding algorithm to decode one region bit string, which is repeated to decode the whole header, is shown in Figure 3.15.

Figure 3.16 gives an example of multiple bit string. Source node 0 sends message

Algorithm multiple region bit stringInput: A region (b, e, T)

Output: Send each sub-region to an output port

Procedure:

begin

 if ($counter \leq 0$) then exit; while ($b \leq e$) and ($T \neq 0$) do $(j, e', T') := Route(b, e, T)$; send region (b, e', T') to port j ; $b :=$ corresponding node of the first non-zero t_i after node e' ;

end while;

 decrease $counter$ by one;

end

Figure 3.15: Multiple region bit string decoding algorithm for MIN.

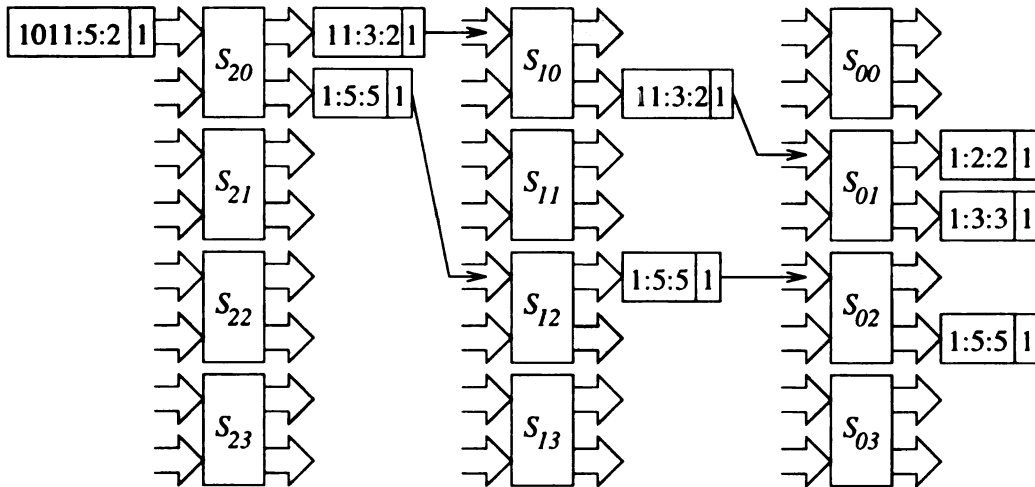


Figure 3.16: An example of multiple region bit string decoding.

to nodes 2, 3 and 5. The header incoming to switch S_{20} is $(2, 5, 1101_2)$. S_{20} splits this region into two regions $(2, 3, 11_2)$ and $(5, 5, 1_2)$ for port 0 and port 1, respectively. Both regions are forwarded by switches S_{10} and S_{12} . The first region is further split by S_{01} to direct the message to nodes 2 and 3. The second region is forwarded to node 5 by switch S_{02} .

3.5 Performance of Multi-Address Encoding and Decoding on Multistage Interconnection Networks

In this section, we evaluate the performance of the proposed schemes on different address patterns. Such performance is dependent on both the number of destinations and the distribution of destinations. Here, a 256×256 cube MIN is considered. Since an address requires 8 bits, a flit is assumed to contain 8 bits. A region may require 2 or 3 flits. The destination sets of these patterns are specified as follows:

Pattern 1: $\{1:12:1, 32:50:2, 60:75:1, 90:117:3\}$

Pattern 2: $\{1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 64, 72, 79, 85, 90, 94, 97, 99\}$.

Pattern 3: node 1 to node 48.

Pattern 1 contains 48 nodes. The all-destination scheme requires 48 flits to carry all addresses. The buffered bit string and hierarchical bit string schemes require 256 and 510 bits or 32 and 64 flits, respectively. Twenty-eight nodes of these destinations

are located in two contiguous regions and others are located separately; therefore, the multiple region broadcast requires 22 regions. The multiple region stride outperforms others since it can encode these destinations by 4 regions. The multiple region mask encodes the first 38 destinations into 3 regions (00000001:00001100:00001111), (00100000:00110010:00111110), and (00111100:01001011:01111111). The Hamming distance of the last 10 destinations is no longer 1. Therefore, the multiple region mask is unable to group them into a single region mask and encodes them into 10 regions. Hence, it has 13 regions. Since only the distance between nodes 12 and 32 is larger than 16 (the length of two addresses), the multiple region bit string encodes these destinations into 2 regions, where the first region contains nodes 1 to 12 and the second region consists of all other destinations.

There are 18 destinations in the second pattern. Therefore, the all destination scheme needs 18 flits to carry the addresses. Since there is no contiguous region or regular stride, the multiple region broadcast takes 18 regions, where each region has one destination; and the multiple region stride requires 9 regions, where each region contains two destinations. These destinations form 4 2-cubes, (1,3), (64,72), (90,94) and (97,99); therefore, the multiple region mask has 14 regions. The region bit string encodes all destinations into a single region since the distance between any two neighboring nodes is less than the length of two addresses. It takes 15 flits, one for the beginning address, one for the ending address and the other thirteen for the bit string.

Pattern 3 is easily encoded and is omitted here. The length of headers of these schemes for these three patterns is summarized in Table 3.1.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

Table 3.1: Number of flits in a header based on various multi-address encoding schemes. A header consists of a counter and addresses.

Encoding schemes	Pattern 1	Pattern 2	Pattern 3
all-destination	49	19	49
buffered bit string	32	32	32
hierarchical bit string	64	64	64
multiple region broadcast	45	37	3
multiple region stride	13	28	4
multiple region mask	40	43	4
multiple region bit string	19	14	8

A naive way to implement multicast is to send multiple unicast messages, one for each destination. This approach has been implemented in a lot of communication software and is named as *separate addressing* [39]. This approach has a fixed length of header, one for each destination. No counter flit is needed. However, the separate addressing requires the source node to send $m \times (\ell + 1)$ flits, where ℓ is the length of a multicast message and m is the number of destinations. The number of flits sent by the source node for patterns 1, 2, and 3 is based on different sized messages. The different encoding schemes are given in Figure 3.17(a), (c) and (e), respectively. To compare the proposed schemes with separate addressing, the reduction rate, R_{red} , is defined as one minus the ratio of the number of flits sent in a proposed scheme to the number of flits sent using the separate addressing.

$$R_{red} = 1 - \frac{\text{number of flits sent in a multi-address encoding scheme}}{\text{number of flits sent in the separate addressing}}$$

Hence, a larger reduction rate, R_{red} , indicates a better scheme. The reduction rate for patterns 1, 2 and 3 is shown in Figure 3.17(b), (d) and (f), respectively.

360

239
No.
of
figs.
120

(

360

239
No.
of
figs.
120

0

(

360

239
No.
of
figs.
120

(

0

(

Fig
vs ϵ

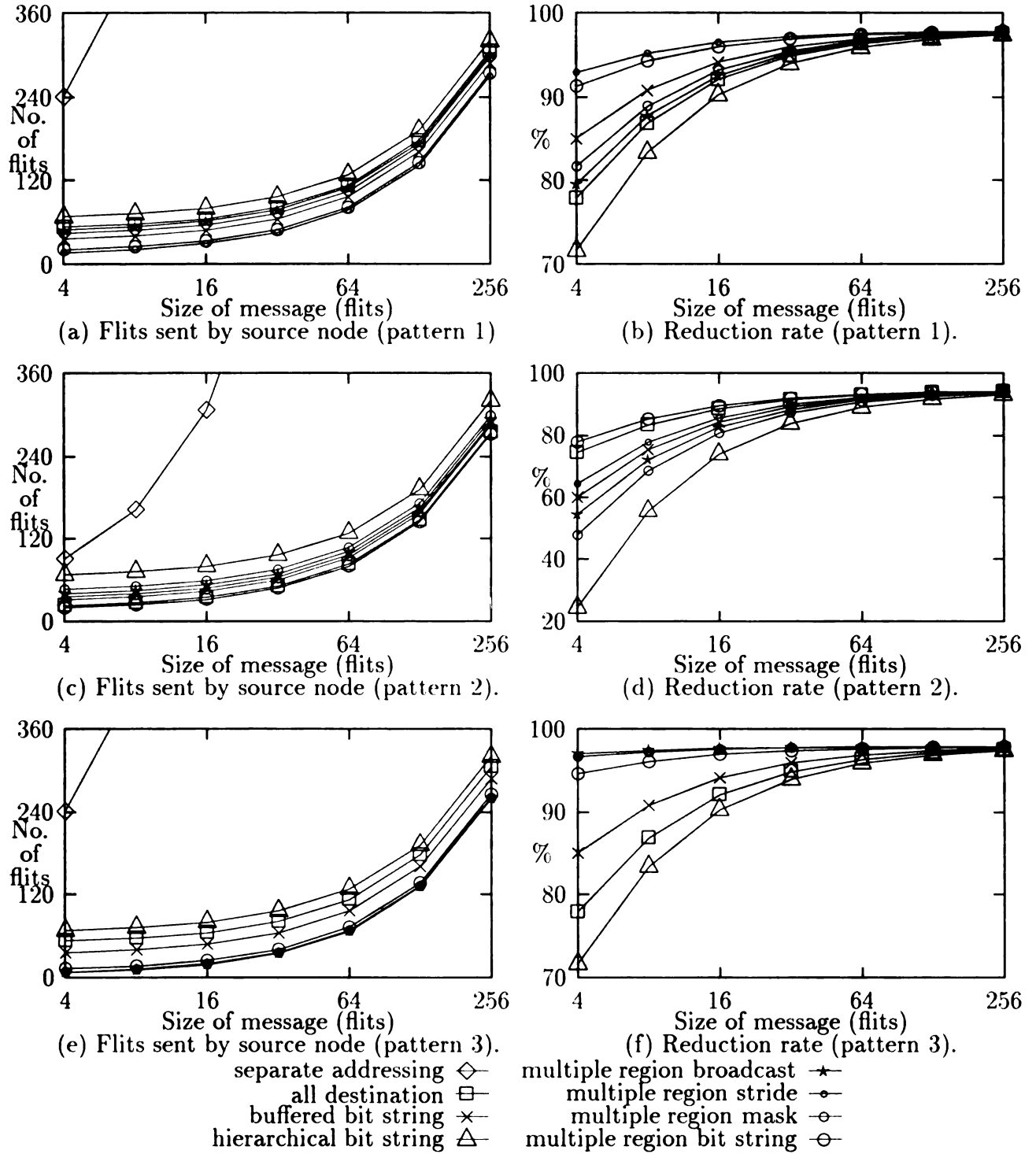


Figure 3.17: The length of header and reduction rate on different destination patterns vs different multi-address encoding/decoding schemes.

The performance of the separate addressing scheme is dependent on both the number of destinations and the length of messages. In order to complete a multicast, it needs to transmit the message m times where m is the number of destinations. However, unless the message is short and the number of destinations is small, its performance is worse than any proposed multi-address encoding/decoding scheme. The performance gap between the separate addressing and the proposed schemes increases as the size message increases and/or the number of destinations increases. The performances on various multi-address encoding schemes become closer to each other when the message size increases, which implies that the data portion of the message comes to dominate.

It is easy to observe that there is no single multi-address encoding/decoding scheme that can outperform all others in all destination patterns. For example, the multiple region stride has much shorter header than others if all destinations have a constant stride. And in this latter case, the worst multi-address scheme is the hierarchical bit string. However, the hierarchical bit string will become better than all-destination when, for example, there are more than 64 destinations; and multiple region broadcast when there are more than 32 regions.

It is also easy to observe that the length of header in most multi-address encoding/decoding schemes is dependent not only on the number of destinations but also on the distribution of destinations. The all-destination encoding scheme is dependent on the number of destinations only. However, its performance becomes worse as the number of destinations increases. On the other hand, the buffered bit string and hierarchical bit string are dependent on the number of nodes in a network but not

the number of destinations.

The multiple region stride is a superset of multiple region broadcast. Hence, the number of regions in the multiple region stride is always less than or equal to the number of regions in the multiple region broadcast. When the number of regions in the multiple region stride is less than $\frac{2}{3}$ of that in the multiple region broadcast, the multiple region stride has a shorter header. The multiple region mask is also a superset of multiple region broadcast since a single contiguous region in a cube MIN can always form a subset of a subcube. However, it is not a superset of the subset of the multiple region stride encoding scheme.

3.6 Summary

In this chapter, we give a brief review of delta class MINs followed by optimized multi-address encoding and decoding schemes on a multistage cube network. Since the baseline, butterfly, cube and omega MINs are topology equivalent [31], the optimized multi-address encoding and decoding schemes may apply to baseline, butterfly, and omega MINs directly.

The hardware multicast tree injects fewer flits into the network which implies decreasing the load of the network as well as communication latency. The reduction rate of the network traffic becomes larger as the size of the message increases. A multi-address encoding and decoding scheme may outperform the other schemes for some traffic patterns. However, the performance gap decreases as the size of the message increases. Hence, a system may choose a single multi-address encoding and

decoding scheme if supporting all multi-address schemes is expensive.

Since a message destined to multiple destinations will reduce the network traffic and communication latency, the implementation of a hardware multicast tree becomes the target of our study and is discussed in the next chapter.

(

H

V

Th

the

nu

tre

we

mo

base

are

depr

forw

unles

CHAPTER 4

Hardware Multicast

Wormhole-Switched

The hardware multicast implementation offers a significant better performance than the software approach; however, it also suffers potential deadlock due to multiple multicasts. Two hardware multicasts have been introduced: the path-based and the tree-based. However, we will show that tree-based hardware multicast, *multi-head worm*, is more suitable for a MIN than the path-based hardware multicast, which is more suitable for direct network in avoiding deadlock.

In order to provide efficient deadlock-free hardware multicast, two different tree-based hardware multi-head worm implementations, asynchronous and synchronous, are studied. The asynchronous multi-head worm allows each branch to forward independently, while the synchronous multi-head worm insists on having all branches forward synchronously. Unfortunately, both implementations are not deadlock-free unless certain rules are applied. Hence, current hardware implementations of mul-

ticast either exhibit some undesirable properties or are restricted in their use. To implement deadlock-free multiple asynchronous multi-head worms is very difficult, if not impossible. A deadlock-free and starvation-free hardware implementation supporting multiple synchronous multi-head worms is presented in this chapter.

Unlike a unicast message whose header has a fixed size, the size of a multicast message header is dependent on the number of destinations, the distribution of destinations, and the encoding scheme [40, 41]. Since a single flit may not be able to carry all destination information, a multi-head worm may have different distances of branches toward different destinations. The difference in latency among multiple branches may cause deadlock in synchronous multi-head worms. We address this issue and propose a pseudo multi-address encoding scheme to eliminate the potential deadlock.

4.1 Implementation of Multi-head Worms

The path-based hardware multicast, which was shown to easily avoid deadlock in direct networks [25], can easily cause deadlock in banyan networks due to self blocking. Figure 4.1 gives an example of such a situation where the source is node 0 and the destinations are nodes 4, 8 and 9 on a 16-node multistage cube network. We further assume that there is one buffer for each outgoing link as shown by a blank square. A router replicates an incoming flit and then forwards one copy to the processor and the other copy to the network. Three paths are required to complete the path-based multicast: from node 0 to node 4, from node 4 to node 8, and from node 8 to node

9. However, the latter two paths will share channels between switches s_{24} and s_{14} as well as between switches s_{14} and s_{04} , marked as a shadowed area. As indicated in the figure, the first two paths, shown as bold lines in the figure, are established completely. When the first flit heads from switch s_{30} to switch s_{24} , it is blocked since the upper buffer in switch s_{30} is used to store flit 5 unless the message is very short. Thus, the path forms a cycle due to self blocking and results in a deadlock. Due to the unique-path routing property in banyan networks, it is impossible to avoid deadlock unless extra stages are added or there are multiple virtual channels per physical channel.

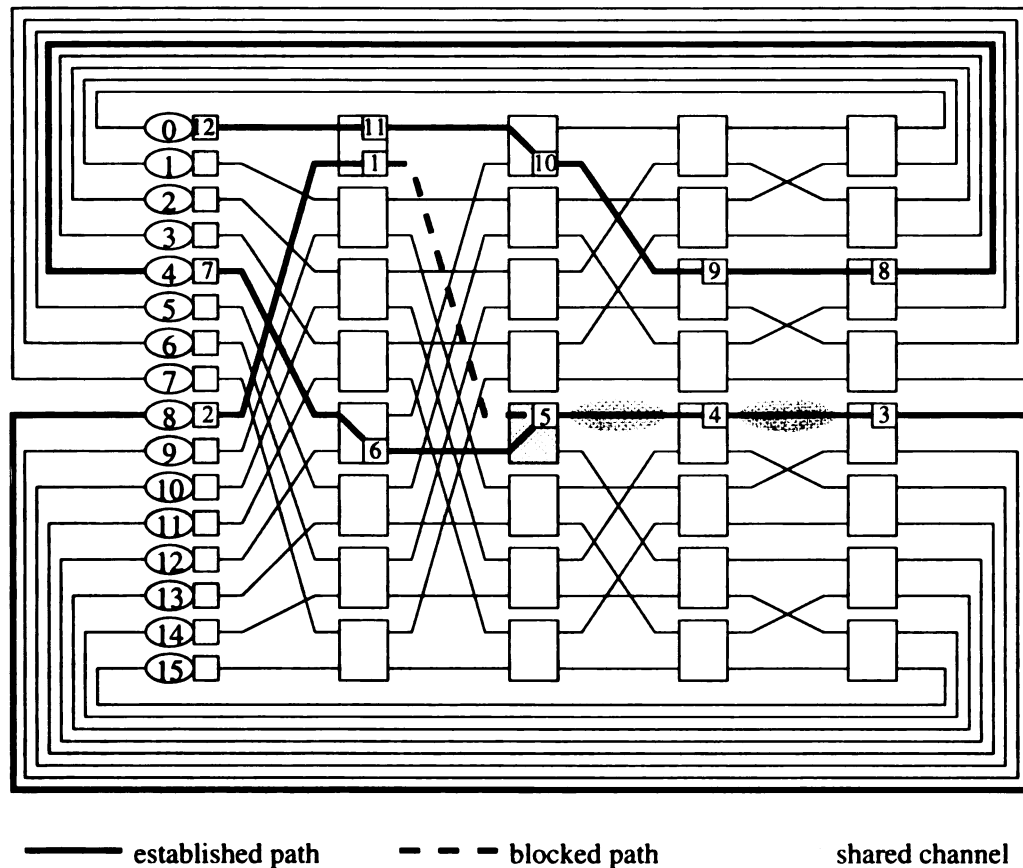


Figure 4.1: An example of deadlock for a path-based multicast on a 16-node multi-stage cube network.

Without considering the potential deadlock, the hardware multicast provides several advantages over the software implementation, such as sharing the common resources, higher throughput, lower latency, less network traffic, *etc.* Hence, some systems restrict one multicast at a time (e.g., the control network in the TMC CM-5), which results in low network utilization and longer delay. An implementation to eliminate such restriction and to guarantee a deadlock-free network is the objective of this chapter. We concentrate our work on tree-based hardware multicast.

Establishing a multicast tree on the banyan MINs is straightforward due to the unique path property. The multicast tree is also unique. In addition to unicast, a switch must be capable of replicating the incoming flit(s) and forwarding them to the associated output port(s). Since every outgoing port can be either used (enabled) or unused (disabled), there are $(2^k - 1)$ possible output port combinations on a $k \times k$ switch for a multi-head worm. When a header is received, the switch enables the associated outgoing port(s) and forwards the header to all enabled port(s). All following flits are forwarded to all enabled outgoing ports directly. The buffers and channels are released after the tail flit passes through the switch. An example to establish a multicast tree which is initiated by node 2 and heading for nodes $\{3, 4, 5, 6, 7\}$ on an 8-node cube network with 2×2 switches is given, and the snapshots at the first and third cycles are shown in Figure 4.2. When switch s_{22}^1 receives the header from its upper incoming port, it replicates and forwards the header to both buffers since destination 3 requires the upper channel and other destinations need the lower

¹In the following context, s_{ij} and $s_{i,j}$ indicate the switch at stage G_i and row j . $s_{i,j}$ is used when there is potential confusion on i and j .

channel (see Figure 4.2(a)). Such a process is repeated on switches s_{10} and s_{12} in the next cycle and then on switches s_{01} , s_{02} and s_{03} at the third cycle (see Fig 4.2(b)).

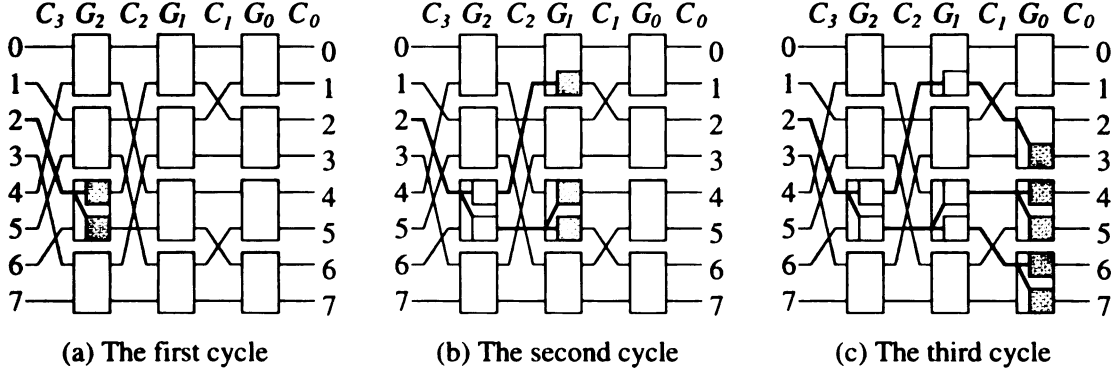


Figure 4.2: An example to establish a wormhole-switched multicast tree, where the shadow box represents the header flit and white box stands for data flits.

Deadlock is possible if the network allows simultaneous transmission of multiple multicasts. Depending on the coordination mechanism of multiple branches in a tree (or multiple heads in a multi-head worm), two different multi-head worm implementations are described below, which provide different degrees of difficulty in handling deadlock-free multiple multicasts.

The headers of different branches in an *asynchronous multi-head worm* are forwarded independently. When any branch is blocked, other branches may still be forwarded if the required outgoing channel is available. However, every flit on the blocked path will remain in the current switch. The buffers of the forwarding branches may become empty if the subsequent flits are blocked.

The same example as the previous one is given in Figure 4.3 except that there exists a unicast from node 7 to node 5. The first cycle is similar to Figure 4.2(a), and the second cycle of multicast tree establishment is shown in Figure 4.3(a). The branch toward nodes 4 and 5 is blocked at switch s_{12} since the buffer to node 5 is used

by the unicast. All following flits on the branch are stopped. The first data flit toward node 3 is forwarding since it is not on the blocked path, but the one toward nodes 6 and 7 is stopped since it shares the same channel used by the blocked branch. In the first cycle after the blocking, the header flit is forwarded to switches s_{01} and s_{03} . The data flit located at the upper buffer in switch s_{22} forwards to switch s_{10} while the one at the lower buffer remains in the current switch. All subsequent flits are kept at the source node since one buffer in s_{22} is unavailable as shown in Figure 4.3(b). The next cycle is shown in Figure 4.3(c). The paths to nodes 3, 6 and 7 are established but not the paths to nodes 4 and 5. Nodes 3, 6, and 7 receive the header flit; in the subsequent cycle, node 3 will receive the first data flit but nodes 6 and 7 will not. After that, none of these nodes will receive any flit until the unicast is done and the paths to nodes 4 and 5 are established.

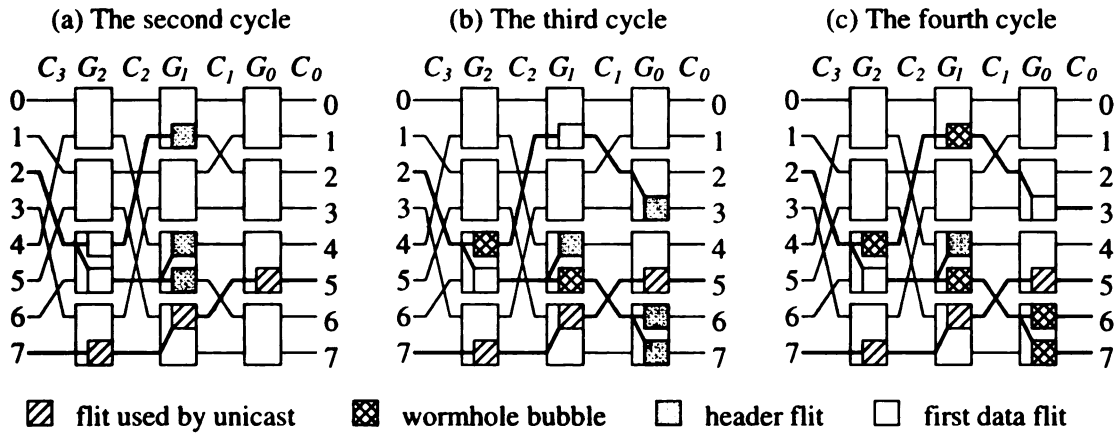


Figure 4.3: Establishment of an asynchronous multi-head worm with network contention.

The empty buffers in a multicast tree are referred to as *wormhole bubbles* since they contain no data and are unusable to other transmission. Wormhole bubbles may cause deadlock. Figure 4.4 gives an example of such a situation. Node 0 initiates

a multi-head worm, T_0 , which is destined for nodes $\{2, 4, 6, 8, 10, 12, 14\}$ when a unicast from node 4 to node 1 is on the network. Hence, the branches to nodes $\{2, 4, 6, 8\}$ are blocked at switch s_{30} , and the other branches are completely connected to the associated destinations. While T_0 is waiting for the completion of the unicast, node 6 starts a new multi-head worm, T_6 , which is destined for nodes $\{5, 7, 13, 15\}$. The branches heading to nodes 5 and 7 are established, while the branches to nodes 13 and 15 are stopped at switch s_{26} by T_0 . This situation is shown in the same figure. When the unicast is done, T_0 will grab additional channels to reach node 2 but will wait for T_6 to release switch s_{12} . Hence, a deadlock is formed.

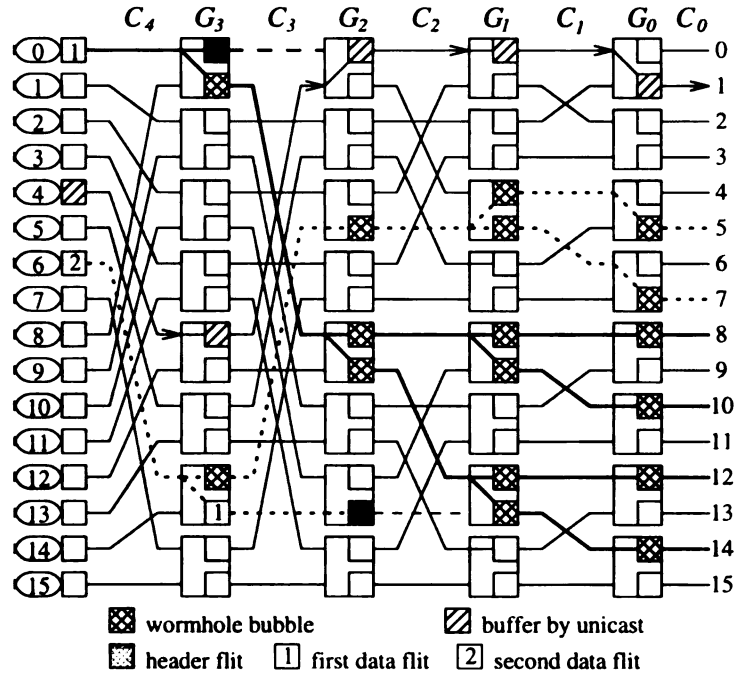


Figure 4.4: An example of a deadlock situation with two multi-head worms, where the bold line represents the multi-head worm T_0 and the dotted line stands for T_6 .

It is easy to observe that establishing a deadlock-free asynchronous multi-head worm is very difficult, if not impossible, without any restriction and extra hardware. This is due to the fact that each branch forwards independently, and a switch can

not forward the next flit until all the previous flits are forwarded.

4.2 The Synchronous Multi-head Worm

Unlike the asynchronous multi-head worm, the synchronous worm has a strong coordination of different branches. All headers must be forwarded synchronously. Whenever one branch is blocked, every branch is blocked. Since there is no path on which headers of different branches can exchange status, the information is sent backward to the source node. The source node will then push the whole multicast tree one flit forward if all requested buffers are free. Such coordination must be efficiently implemented in hardware to minimize the latency. A dedicated wire is established to the source node from a switch as soon as the switch is obtained by the multi-head worm. Wires from different branches are connected together like a reduction tree. The connection from different branches can be based on wired-OR (or wired-AND) logic. Thus, the signal can be immediately sent to the source, and the signal from the source can be immediately sent to all branches. A similar design was used in the nCUBE-2 to support hardware broadcast within a subcube [23] and in the Cray T3D in their synchronization network design [42].

Figure 4.5 gives the same example as shown in Figure 4.3 but for a synchronous multi-head worm. Since all request buffers are available, the header flit is forwarded to switch s_{22} and then switches s_{10} and s_{13} , as illustrated in Figure 4.5(a) and (b), respectively. However, the lower buffer in switch s_{12} is used by another transmission. Switch s_{12} gets a negative signal which is passed backward to the source. The source

informs every branch to stay in the current state, as shown in Figure 4.5(b).

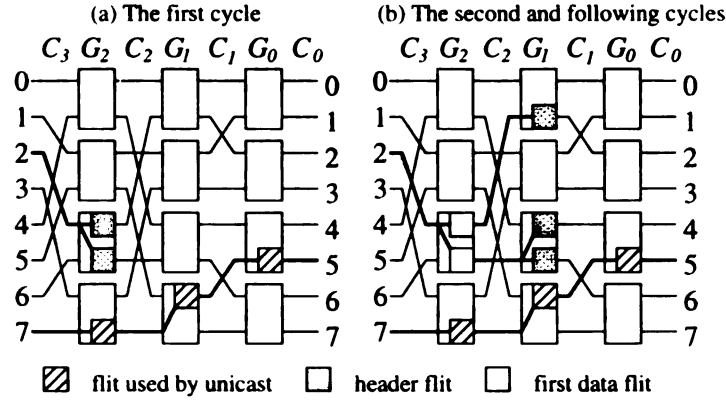


Figure 4.5: Establishment of a synchronous multi-head worm with network contention.

Due to a strong coordination of headers, it may mislead us into concluding that the multiple synchronous multi-head worms are deadlock-free. Let us consider an instance as shown in Figure 4.6 where the source nodes 2 and 8 initiate multi-head worms, T_2 and T_8 , to all odd nodes and all even nodes, respectively.

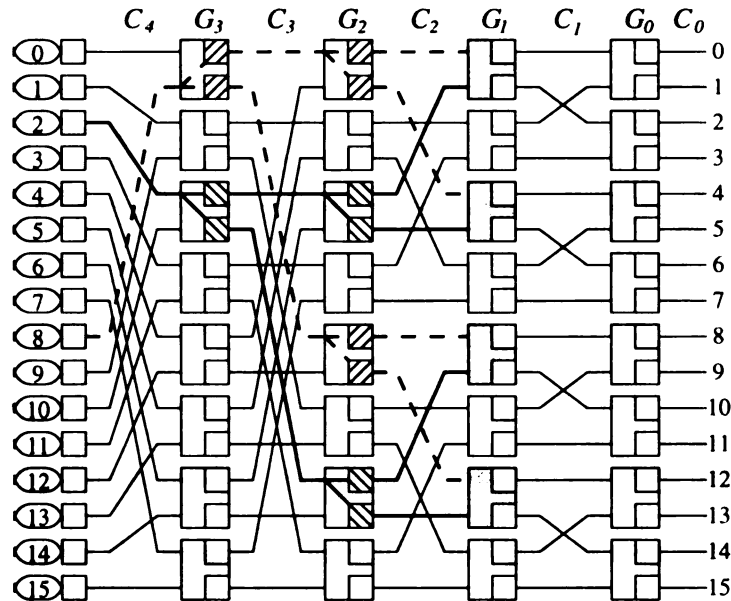


Figure 4.6: Two multi-head worms requesting common channels to achieve destination sets may cause deadlock or starvation.

As shown in the figure, both multi-head worms compete for the buffers of shadow switches in stage G_1 . When the negotiation between stages G_2 and G_1 starts, all buffers are free and such information is sent to both sources. Hence, both sources inform each branch to forward. If, unfortunately, each multi-head worm grabs part of those free buffers, a deadlock is formed. If a switch grants free status to only one of the multi-head worms, starvation becomes critical since each multi-head worm may gain permission from some switches but not others at each cycle. Several *priority* schemes are studied to insure that permissions at different switches are granted to the same message when multiple multicasts are competing for the same buffers. In order to have the priority work properly, priority is given to each message either based on the message or determined by the switch. Here, a *proper priority* scheme shall assign the same priority to all branches of a multi-head worm at the same stage but may be varied in different stages. All multi-head worms must have a distinct priority at the same stage. Such a scheme guarantees deadlock-freeness and is formally specified in Lemma 1.

Lemma 1 *The proper priority scheme provides a deadlock-free hardware multicast on MINs.*

Proof:

First of all, we will show that no two multi-head worms will block each other in the same stage under the proper priority scheme. Assume that there are two multi-head worms, T_0 and T_1 , which block each other at stage G_i , $0 \leq i \leq n$. Since the higher priority message will be forwarded, such a block means different priority in

these branches which contradicts to the proper priority.

Next, we need to prove that there is no cyclic wait. Let there be m multi-head worms, T_1, T_2, \dots, T_m , which form a cyclic wait on an N -node MIN, $N \geq m$. Furthermore, assume that T_i blocks T_{i+1} at stage g_i for $1 \leq i \leq m-1$ and T_m blocks T_1 at stage g_m . In the first half of the proof, we have shown that a blocked multi-head worm must be at least one stage behind the blocking one. Therefore, we have $g_1 < g_2 < \dots < g_m$ as well as $g_m < g_1$, which is a contradiction. \square

When multiple messages reach the same switch at the same time and request the same output channel, some arbitration mechanism must be used. *First come first served* (FCFS) is the most intuitive approach. Unfortunately, the FCFS scheme is not *proper priority* since two multicasts may have the same priority if they arrive at a stage in the same time. The *upper channel first*, which prefers the message coming from the upper channel, is a distributed scheme. Each switch makes the decision independently. Such a scheme is highly network dependent. If the positions of the incoming channels at all switches used by a multi-head worm are identical in the same stage, the priority given to a message is identical for all branches. For the baseline, butterfly, cube, and omega networks, the algorithm is a proper priority scheme which implies a deadlock-free result. A formal specification is given in Lemma 2 for the cube network. The formal specification of other networks and the associated proofs are omitted due to the similarity to Lemma 2.

Lemma 2 *The upper channel first priority scheme guarantee a deadlock-free on mul-*

tistage cube networks.

Proof: To prove this lemma, we need to show that priority given to all branches is identical at the same stage or that the same position of incoming ports in any switches at the same stage are used by a multicast tree.

Consider a source and destination pair: $s_{n-1} \cdots s_0$ and $d_{n-1} \cdots d_0$. The channel before entering stage G_{n-1} (G_{n-1}) is $s_{n-2} \cdots s_0 s_{n-1}$ and stage i ($G_i, 0 \leq i \leq n-2$) is $d_{n-1} \cdots d_{i+1} s_{i-1} \cdots s_1 s_0 s_i$ based on the construction of baseline network and destination routing. The switch $d_{n-1} \cdots d_{i+1} s_{i-1} \cdots s_1 s_0$ and the s_i^{th} incoming channel are used for the message. Since the position of the incoming channel is dependent on the source node only, all branches of a message are given the same priority, s_i , at stage G_i . From Lemma 1, such a scheme is deadlock-free. \square

The *predefined priority* based on the location of the source node is another proper priority scheme. The priority is given to a message based on the location of the source node, such as ascending order, descending order, or any predefined order. Since the priority is constant for a message, it does guarantee deadlock-freeness.

Although the above proper priority schemes offer a solution to the deadlock problem, they face another critical problem — potential starvation. The lower priority multicast trees may be blocked by the high priority multicast trees forever. The *clock rotating* scheme is proposed to avoid starvation and deadlock. The priority is given to a message based on the location of the source node and the initiated time, the time when the message is initiated by the source node. Deadlock-freeness is guaranteed

by the priority based on the location and initiated time, while starvation-freeness is secured by the priority based on initiated time. To have a consistently initiated time among different nodes, the system requires a global clock, which offers many advantages and is adopted by many SPCs, such as the TMC CM-5, Cray T3D, and BBN TC-2000.

4.3 Multi-address Encoding

Unlike a unicast message, the size of a header may be varied depending on the number of destinations, the encoding schemes, and the distribution of destinations [40]. Six *multi-address encoding/decoding schemes* were proposed, including all-destination, bit string, multiple region broadcast, multiple region mask, multiple region stride, and multiple region bit string. All destination addresses are carried in the header in the all-destination scheme, which is the most intuitive approach and the one used by many researchers. The multiple region broadcast and mask allow a message to deliver to several contiguous regions and several sub-cubes or partial sub-cubes, respectively. The header of the multiple region stride is capable of indicating multiple regions of destinations with the same strides. The stride among distinct regions may be different. The bit string fixes the size of the header, and the multiple region bit string tries to minimize the header size. Here, we will concentrate on two extreme cases: the all-destination and region broadcast. The former one allows destinations in any locations, while the latter one requires all destinations located in a contiguous region.

Assume that a single flit can carry single destination information in the all-

destination scheme in order to conform with unicast. Hence, the branch to the next destination will be exactly one stage behind the current branch. A switch splits the header whenever it detects a destination heading for different outgoing ports. Intuitively, this scheme will lead to deadlock unless destinations are encoded in some order, such as ascending or descending order. Unfortunately, this scheme is not deadlock free even though the destinations are sorted. This property is formally specified in Lemma 4. Although the lemma concentrates on the all-destination scheme, it may be easily applied to other schemes since branches to every region are one stage behind branches to their previous region.

Lemma 3 *The all-destination scheme is deadlock-free if there are at most two synchronous multi-head worms on the network.*

Proof: Since deadlock-freeness is obvious when there is one multi-head worm, we will show the condition when there are two multi-head worms. Assume that there is a deadlock when two multi-head worms, T_1 and T_2 , block each other at switches s_{i_1, j_1} and s_{i_2, j_2} , respectively. For a multistage network, we have that T_1 and T_2 meet in the same stage, say G_i . Hence, $i_1 = i$ and $i_2 = i$, or they block each other at stage G_i . Since a branch is at least one stage behind its previous one, the branch of a multi-head worm won't reach stage G_i if its previous one is blocked at stage G_i . In other words, if $j_1 < j_2$ then the branch of T_2 to reach s_{i, j_2} does not exist. Likewise, the branch of T_1 to reach s_{i, j_1} does not exist, which is a contradiction. \square

Lemma 4 *The all-destination scheme is not deadlock-free when there are more than*

two synchronous multi-head worms on the network.

Instead of a trivial proof by different distances among multiple branches, we give an example of such a situation in Figure 4.7. There are three multi-head worms, T_6 , T_{12} and T_{30} , on the network. Here, T_6 , T_{12} and T_{30} are initiated by nodes 6, 12, and 30 and destined for $\{2, 5, 21\}$, $\{4, 6, 8, 24, 25\}$, and $\{9, 10, 11, 16\}$, respectively. Furthermore, assume that T_{12} starts first, at time t , follow by T_{30} and T_6 , at time $t+1$. For the first few cycles, the multi-head worms successfully forward their branches. At time $t+5$, the first branch of T_{12} reaches stage G_0 and the second branch reaches G_1 , *etc.* At the same time, the second branch heading to node 5 from T_6 reaches stage G_2 and the third branch reaches stage G_2 . The second branch will be blocked in the next cycle by T_{12} , which forces T_6 to be blocked. The first branch of T_{30} reaches stage G_0 at the same time, which blocks the third branch of T_{12} . However, its last branch heading to node 16 is blocked by T_6 in stage G_3 at switch $s_{3,14}$. Hence, a cyclic wait is formed.

To eliminate such potential deadlock, a constant distance among different branches is needed. To enforce this constant latency, pseudo addresses are added. After sorting the destination in lexicographic order, w_i pseudo addresses are added to two adjacent destinations, d_i and d_{i+1} , where $w_i = d_{i+1} - d_i$ and $1 \leq i \leq m - 1$ when there are m destinations. The value of the pseudo addresses is equal to d_i . This scheme is named as *pseudo all destination* encoding/decoding scheme. This scheme is guaranteed to be deadlock-free and is formally described in Lemma 5. This property may apply to multiple region broadcast, mask, and stride, where a dummy region may be added to

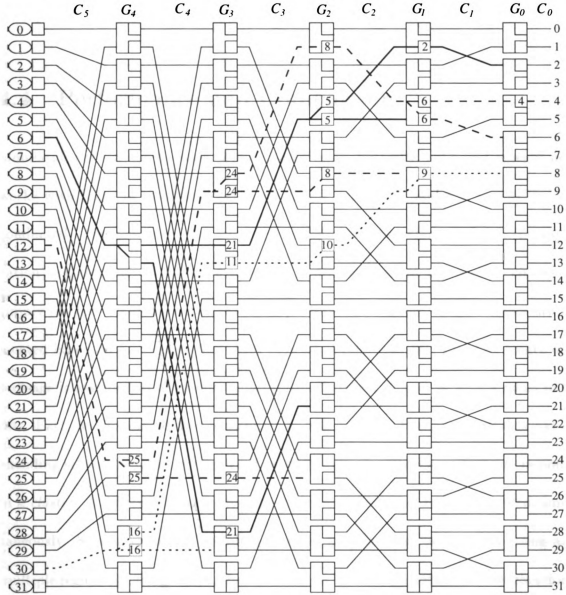


Figure 4.7: A deadlock example with all-destination encoding/decoding scheme with 3 synchronous multi-head worms, where the destination information carried by a flit is numbered within the box.

enforce a uniform distance.

Lemma 5 *The synchronous multi-head worm with pseudo all-destination scheme is deadlock-free.*

4.4 Performance and Comparison

Simulations of various 64-node MINs, the baseline, butterfly, cube and omega² networks with different sized switches, including 2×2 , 4×4 , and 8×8 switches, are conducted. The processors are allocated to different jobs, where each job (or application) usually has an exclusive subset of processors, called a *processor cluster* or simply *cluster*. To enforce the locality of processor allocation, the processors are allocated based on a complete base cube instead of randomly [21, 43]. The length of a message is assumed to be uniform distribution between 32 and 96 with a mean of 64 flits. Each destination address is assumed to take one flit. The pseudo all-destination encoding scheme requires either $m - 1$ or m flits as a header when there are m nodes in a *cluster*. A different header scheme is to force all branches to be forwarded simultaneously. Several encoding/decoding schemes may be used, such as bit string and multiple region bit string. Here, the destinations are located within a cube; therefore, the region broadcast is used. The header of the region broadcast takes a single flit. All performance reports are based on a 95% confidence interval. The system load is defined as the average number of flits entering the network from a node times the

²The performance of omega network is the same as cube network; therefore, only performance on cube network is reported.

number of destinations; that is, the average number of flits expected in an output port. Since a network can consume up to one flit per output port per time unit, the simulation is run in the range of 0.1 to 0.9 of load.

An alternative approach to provide multicast is software implementations, including the *separate addressing* (SA) and the *C-min* algorithm [43]. The source sends an individual copy of a message to every destination in the separate addressing. The C-min algorithm is a software-based multicast tree. The source forwards the message to only a subset of the destinations. Each recipient of the message forwards it to some subset of the destinations that have not yet received it. Unlike the path-based multicast tree, the software-based multicast tree is basically a store-and-forward algorithm.

As specified earlier, the hardware implementation should offer better performance. Figure 4.8 gives the latency for 16 nodes per cluster on a cube network with software implementations, including separate addressing, C-min algorithms, and the hardware implementation. The result verifies the intuitive assumption. The difference between region broadcast (RB) and pseudo all-destination (AD) encoding schemes is due to the routing process and blocking rate. Although the switch to support hardware multicast is more complicated, the performance improvement is well worth it. In the light load situation, the latency of hardware multicast is about a quarter of the most efficient software multicast. In the medium load and heavy load environments, the hardware multicast provides much better performance. If the time unit to forward a flit is doubled in the switch supporting hardware multicast, the performance is still about twice faster than the software approaches. In our simulation, the software

latency introduced by each node in the software multicast tree is ignored. In practice, the software latency can be two or three orders of magnitude higher, which further justifies the need for hardware multicast.

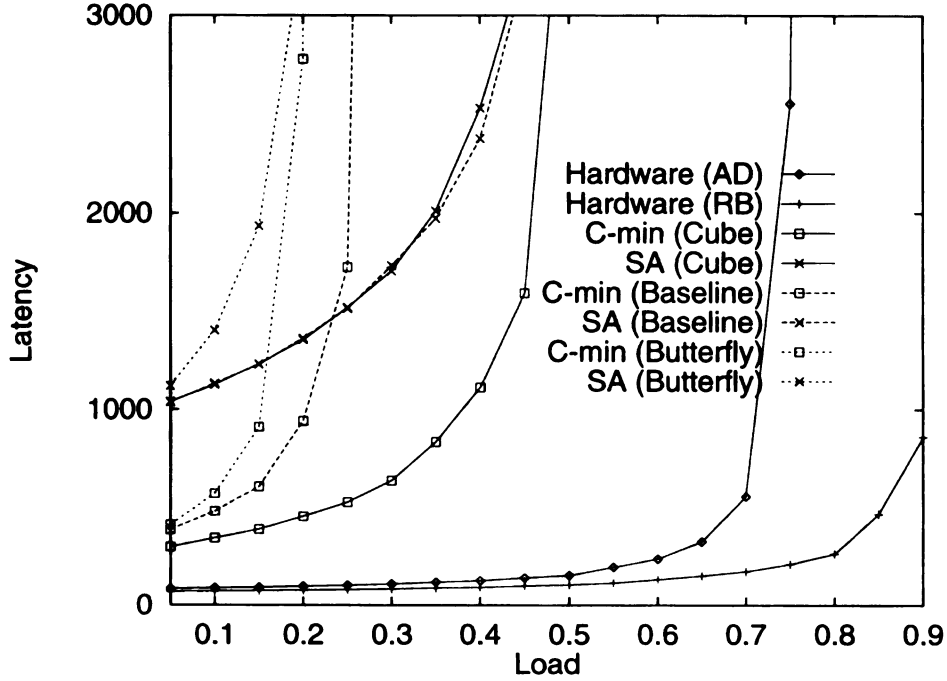


Figure 4.8: Comparison among hardware and software implementations.

A common channel to reach different destinations can be shared in a multicast tree; therefore, we expect that the performances on different networks will be similar to each other. Figure 4.9 gives the latency of different sized switches on various networks when there are 8 nodes in each cluster. It not only confirms our expectation but also shows that the performance is independent of switch sized. The latter one becomes false when the number of nodes in a clusters is smaller than the size of switches on the butterfly network. External interference among different clusters occurs. Figure 4.10 confirms this observation. Every two multicasts, 4 nodes in each cluster, share one channel in the connection C_1 on butterfly network with 8×8 switches which forces

the latency increasing rapidly. Such situation will occur to the baseline network when the size of network become larger [43].

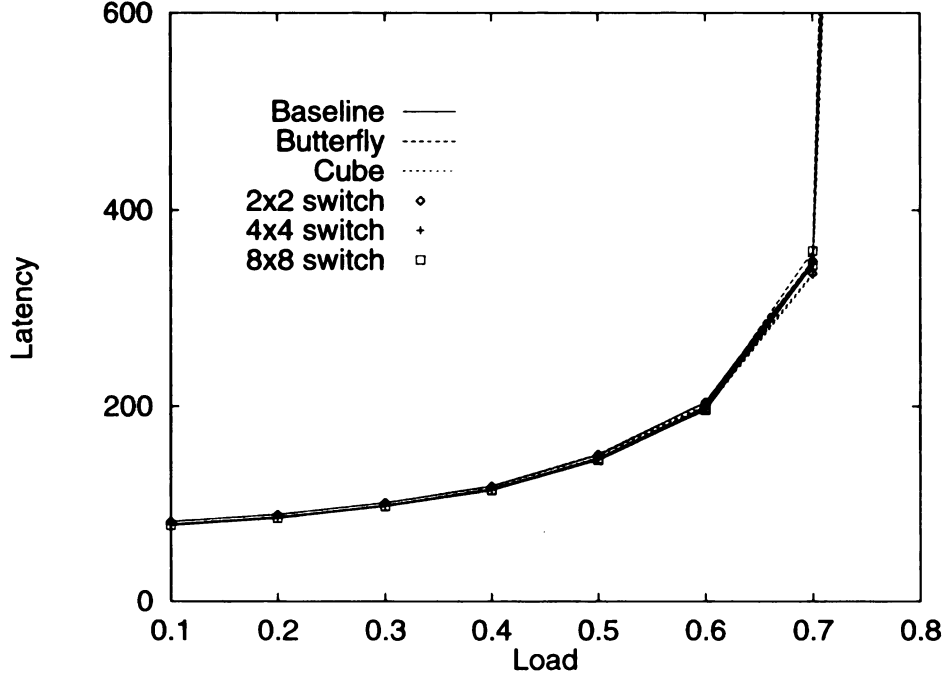


Figure 4.9: Latency comparison of different switch sizes and different number of destinations on a cube network.

In the following evaluation, 4×4 switches are used as the basic network unit unless otherwise specified. In hardware implementation, we expect that the latency should be independent of the number of destinations when the network load is light. This is confirmed in Figure 4.11 which gives the latency on various networks with a different number of nodes in each cluster when the region broadcast is used to encode the header. Surprisingly, the latency decreases as the number of nodes increases. This is due to the utilization of outgoing channels used by a multi-head worm. There are $(m - 1)$ channels used while a cluster contains m nodes. As the number of nodes decreases, the utilization decreases which implies that the latency increases.

While the pseudo all-destination is more flexible than the region broadcast, the

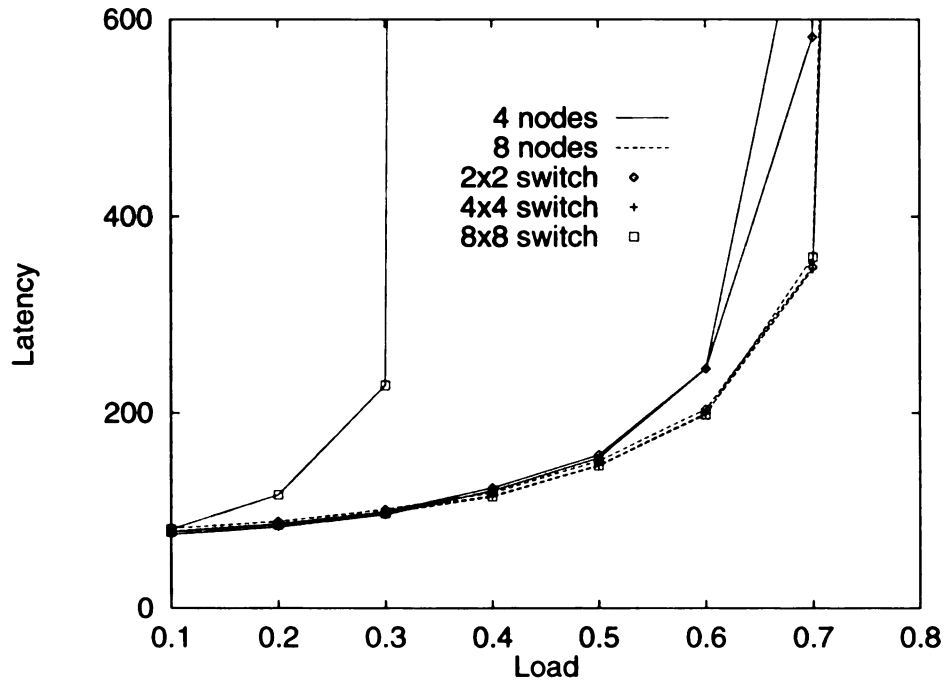


Figure 4.10: Comparison on a butterfly network with different sized switches and a different number of destinations.

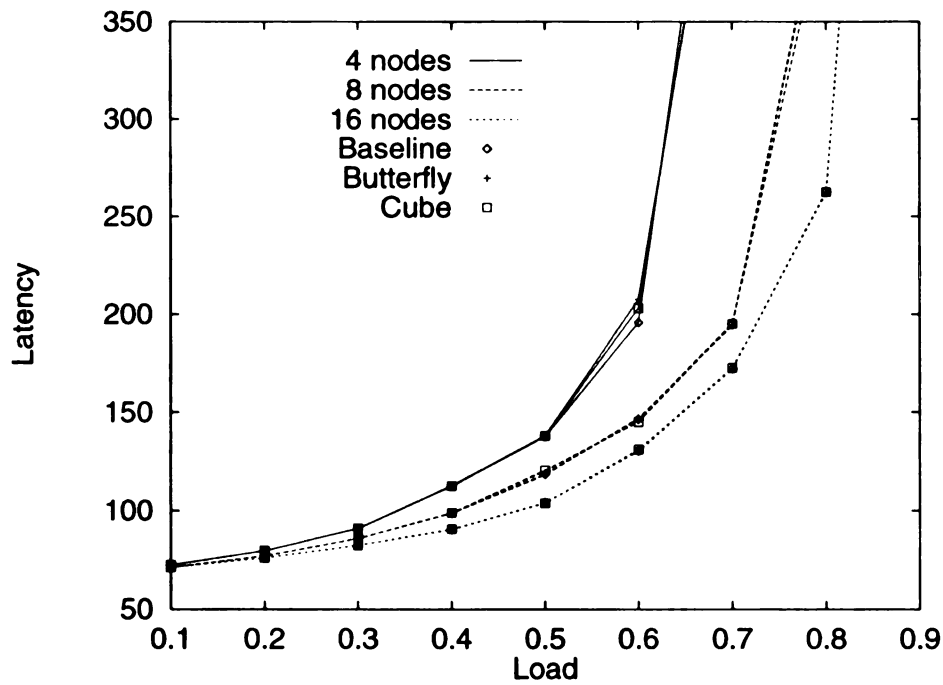


Figure 4.11: Latency comparison of the different number of nodes in each cluster.

latency increases faster than the region broadcast as shown in Figure 4.12. The region broadcast has smaller latency as the load becomes higher and the number of nodes becomes greater. There are several reasons for such phenomena. First, the routing cycle of the pseudo all-destination is linear in the number of destinations plus the number of stages, while the region broadcast is linear in the number of stages but not the number of destinations. Second, the region broadcast has a lower blocking rate since it occupies each stage in an all or none fashion. The pseudo all-destination may block more channels while waiting for the blocked channels to be released.

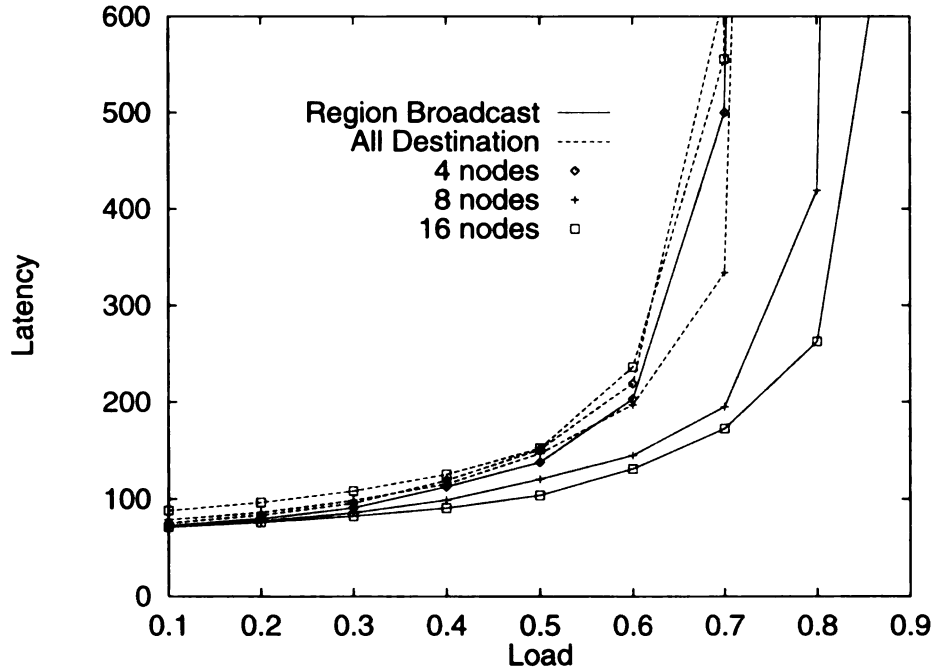


Figure 4.12: Comparison between the region broadcast and pseudo all-destination encoding schemes.

4.5 Summary

In this chapter, a deadlock-free and starvation-free hardware multicast implementation is proposed for a unidirectional wormhole-switched network. A pseudo address scheme is presented to prevent deadlock while multi-address encoding/decoding is considered. Based on the performance, the hardware multicast is highly recommended and worth the investment even if the switch to support synchronous multi-head worm would be more complicated. Although the pseudo all-destination has higher flexibility, a single region header is highly suggested. Hence, all branches can be forwarded in the same stage and the latency is shorter than other approaches. In other words, processors should be allocated based on certain relations, such as cube, stride, and contiguous region, if possible. The multistage cube and omega networks are intertask contention-free as long as the processors are allocated in a cube (as shown in Chapter 5), which implies that cube and omega networks are a better choice for constructing unidirectional MINs among others.

CHAPTER 5

Network Partitionability and Traffic Localization

Since the distance between any two nodes on a delta class MIN is fixed, different processor allocations will not affect the communication overhead in terms of path length. Instead, such overhead depends on the degree of *channel contention* and *partitionability*. Channel contention occurs whenever two or more transmissions are directed to the same output port or are competing for the same channel in a switch at any stage. If these transmissions are from different applications, it is called *intertask contention* (*external contention*); otherwise, it is called *intratask contention* (*internal contention*)

To avoid intertask contention caused by common destinations or channels, an SPC usually allocates an exclusive subset of processors, called a *processor cluster* or *cluster*, to each individual application or task. Intratask contention for a unicast communication can be minimized by flexible resource allocation that takes into account mapping

and reconfiguration [44]. For multicast or broadcast communication in a cluster, the intratask contention is unavoidable unless at most one multicast communication is allowed at a time. Hence, we concentrate on the network partitionability to eliminate the intertask contention in this chapter. Before discussing the methodology in partitioning delta class MINs, we would like to show the influence of intertask contention first.

5.1 Influence of Traffic Localization

In addition to computation of a parallel task, communication is required to achieve certain functions, such as exchanging information, barrier synchronization, *etc.* An application changes between those communication and computation phases alternatively. Let T_{comm} stand for the total communication time and T_{comp} represent the total computation time. The total execution time, T , of an application is equal to $T_{comp} + T_{comm}$.

For the sake of simplicity, let the average network contention rate be c . Since the network is not available to a communication during contention, the actual communication time T'_{comm} can be estimated by

$$T'_{comm} = \frac{T_{comm}}{(1 - c)} \quad (5.1)$$

where T_{comm} is the communication time without network contention. Hence, the total execution time of an application can be estimated by the following equation.

$$T = T_{comp} + \frac{T_{comm}}{(1 - c)} \quad (5.2)$$

Since the processing power and communication capacity are fixed after a machine is built, the T_{comp} and T_{comm} are close to constant values for an application with the same computing and communication requirements. The network contention rate is the only variable affecting the total execution time based on simple estimations in Equations 5.2. Hence, decreasing the intertask contention and/or intratask contention will reduce the network contention rate as well as the execution time. In other words, to force traffic localization will definitely improve the performance of a system.

To examine the influence of traffic localization, we eliminate the intratask contention by limiting the number of multicast communications to be at most one in each task at any time. Let each task require 8 nodes and let there be 8 tasks on a 64-node cube network. Three different processor allocation approaches are applied. The first approach is to allocate processors randomly. The influence of the intertask contention is not predictable. The second approach allocates all but one of the processor nodes into a cube. This out-of-cube node will introduce some intertask contention but not as much as the previous approach. The last approach enforces the traffic localization by allocating a complete *binary cube* to each application. Note that the binary cube will be formally defined and proved to be external contention-free for the cube network in the following sections.

Unlike previous simulations which considered the load of a network, we concentrate on the ratio of communication time to execution time which is formally defined below.

$$R = \frac{T_{comm}}{T} = \frac{T_{comm}}{T_{comm} + T_{comp}}$$

Like the simulation in Chapter 4, the length of a message is uniformly distributed between 32 and 96 flits, with an average of 64 flits. The time unit is a single cycle to forward a single flit one step toward the destination. Such a cycle is dependent on the clock of a network and can be just tens or hundreds of nanoseconds. For example, the cycle is 40 nanoseconds on a 25 MHz network. A communication phase maybe overlapped with a computation phase to reduce the communication overhead. The source node may issue the second communication command before the previous one is completed. Such multicast communication is named as *non-blocking multicast*. To avoid intratask caused by non-blocking multicast, we concentrate on single blocking multicast at a cluster. In blocking multicast, the source node will not continue to the computation phase until the communication phase is completed. The latency of a message in different processor allocation approaches is given in Figure 5.1(a).

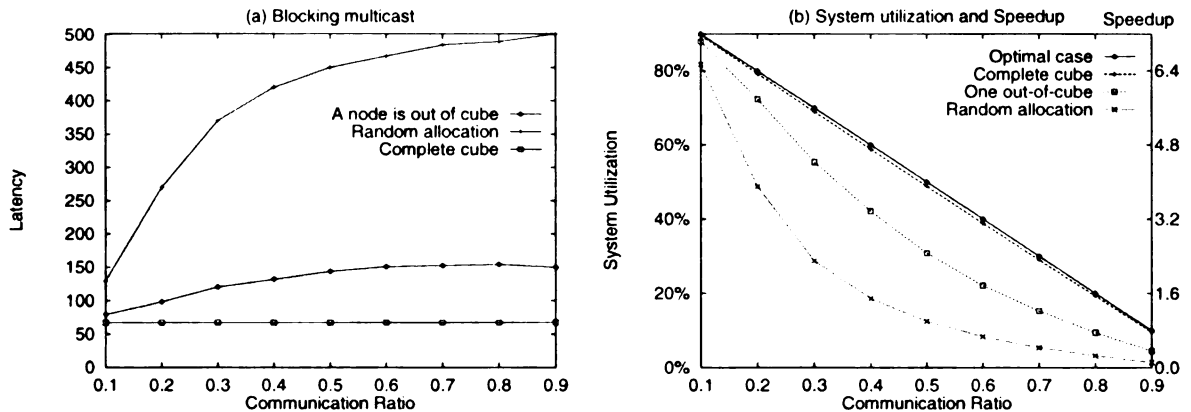


Figure 5.1: Average latency and associated system utilization of a blocking multicast message on a 64-node cube network.

Intuitively, increasing the communication ratio R would increase the contention rate c since there is more traffic on the network. Hence, the latency gets longer as the communication ratio gets higher. The latency of both the first and the second allocation approaches given in Figure 5.1(a) verifies such observation. However, the latency of cube allocation is not affected by the communication ratio as shown in the same figure. This is not contrary to our observation or estimation in Equation 5.1. As specified earlier, this approach enforces the traffic localization and eliminates the intertask contention. Since the intratask contention is avoided by single blocking multicast, this result confirms the estimation and the assumption that cube allocation is indeed an intertask contention-free allocation approach.

To further clarify the influence of traffic localization, the corresponding system utilization and speedup are shown in Figure 5.1(b). The system utilization ρ and speedup S_p are defined as following.

$$\rho = \frac{T_{comp}(1)}{T_{comm}(n) + T_{comp}(n)}$$

$$S_p = \frac{T_{comp}(1)}{T_{comm}(n) + T_{comp}(n)}$$

where $T_{comp}(n)$ and $T_{comm}(n)$ are the total computation time and the total communication time of a node when n nodes are used for a task, respectively. As shown in this figure, the intertask contention degrade the performance rapidly. For example, the system utilization degrades to 28% in random allocation and to 55% in one out-of-cube allocation approach, while the utilization is about 69% in cube allocation

when the communication ratio is 0.3. Note that the optimal utilization is 70% when R is 0.3.

Since multiple multicast communications may share the same destination in the same cluster, the intratask contention is unavoidable. We concentrate on providing an intertask contention-free network partitionability of delta class MINs in this chapter. All contention in the following context refers to intertask contention unless otherwise specified. Further evaluation of the influence of traffic localization, or network partitionability, will be given in Section 5.6.

5.2 Definition of Different Cubes

A cube, which provides an intertask contention-free partitionability on some delta class MINs, is formally defined as follows:

Definition 4 *In a MIN with $N = k^n$ nodes, a k -ary m -cube (cube) consists of k^m nodes which have the same $n - m$ radix- k digits (fixed variables) in their node addresses, where these same digits can be in any $n - m$ locations of the n possible locations. Two cubes are disjoint if they have different fixed variables and one is not a subset of the other.*

Definition 5 *A k -ary m -cube is referred to as a base k -ary m -cube (base cube) if these $n - m$ digits are in the most significant $n - m$ locations (or the remaining m digits are in the least significant m locations) of their node addresses.*

A base cube is a special case of a cube. Consider a system with $N = 4^4$ nodes. The cluster (12XX) has 16 nodes ranging from (1200) to (1233) and is a base 4-ary 2-cube. The cluster (2X1X) has 16 nodes ranging from (2010) to (2313) and is a 4-ary 2-cube but not a base cube. Note that an X represents a digit based on k , or $0 \leq X < k$, in the following context of this chapter.

Most research is based on 2×2 switches, while larger switches provide better performance in the real world. Allocating a complete k -ary m -cube to an application which requires only part of the cube may reduce the overall utilization of a system. For example, an 8-ary 2-cube will be allocated to a 16-node application on a network with 8×8 switches. To reduce such penalty, we also consider allocation of binary cubes on a network with $k \times k$ switches. The formal definitions of both binary m -cube and base binary m -cube on a MIN with $k \times k$ switches are given below.

Definition 6 *In a MIN with $N = k^n$ nodes, where $k = 2^p$, a binary m -cube consists of 2^m nodes which have the same $(p \times n - m)$ bits (fixed variables) in their node addresses (based on 2), where these same bits can be in any $(p \times n - m)$ locations of the $(p \times n)$ possible locations. Two binary cubes are disjoint if they have different fixed variables and one is not a subset of the other.*

Definition 7 *A binary m -cube is referred to as a base binary m -cube if these $(n \times p - m)$ bits are in the most significant $(n \times p - m)$ locations (or the remaining m bits are in the least significant m locations) of their node addresses (based on 2).*

Based on these definitions, a k -ary m -cube, where $k = 2^p$, is a binary $(p \times m)$ -cube. An example of different cubes is given in Figure 5.2. There are three cubes: 0xxx,

$10xx$ and $11x1$. The second cluster $10xx$, or $2X$, is a base 4-ary 1-cube, or a base binary 2-cube. The first one is a base binary 3-cube and the third one is a binary 1-cube but not a base binary 1-cube. The latter two are not 4-ary cubes. Note that in the following context of this chapter, an x stands for a digit based on 2, or $0 \leq x < 2$.

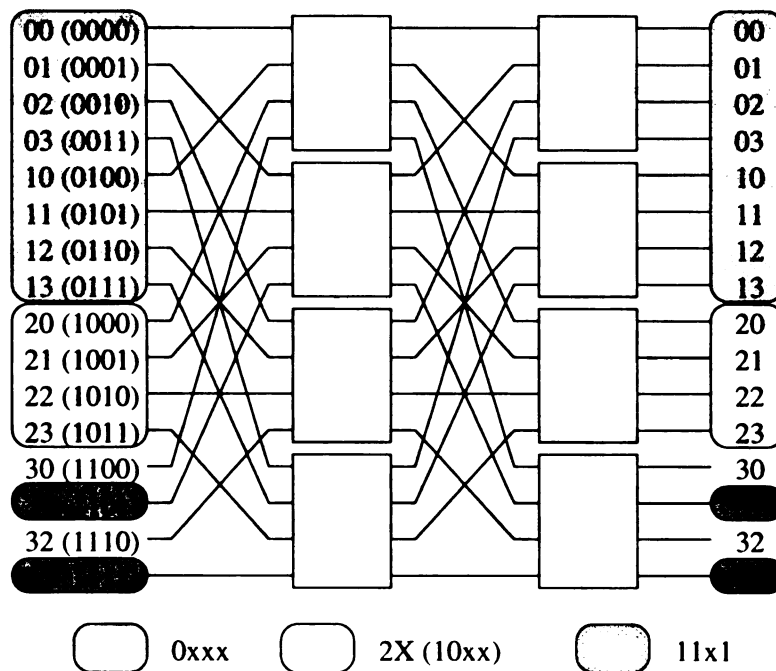


Figure 5.2: Three different processor clusters, $0xxx$, $10xx$ or $2X$, and $11x1$.

5.3 Contention Free and Channel Balanced Partition

In addition to guaranteeing contention-free network partitioning, it is important that the number of communication channels between two adjacent stages is the same as the number of nodes in the corresponding cluster. Thus, if a cluster has c nodes, the number of channels (or channel pairs) allocated to the cluster should be c between any

two adjacent stages, and this is referred as *channel-balanced allocation*. If the number of channels allocated is less than c , it implies the possibility of channel congestion within that cluster. If the number of channels allocated is greater than c , it implies that other clusters may be allocated with fewer channels or that the channels have to be shared with other clusters. Before showing this property on different delta networks, two basic lemmas will be proven. The first one is related to channel-balanced allocation and the second one concerns the external contention-free allocation. Both lemmas are given below.

Lemma 6 *A k -ary cube is channel-balanced if s_i is replaced by d_i for all $0 \leq i \leq n-1$ while the address pattern is changed from $s_{n-1} \cdots s_1 s_0$ to $d_{n-1} \cdots d_1 d_0$.*

Proof: We shall prove that the number of channels used by an m -cube cluster is always k^m between any two adjacent stages. For a k -ary m -cube, there are $n - m$ fixed variables and m free variables in the definition of the cluster. For communication within each cluster and that s_i is replaced by d_i for all $0 \leq i \leq n - 1$, all those $n - m$ fixed variables and the number of free variables m are unchanged for a given k -ary m -cube cluster. Thus, the number of channels between any two adjacent stages of an m -cube cluster remains k^m . \square

Lemma 7 *Disjoint cubes are contention-free if s_i is replaced by d_i for all $0 \leq i \leq n-1$ while the address pattern is changed from $s_{n-1} \cdots s_1 s_0$ to $d_{n-1} \cdots d_1 d_0$.*

Proof: We will show that the channels from disjoint k -ary cubes (clusters) are always distinct. Suppose that a channel is shared by two different clusters. The

channel address is thus the same from both clusters. This implies that two different clusters have the same fixed variables, a contradiction. \square

With these two lemmas, we would show that a cube network can be partitioned into contention-free and channel-balanced disjoint k -ary cubes in the following lemma.

Lemma 8 *A cube network with $N = k^n$ nodes can be partitioned into contention-free and channel-balanced disjoint k -ary cubes.*

Proof: Based on Lemmas 6 and 7, we shall prove that s_i is always replaced by d_i when the address pattern is changed from $s_{n-1}s_{n-2}\cdots s_1s_0$ to $d_{n-1}d_{n-2}\cdots d_1d_0$.

Consider a source and destination pair: $s_{n-1}\cdots s_0$ and $d_{n-1}\cdots d_0$. The channels before entering and after exiting stage $n-1$ (G_{n-1}) are $s_{n-2}\cdots s_0s_{n-1}$ and $s_{n-2}\cdots s_0d_{n-1}$, respectively. The former is due to perfect k -shuffle connection and the later is due to destination tag routing with $t_{n-1} = d_{n-1}$. For stage i ($G_i, 0 \leq i \leq n-2$), the channels entering and exiting G_i are $d_{n-1}\cdots d_{i+1}s_{i-1}\cdots s_1s_0s_i$ and $d_{n-1}\cdots d_{i+1}s_{i-1}\cdots s_1s_0d_i$, respectively. The former is due to butterfly connection β_i^k and the latter is due to destination tag routing with $t_i = d_i$. As the address pattern is changed from $s_{n-1}\cdots s_0$ to $d_{n-1}\cdots d_0$, s_i is always replaced by d_i for all $0 \leq i \leq n-1$.

\square

In a more general case, when k is a power of 2, the restriction of k -ary cubes can be relaxed to binary cubes. From the proof of Lemma 8, we can immediately obtain

the following result:

Theorem 1 *A cube network with $N = k^n$ nodes, where $k = 2^p$ for some p , can be partitioned into contention-free and channel-balanced disjoint “binary” cubes.*

Proof: We shall prove that the number of channels used by a binary m -cube cluster is always 2^m between any two adjacent stages and that the channels from different clusters (binary cubes) are always distinct.

Let k be 2^p where $p \geq 2$. Consider a source and destination pair: $s_{n-1} \cdots s_0$ and $d_{n-1} \cdots d_0$, or $s_{n-1,p-1} s_{n-1,p-2} \cdots s_{n,0} \cdots s_{1,0} s_{0,p-1} s_{0,p-2} \cdots s_{0,0}$ and $d_{n-1,p-1} d_{n-1,p-2} \cdots d_{n-1,0} \cdots d_{1,0} d_{0,p-1} d_{0,p-2} \cdots d_{0,0}$, respectively. The channels before entering and after exiting stage $n-1$ (G_{n-1}) are $s_{n-2} \cdots s_0 s_{n-1}$ and $s_{n-2} \cdots s_0 d_{n-1}$, respectively. The former is due to perfect k -shuffle connection and the later is due to destination tag routing with $t_{n-1,j} = d_{n-1,j}$, where $0 \leq j \leq p-1$. For stage i ($G_i, 0 \leq i \leq n-2$), the channels entering and exiting G_i are $d_{n-1} \cdots d_{i+1} s_{i-1} \cdots s_1 s_0 s_i$ and $d_{n-1} \cdots d_{i+1} s_{i-1} \cdots s_1 s_0 d_i$, respectively. The former is due to butterfly connection β_i^k and the latter is due to destination tag routing with $t_{i,j} = d_{i,j}$. As the address pattern is changed from $s_{n-1} \cdots s_0$ to $d_{n-1} \cdots d_0$, $s_{i,j}$ is always replaced by $d_{i,j}$ for all $0 \leq i \leq n-1$ and $0 \leq j \leq p-1$. Note that for a binary m -cube, there are $(n \times p - m)$ fixed variables and m free variables in the definition of the cluster. For communication within each cluster, it implies that $s_{i,j} = d_{i,j}$ for all those i and j of fixed variables and the number of free variables m is unchanged for a given binary m -cube cluster. Thus, the number of channels between any two adjacent stages of an m -cube cluster remains 2^m .

Lemma 7 implies that the channels from different clusters are always distinct. Hence, different clusters are contention-free.

□

An example to partition a 16-node cube network with 4×4 switches, $k = 4$, into four contention-free and channel-balanced binary cube clusters is shown in Figure 5.3. The clusters $x0xx$, $01xx$, $11x0$ and $11x1$ consist of 8, 4, 2 and 2 nodes, respectively. As shown in the figure, all these four clusters are channel-balanced and contention-free.

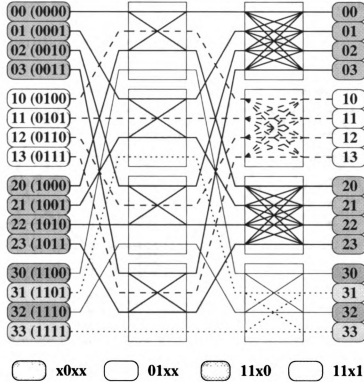


Figure 5.3: A 16-node cube network based on 4×4 switches is partitioned into four contention-free and channel-balanced binary cube clusters, $x0xx$, $01xx$, $11x0$ and $11x1$.

The Lemma 8 and Theorem 1 may apply to the omega network directly. Hence, we have Lemma 9 and Theorem 2 as following.

Lemma 9 *An omega network with $N = k^n$ nodes can be partitioned into contention-free and channel-balanced disjoint k -ary cubes.*

Theorem 2 *An omega network with $N = k^n$ nodes, where $k = 2^p$ for some p , can be partitioned into contention-free and channel-balanced disjoint “binary” cubes.*

The proof is similar to that of Lemma 8.

Figure 5.4(a) shows the partitioning of an 8-node cube network into contention-free and channel-balanced binary cube clusters: XX0, 0X1, and 1X1, while Figure 5.4(b) shows the same partitioning on an 8-node omega network.

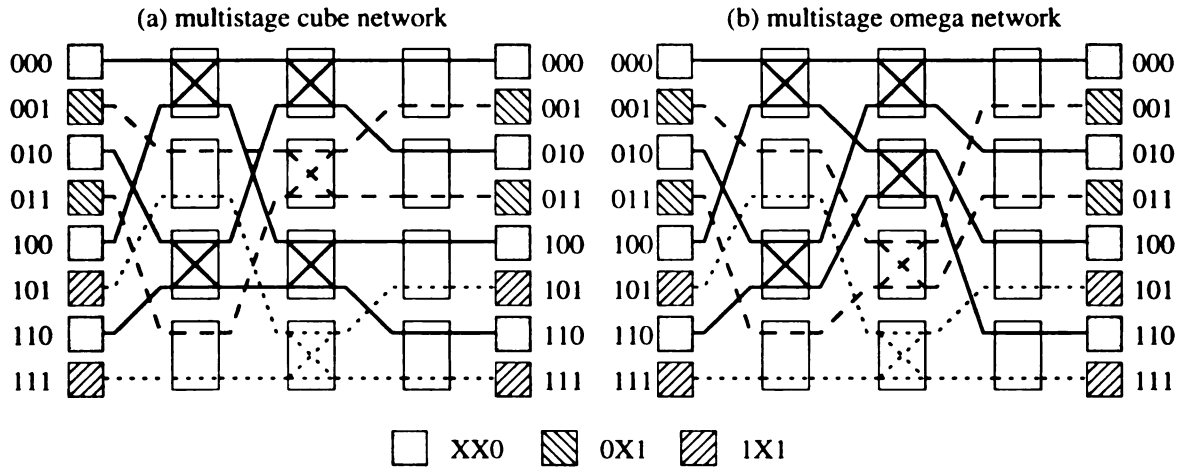


Figure 5.4: An 8-node cube network and an 8-node omega network are partitioned into three contention-free and channel-balanced binary cube clusters.

Unlike the cube network or omega network, not all delta networks can be partitioned into contention-free and channel-balanced clusters. The following section shows that neither the baseline network nor the butterfly network possess these latter properties.

5.4 Non Contention-Free Partition

Although baseline and butterfly networks are topology equivalent to omega and cube networks, the intertask contention exists on both of them. This section gives the description and formal proof of this property.

Lemma 10 *A baseline network with $N = n^k$ nodes may not be partitioned into contention-free and channel-balanced disjoint k -ary cubes.*

Proof: Consider a source and destination pair: $s_{n-1} \cdots s_0$ and $d_{n-1} \cdots d_0$. The channels before entering and after exiting stage $n - 1$ (G_{n-1} , the leftmost stage) are $s_{n-2} \cdots s_0 s_{n-1}$ and $s_{n-2} \cdots s_0 d_{n-1}$, respectively. The former is due to perfect k -shuffle connection and the later is due to destination tag routing with $t_{n-1} = d_{n-1}$. For stage i ($G_i, 0 \leq i \leq n - 2$), the channels entering and exiting G_i are $d_{n-1} \cdots d_{i+1} s_{n-2} s_{n-3} \cdots s_{n-i-2}$ and $d_{n-2} \cdots d_{i+1} s_{n-2} \cdots s_{n-i-1} d_i$, respectively. The former is due to baseline connection B_i^k , and the later is due to destination tag routing with $t_i = d_i$. As the address pattern is changed from the $s_{n-1} \cdots s_0$ to $d_{n-1} \cdots d_0$, s_{n-i-2} is always replaced by d_i for $0 \leq i \leq n - 2$ and s_{n-1} is replaced by d_{n-1} . Thus, a free variable may be replaced by a fixed variable, implying the number of channels is reduced at that stage. If a fixed variable is replaced by a free variable, this implies that more channels are used by that cluster and the channels may be shared with other clusters. \square

Two examples of partitioning a 16-node baseline network into different binary cube clusters are given in Figure 5.5. There are three contention-free clusters: $x0xx$,

010 x , and 11 x . In all three clusters, the number of channels is reduced to half at connections C_2 and C_1 as shown in Figure 5.5(a). All 16 channels at connection C_2 and C_1 are shared by two clusters, $xxx0$ and $xxx1$, in Figure 5.5(b).

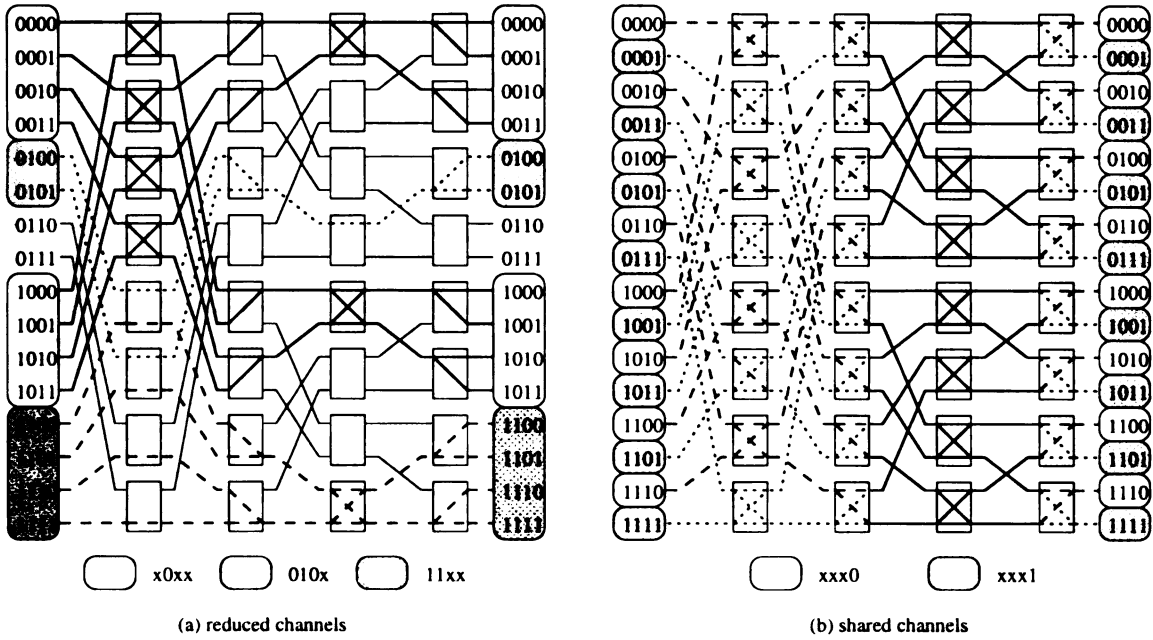


Figure 5.5: A 16-node baseline network is partitioned into different binary clusters.

Similarly, we have the following lemma for a butterfly network.

Lemma 11 *A butterfly network with $N = n^k$ nodes may not be partitioned into contention-free and channel-balanced disjoint k -ary cubes.*

Proof: Consider a source and destination pair: $s_{n-1} \cdots s_0$ and $d_{n-1} \cdots d_0$. As the address pattern is changed from $s_{n-1} \cdots s_0$ to $d_{n-1} \cdots d_0$, s_j is always replaced by d_{j+1} for $0 \leq j \leq n-2$ and s_{n-1} is replaced by d_0 . Thus, a free variable may be replaced by a fixed variable, implying that the number of channels is reduced at that stage. If a fixed variable is replaced by a free variable, this implies that more channels are used by that cluster and that the channels may be shared with other

channels.

□

Figure 5.6 demonstrates the partitioning of a 16-node butterfly network into different binary cube clusters. There are three contention-free clusters: $x0xx$, $010x$, and $11xx$. In all three clusters, the number of channels is reduced to half between stages G_2 and G_1 as shown in Figure 5.6(a). In Figure 5.6(b), there are two 8-node clusters: $xxx0$ and $xxx1$. Both clusters share those 16 channels in connections C_3 , C_2 and C_1 .

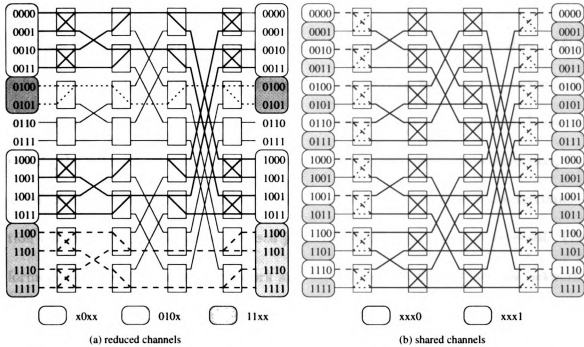


Figure 5.6: A 16-node butterfly network is partitioned into different binary clusters.

5.5 Modification of Baseline and Butterfly Networks

As discussed in previous sections, the cube and omega networks provide contention-free and channel-balanced network partitionability but not the baseline or butterfly networks. Due to the advantage of network partitionability, the cube and omega networks are highly recommend in constructing new networks. For a current system, such as NEC Cenju-3, it may not be cost effective to update the network. Hence, to provide contention-free and channel-balanced network partitionability for both baseline and butterfly networks with minimum modification becomes the goal of this section.

Since the interconnection patterns between adjacent stages are fixed after a MIN is built, we will concentrate on modifying those connections between the source node and network, C_n and C_0 . To be consistent with most delta networks, we choose to modify the leftmost interconnection, C_n , and keep the rightmost interconnection, C_0 , as an *identity* connection, which is a straight connection. Two modified interconnection patterns will be used, the *reversed perfect k-shuffle* and the *digit-reversed*. The former one is for the butterfly network and the later one is for the baseline network. Furthermore, we will show that both modified baseline and butterfly networks can be partitioned into contention-free and channel-balanced k -ary cubes.

Before giving the definition of these interconnection patterns, let us carefully examine the address transition on both baseline and butterfly networks in the following subsections.

5.5.1 Modification of a Baseline Network

Consider a source and destination pair: $s_{n-1}s_{n-2} \cdots s_0$ and $d_{n-1}d_{n-2} \cdots d_0$. As shown in the proof of Lemma 10, s_j is always replaced by d_{n-j-2} for $0 \leq j \leq n-2$ and s_{n-1} is replaced by d_{n-1} as the address pattern is changed from $s_{n-1}s_{n-2} \cdots s_0$ to $d_{n-1}d_{n-2} \cdots d_0$. The replacement of s_{n-1} by d_{n-1} is caused by the perfect k -shuffle in the leftmost interconnection. If an identity connection is used in C_n , s_j will be replaced by d_{n-j-1} for $0 \leq j \leq n-1$. In order to have s_j be replaced by d_j , we shall reverse the order of source address in the leftmost interconnection. Hence, the digit reversing pattern, γ , is introduced to achieve such purpose and is formally defined below.

Definition 8 *The digit-reversing connection γ^k is defined by*

$$\gamma_i^k(x_{n-1}x_{n-2} \cdots x_2x_1x_0) = x_0x_1x_2 \cdots x_{n-2}x_{n-1} \text{ where } 0 \leq x_i \leq k-1.$$

When a baseline network connects to the source nodes based on digit-reversing connection, it can be partitioned into contention-free and channel-balanced disjoint k -ary cubes. The formal description and proof are given in Lemma 12.

Lemma 12 *A baseline with digit reversing pattern as C_n can be partitioned into contention-free and channel-balanced disjoint k -ary cubes.*

Proof: Consider a source and destination pair: $s_{n-1}s_{n-2} \cdots s_1s_0$ and $d_{n-1}d_{n-2} \cdots d_1d_0$. The channels before entering and after exiting stage $n-1$ (G_{n-1})

are $s_0s_1 \cdots s_{n-2}s_{n-1}$ and $s_0s_1 \cdots s_{n-2}d_{n-1}$, respectively. The former is due to digit reversing connection and the later is due to destination tag routing with $t_{n-1} = d_{n-1}$. For stage i (G_i , $0 \leq i \leq n-1$), the channel entering G_i is $d_{n-1} \cdots d_{i+1}s_0s_1 \cdots s_{i-1}s_i$ due to baseline connection δ_i^k and the channel exiting G_i is $d_{n-1} \cdots d_{i+1}s_0s_1 \cdots s_{i-1}d_i$ because of destination tag routing with $t_i = d_i$.

As the address pattern is changed from $s_{n-1} \cdots s_1s_0$ to $d_{n-1} \cdots d_1d_0$, s_i is always replaced by d_i for all $0 \leq i \leq n-1$. Hence, such network can be partitioned into channel-balanced and contention-free k -ary cubes based on 6 and 7.

□

Figure 5.7 demonstrates the partitionability of a *modified baseline network*. For comparison purposes, the partitions in Figure 5.7 are identical to those in Figure 5.5 in the previous section. Those three clusters, $x0xx$, $010x$, and $11xx$, which are contention-free but in which the number of channels are reduced to half in connections C_2 and C_1 of a baseline network as shown in Figure 5.5(a), become channel-balanced in a modified baseline network as shown in Figure 5.7(a). The channel sharing of clusters $xxx0$ and $xxx1$ on a baseline network as given in Figure 5.5(b) is eliminated in a modified baseline network as shown in Figure 5.7(b).

Like the cube network, when k is a power of 2, the restriction of k -ary cubes can be relaxed to binary cubes. From the proof of Lemma 12, we can immediately obtain the following result:

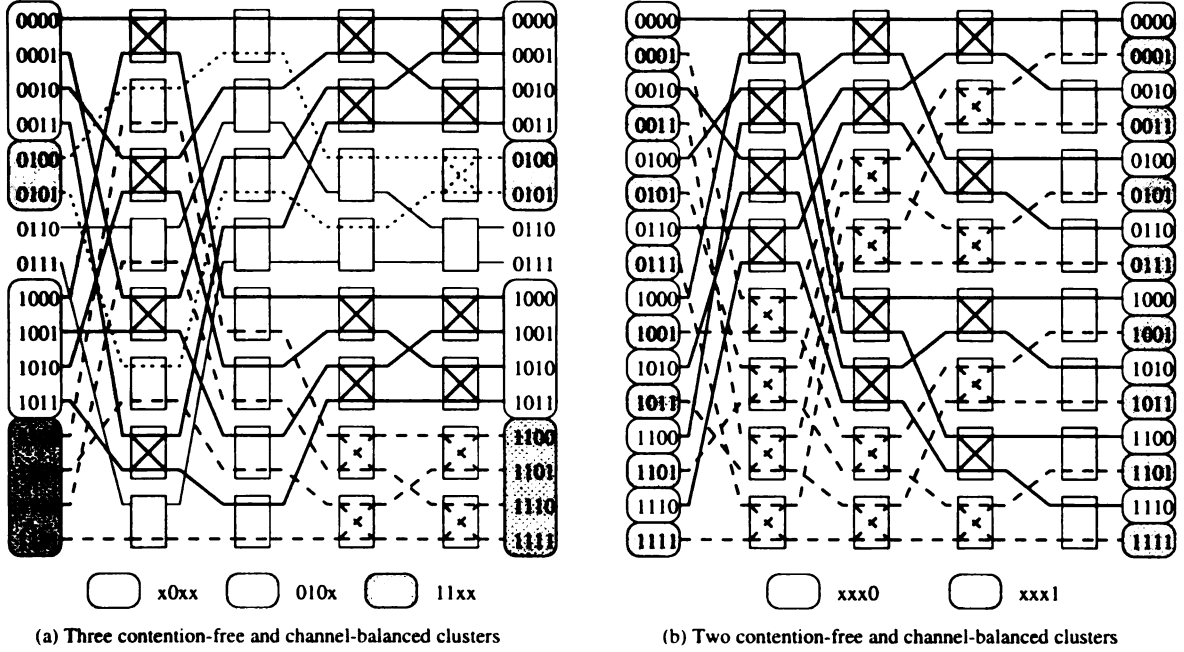


Figure 5.7: An example to partition a 16-node modified baseline network into contention-free and channel-balanced disjoint k -ary cubes.

Theorem 3 *A modified baseline network with $N = k^n$ nodes, where $k = 2^p$ for some p , can be partitioned into contention-free and channel-balanced disjoint “binary” cubes.*

5.5.2 Modification of a Butterfly Network

To examine the address transition, we consider a source and destination pair: $s_{n-1}s_{n-2}\cdots s_0$ and $d_{n-1}d_{n-2}\cdots d_0$. While the address pattern is changed from $s_{n-1}\cdots s_1s_0$ to $d_{n-1}\cdots d_1d_0$, s_j is always replaced by d_{j+1} for $0 \leq j \leq n-2$ and s_{n-1} is replaced by d_0 as shown in the proof of Lemma 11. Based on Lemmas 6 and 7, the s_j shall be replaced by d_j for $0 \leq j \leq n-1$. Therefore, the address pattern entering stage G_{n-1} shall be $s_0s_{n-1}\cdots s_1$ instead of $s_{n-1}\cdots s_1s_0$. To achieve this, a reversed perfect k -shuffle is used for connection C_n . With such modification, the

address pattern entering the stage G_{n-1} becomes $s_0 s_{n-1} \cdots s_1$. When the last digit is replaced in stage j , s_j is always replaced by d_j , for all $0 \leq j \leq n-1$. The definition of reversed perfect k -shuffle and the formal description as well as the proof of such property are given below.

Definition 9 *The reversed perfect k -shuffle connection $\bar{\sigma}$ is defined by*

$$\bar{\sigma}_i^k(x_{n-1}x_{n-2} \cdots x_2x_1x_0) = x_0x_{n-1}x_{n-2} \cdots x_2x_1 \text{ where } 0 \leq x_i \leq k-1.$$

Lemma 13 *A butterfly network with reverse perfect k -shuffle as the leftmost connection, $C_n = \bar{\sigma}^k$, can be partitioned into contention-free and channel-balanced disjoint k -ary cubes.*

Proof: Consider a source and destination pair: $s_{n-1}s_{n-2} \cdots s_1s_0$ and $d_{n-1}d_{n-2} \cdots d_1d_0$. The channels before entering and after exiting stage $n-1$ (G_{n-1}) are $s_0s_{n-1} \cdots s_2s_1$ and $s_0s_{n-1} \cdots s_2d_1$, respectively. The former is due to reversed perfect k -shuffle connection and the later is due to destination tag routing with $t_{n-1} = d_1$. For stage $n-i$ (G_{n-i} , $1 \leq i \leq n-1$), the channel entering G_i is $s_0s_{n-1} \cdots s_{i+1}d_{i-1} \cdots d_1s_i$ due to butterfly connection β_i^k and the channel exiting G_i is $s_0s_{n-1} \cdots s_{i+1}d_{i-1} \cdots d_1d_i$ because of destination tag routing with $t_i = d_{n-i}$. Similarly, the channels entering and exiting the rightmost stage G_0 are $d_{n-1}d_{n-2} \cdots d_1s_0$ and $d_{n-1}d_{n-2} \cdots d_1d_0$, respectively. The later is due to the destination routing tag $t_0 = d_0$.

As the address pattern is changed from $s_{n-1} \cdots s_1 s_0$ to $d_{n-1} \cdots d_1 d_0$, s_i is always replaced by d_i for all $0 \leq i \leq n-1$. Hence, the number of channels between any adjacent stages of a k -ary m -cube remains k^m and different clusters are contention-free based on Lemmas 6 and 7.

□

Those examples which partition a 16-node butterfly network into a different number of clusters in Figure 5.6 are revisited on a modified butterfly network. As shown in Figure 5.8(a), those three contention-free cube clusters, $x0xx$, $010x$ and $11xx$, in which the number of channels is reduced to half in connection C_2 on a 16-node butterfly network, become contention-free and channel-balanced clusters on a 16-node modified butterfly network. Those two clusters, $xxr0$ and $xxr1$, which share channels in Figure 5.6(b) become contention-free as shown in Figure 5.8(b).

Like the modified baseline network, the restriction of k -ary cubes can be relaxed to binary cubes when k is a power of 2. We can immediately obtain the following result from the proof of Lemma 13.

Theorem 4 *A modified butterfly network with $N = k^n$ nodes, where $k = 2^p$ for some p , can be partitioned into contention-free and channel-balanced disjoint “binary” cubes.*

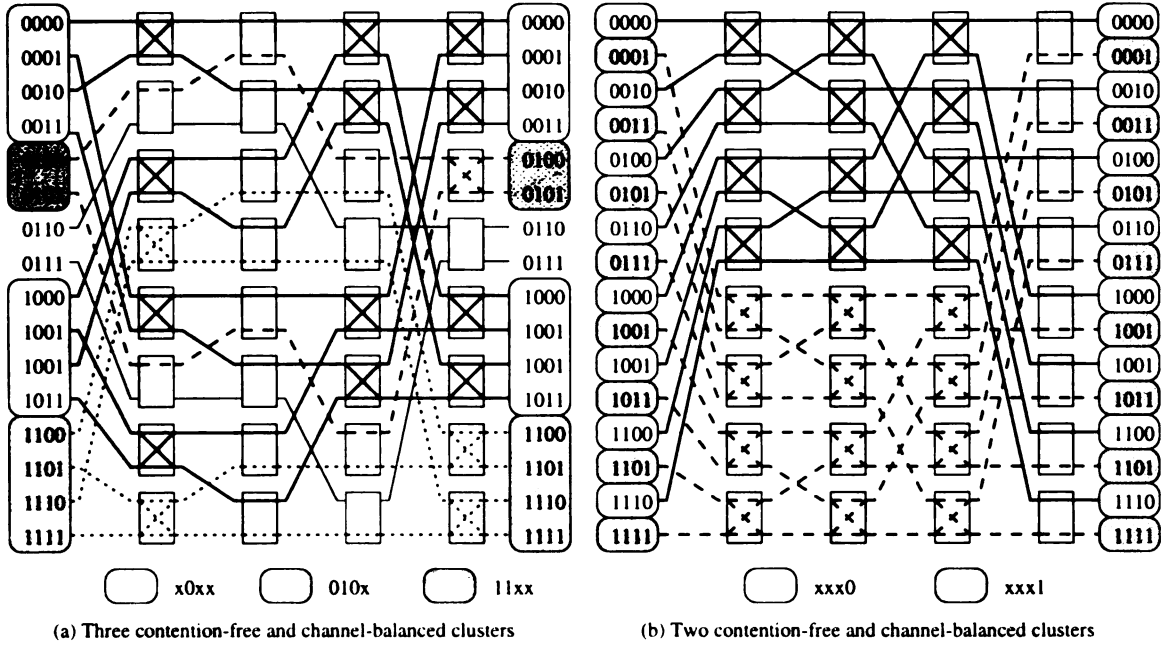


Figure 5.8: An example to partition modified butterfly network into contention-free and channel-balanced disjoint k -ary cubes.

5.6 Performance Evaluation

As discussed earlier, enforcing traffic localization will reduce the intertask contention and decrease the communication latency. In this section, we examine the influence of traffic localization in detail. The performance evaluation is based on a simulator which emulates a 64-node cube network with 4×4 switches. The length of a message is uniformly distributed between 32 and 96 flits. Note that based on Lemmas 8, 9, 12, and 13 as well as Theorems 1, 2, 3, and 4, this result also applies to omega, modified baseline and modified butterfly networks.

The latency among different intertask contention introduced by different allocation approaches is given in Figure 5.9. The latency of a complete cube confirms that binary cube partitioning is intertask contention-free. Such latency depends on the length of a message and the number of stages but not on the number of destinations or the

communication ratio.

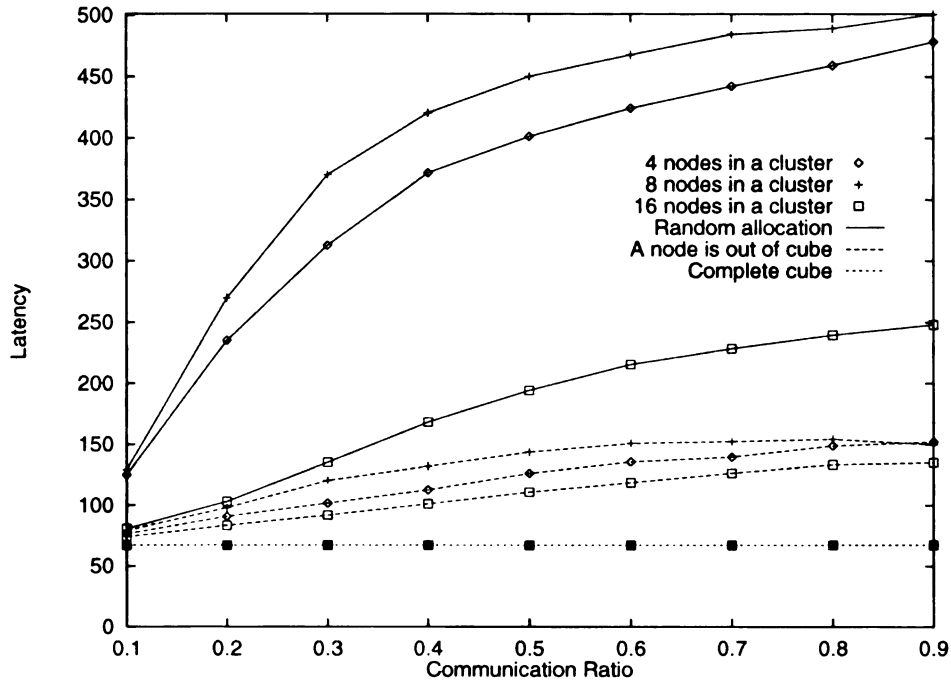


Figure 5.9: Average latency of a blocking multicast message with various intertask contention.

Intuitively, where the number of clusters increases, the potential intertask contention increases. The worst case occurs when each cluster blocks every other cluster and is itself blocked by another cluster. Hence, the average latency becomes close to $(m \times T_{comm})$ when the communication ratio R gets larger. This observation is confirmed by the random allocation of 8-node and 16-node tasks in Figure 5.9. However, when the number of nodes in one cluster is less than the number of clusters in the network, a cluster will not be able to block all other clusters. Hence, the latency decreases. This estimation is confirmed by the 4-node clusters in Figure 5.9.

The network throughput is given in Figure 5.10 for reference. Here, the throughput is defined as the average number of flits passing an outgoing channel per time unit. For a binary m -cube, the throughput is equal to $R \times \frac{2^m - 1}{2^m}$ since there are $(2^m -$

1) destinations. For example, when the communication ratio is 0.8, the expected throughput of a binary 2-cube is 0.6 ($0.8 \times 3/4$). As expected, the throughput of binary cube allocation is close to the expected throughput. The difference is due to the header overhead. The throughput of the other two approaches becomes worse as either the communication ratio increases or the latency increases.

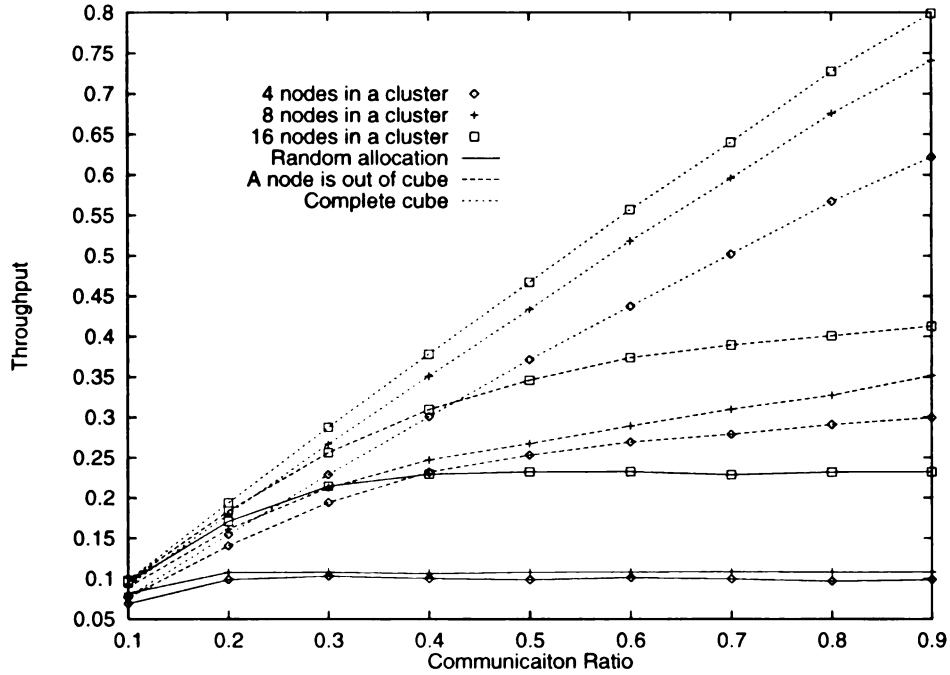


Figure 5.10: Average throughput in cube network with average 64-flit messages.

In non-blocking multicast, the source node continues to the computation phase as soon as the multicast communication request is issued to the router. The computation time and communication time are overlapped to reduce the communication overhead. Hence, intratask contention becomes possible since a communication request may be issued before the previous one is done. The influence of such intratask contention can be observed from the cube allocation in Figure 5.11, which gives the average latency of a non-blocking multicast communication based on various allocation approaches.

As the communication ratio becomes larger, the latency increases. A system becomes unstable when R closes to 0.5.

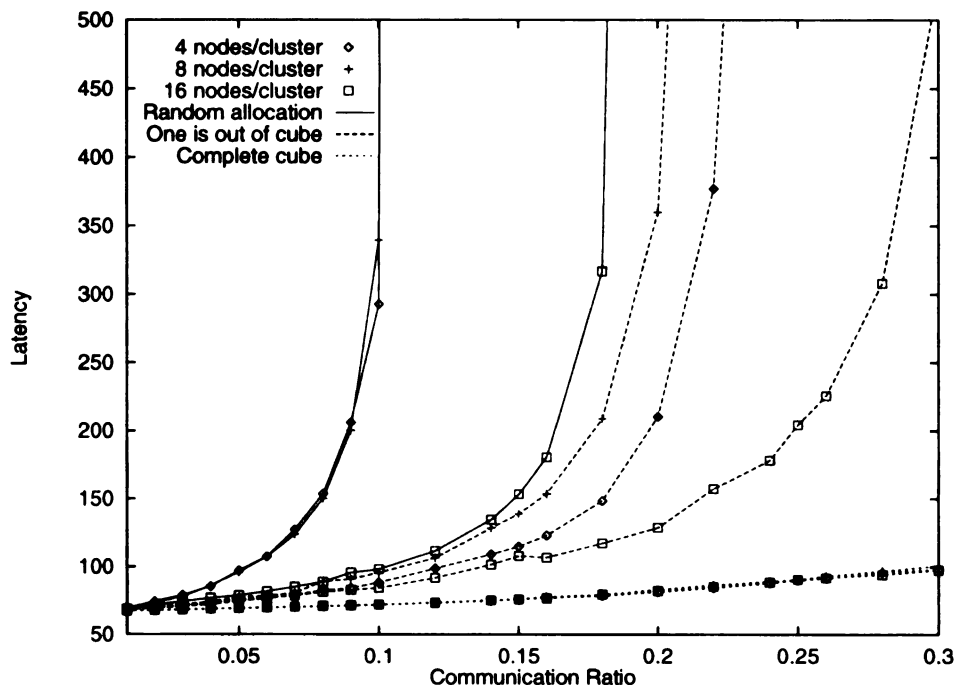


Figure 5.11: Average latency in cube network of non-blocking message with average 64-flit length.

The latency gets worse when intertask contention exists. As shown in this figure, the system becomes unstable when R is close to 0.1 if 4-node or 8-node tasks are randomly allocated. The performance of randomly allocated 16-node tasks is better than the previous two since it has lower intertask contention.

Figure 5.12 gives the average latency of non-blocking multiple multicast communications. Here, 8-node tasks are allocated into a 64-node cube network. Every node may initiate its own non-blocking multicast communication. To avoid network saturation, the network load is used as the X-coordinator. The load is defined as the average number of flits injected into the network times the number of destinations. In other words, the load is the average number of flits expected to pass through an

outgoing port per time unit. The relation of load and communication ratio R for a binary m -cube is given below.

$$R = \frac{\text{load}}{2^m - 1}$$

Hence, $R = \text{load}/7$ in Figure 5.12. As expected, the complete cube still has the best performance. However, the system reaches saturation when the communication ratio becomes 0.1.

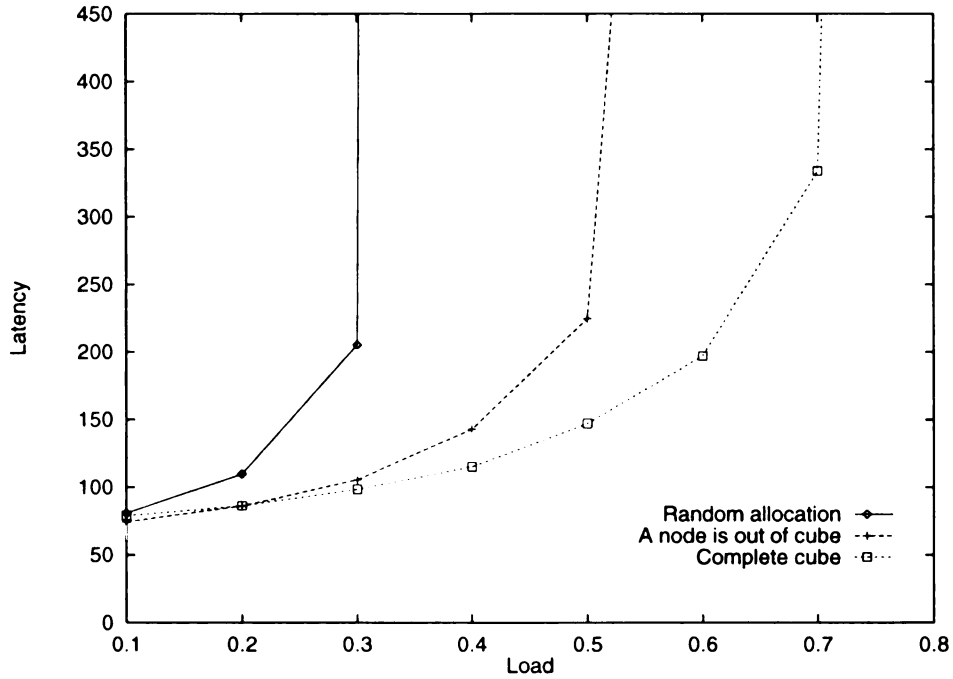


Figure 5.12: Average latency in cube network of non-blocking message with multiple sources.

5.7 Summary

Increasing the network contention increases the communication latency. Such network contention includes both the intertask contention (internal contention) and the

intratask contention (external contention). However, the intratask contention is unavoidable when multiple multicast communications are allowed. Hence, we proposed an allocation scheme, *binary cube* scheme, to eliminate the intertask contention in this chapter.

The binary cube partition algorithm provides an intertask contention-free environment on both the cube and omega networks but not on the baseline nor the butterfly networks. Two interconnection patterns are introduced to reconfigure the baseline and butterfly networks. With such modification, both baseline and butterfly networks can be partitioned into contention-free and channel-balanced binary clusters.

Due to the unique path property of delta class MINs, the path of a multicast communication is also unique. Network partitionability and traffic localization reduce the contention to increase the performance. In the next chapter, we will discuss establishing a multicast tree on an extension of MIN.

CHAPTER 6

Extra Stage Multistage Interconnection Networks

In this chapter, we show that unlike in the delta class MINs, the extra-stage MIN (ESMIN) structure opens up alternate routing strategies with varying tradeoffs. We assume one such routing approach in our work. Then, we estimate the number of multicast trees that can be generated for a given multicast problem (*i.e.*, source and destination list) and an extra-stage design. We discuss the design implications of the “extra stages” with regard to the multicast attributes, and develop an optimality criterion for multicast trees in an ESMIN. Thus, enumeration of *all* the instances of optimum-traffic tree requires exponential complexity. However, a single instance of the optimum-traffic tree can be generated in polynomial time. A multicast tree generation algorithm is proposed towards this.

Figure 6.1(a) gives an extra stage MIN structure constructed by 2×2 switches. It has $N = 2^n$ input ports and N output ports. Unlike n stages in a delta class MIN,

there are h stages, where $h \geq n$. Similar to delta MINs, the h stages are placed horizontally apart, while within each stage there are $\frac{N}{2}$ switches vertically stacked. Each 2×2 switch can connect in one of the four possible ways (Figure 6.1(b)): *straight*, *crossed*, *upper broadcast*, and *lower broadcast*.

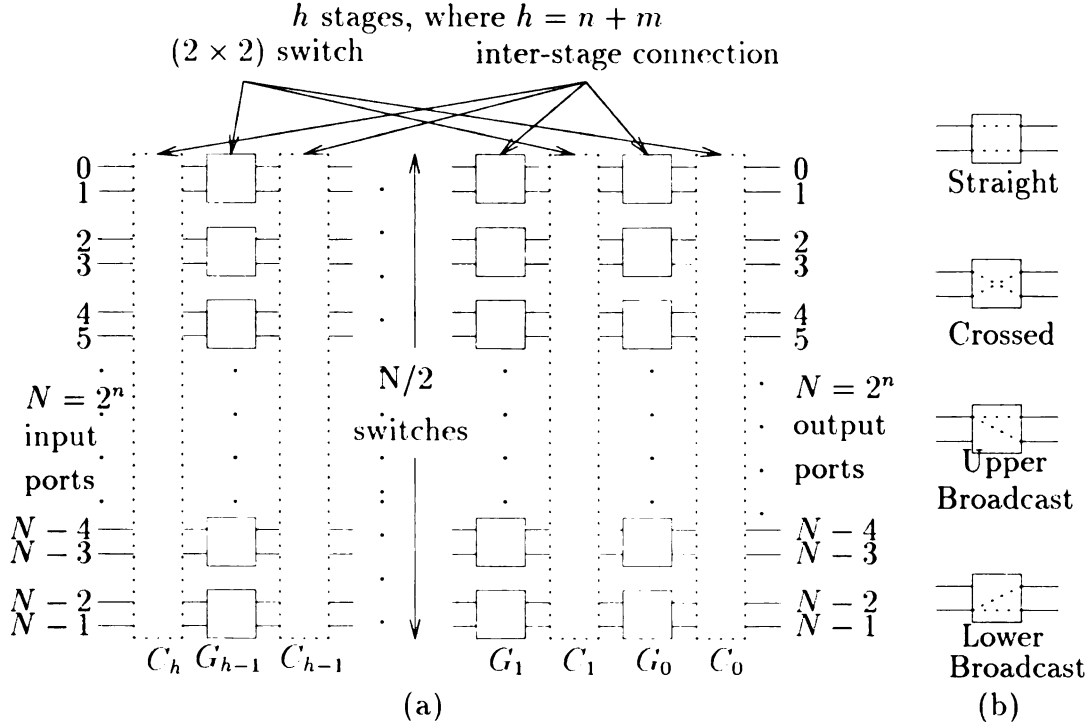


Figure 6.1: A generic MIN structure with $N = 2^n$ input/output ports and h stages, where each 2×2 switch has four connectivity choices.

We consider a multicast communication from a source node S to k destination nodes, $D_i, 1 \leq i \leq k$ for $k \leq N$. Such a problem is usually characterized using two metrics: *time* and *traffic* [22]. *Traffic* is a measure of the local number of messages generated and passed from one node to another in the system. *Time* measures the maximum number of hops taken by the message to reach any destination. Since several inter-node message communications may take place simultaneously, *time* does not measure the total number of messages. Our objective is to develop a multicast

routing approach for the general class of ESMINs as shown in Figure 6.1. The actual connection patterns between the intermediate stages may vary, and the proposed multicast approach is expected to be generic to the connection patterns given in Chapter 3, *e.g.*, the perfect-shuffle or the butterfly connection patterns.

6.1 MINs with Extra Stages

Figure 6.1 shows a generic MIN with N input ports and N output ports, where $N = 2^n$. It has h stages, G_{h-1} to G_0 . As shown in Figure 6.1, each stage, say G_i , has $\frac{N}{2} 2 \times 2$ switches. The connection between two adjacent stages, G_i and G_{i-1} , denoted as C_i , defines the connection pattern for N links. Thus, an h -stage MIN can be represented as

$$C_h G_{h-1} C_{h-1} G_{h-2} \dots G_0 C_0.$$

In this chapter, $\text{MIN}(N)$ refers to a multistage interconnection network with $N = 2^n$ input/output ports and $h = n$ stages. An extra-stage MIN with $N = 2^n$ input/output ports and $h = n + m$ stages (i.e., m extra stages) is represented as $\text{ESMIN}(N:h)$.

6.1.1 Interstage Connection Patterns

A connection pattern C_i defines how those N links should be connected between the N outputs from stage G_i and the N inputs to stage G_{i-1} . Different connection patterns

give different characteristics and topological properties. Note that the connection pattern between a consecutive pair of stages, including C_h and C_0 , can vary, and our study is not restricted to any specific connection pattern. However, the connection pattern must be a variant of the *hypercube derived network*, such as binary cube, perfect shuffle, and butterfly. The links are labeled from 0 to $N - 1$ at C_i . With $N = 2^n$ ports, let $X = x_{n-1}x_{n-2} \dots x_0$ be an arbitrary port number, $0 \leq X \leq N - 1$.

As specified earlier, the butterfly connection, β_p interchanges the zero-th and p -th bits of the index. The value of p is referred to as *butterfly dimension* or *dimension* for short if there is no confusion in the context. Note that $\beta_{(0)}$ defines a straight one-to-one connection and is also called *identity connection*, I . The perfect shuffle connection, π , is defined by

$$\pi(x_{n-1}x_{n-2} \dots x_1x_0) = x_{n-2} \dots x_1x_0x_{n-1}$$

when $N = 2^n$.

An N -port cube MIN (denoted as CU-MIN(N)) is defined as $C_h = \pi$ and $C_i = \beta_{(i)}$ for $h - 1 = n - 1 \geq i \geq 0$. An *Omega network* is a MIN(N) with $C_i = \pi$ for $h = n \geq i \geq 1$ and $C_0 = I$ [33]. Such a network is denoted as PS-MIN(N).

6.2 Structural Equivalence of different Delta Networks

The upper part of Figure 6.2(a) shows a PS-MIN(16). The corresponding lower part shows a CU-MIN(16). Note that if we have $C_h = \pi$ in CU-MIN, the CU-MIN is equivalent to the multistage cube network [32]. With destination-tag routing, the leftmost stage, C_h , makes no difference between CU-MINs and multistage cube networks. Thus, we assume $C_h = I$ for both MIN and ESMIN in this chapter.

The PS-MIN and CU-MIN are structural equivalence as shown in Fig. 6.2(a) with $N = 16$. The PS-MIN involves perfect shuffle (which is left shifting the bit-id) and exchange (introducing a 0 or 1 bit at the least significant bit) operations. A bare exchange operation corresponds to a dimension-0 toggle ($\equiv \beta_1$) of the CU-MIN. An exchange *followed by a number of perfect shuffles* essentially emulates the role of an intermediate stage dimension toggle of the CU-MIN. A formal proof in this regard can be carried out, but this property is either well known or can be derived easily from existing results [31]. Thus, PS-MIN and CU-MIN can be labeled as of equivalent capability. However, we will show that they differ in their flexibility in extra stage attachments (see Section 6.2.2).

6.2.1 Design of Extra-Stage MINs

The regular MIN structure can be augmented in a number of ways. First, the destination nodes can be connected back to the source nodes using cyclical wrap around links. This class of networks has become popular for high-speed networks [45]. Another de-

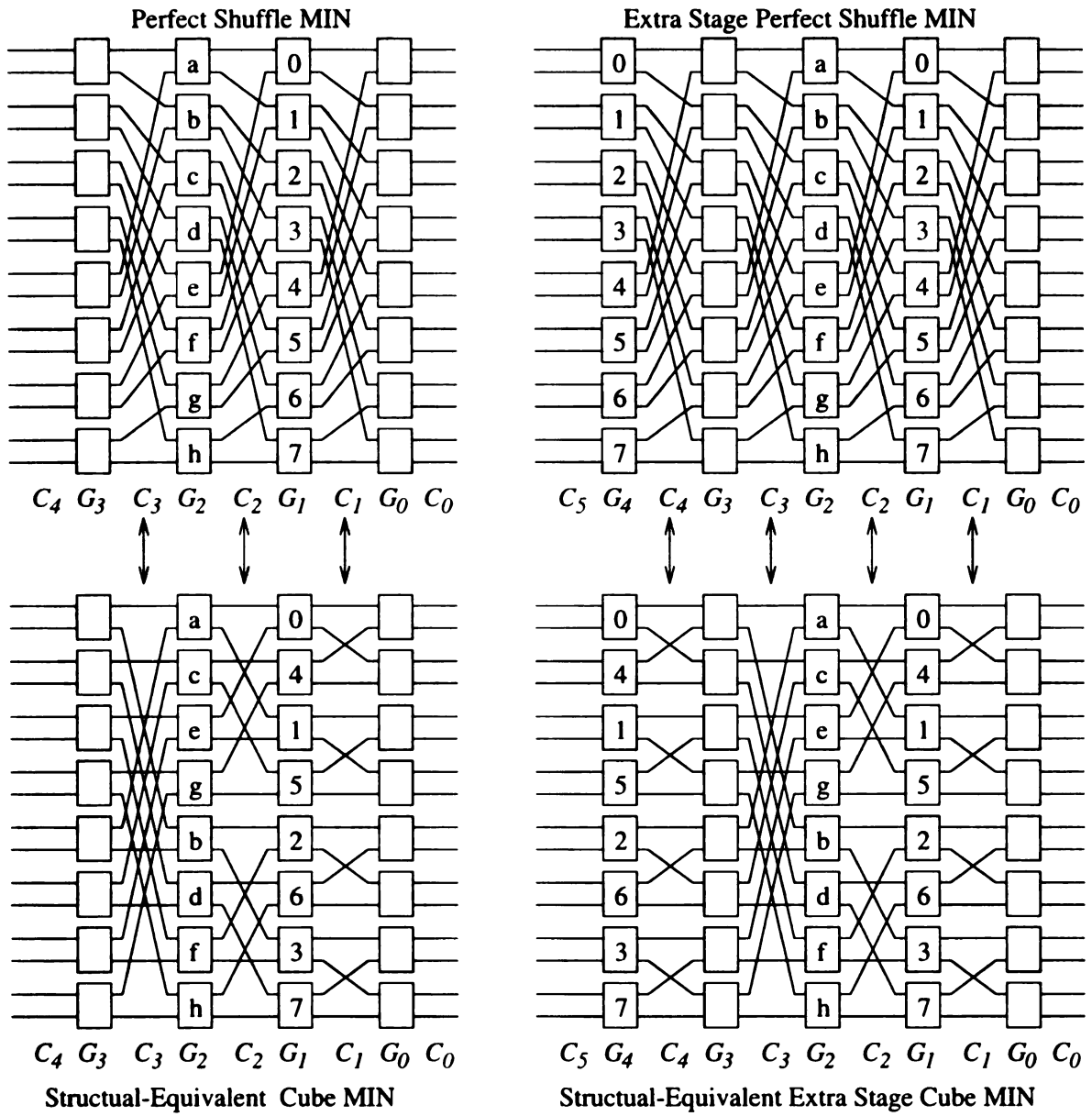


Figure 6.2: Structural equivalence between (a) PS-MIN(16) and CU-MIN(16) and (b) PS-ESMIN(16:5) and CU-ESMIN(16:1,3,2,1,0)

sign extension, in the context of optical passive star based firmware connections, is to allow dynamic on-line dimension selection in the CU-MIN networks. The multicast problem in such designs equates to an optimum ordering of the CU-MIN dimensions.

In this chapter we consider a third type of design extension to the MIN structure, where a few extra stages are added beyond the n stages of the regular MIN. The extra stages bring forth various advantages and flexibilities, one of which namely the multicast traffic reduction opportunity, is the focus of this chapter. Figure 6.2(b) shows two ESMIN designs, the upper part using perfect shuffle connectivity (PS-ESMIN) and the lower part using butterfly connectivity (CU-ESMIN).

We note that the ESMINs can offer significant advantages at the expense of limited design overheads. First, we discuss the overheads and then list out some of the advantages.

The only design overhead in an ESMIN is the hardware cost of the extra stages. Packing density of VLSI technology allows a large number of switch components to be put in a single chip. Thus, a number of extra stage switches and their interconnections may be expected to be packed in a single chip. Note that the number of pin connections (i.e., input and output nodes) to the chip remains identical regardless of the number of stages.

The number of hops between a source node and a destination node increases in the ESMIN design. A regular MIN with n stages involves n hops, while an ESMIN with $h = n + m$ stages would require m extra hops. This additional hops delay may indeed be an overhead with store-and-forward routing. However, with recent cut through switching or similar approaches (e.g., wormhole routing, ATM cell flow), this delay

becomes insignificant. Note that in cut through switching, the overall delay becomes largely insensitive to the number of hops [24], and therefore, no significant routing overhead is expected with ESMINs.

ESMINs can offer a wide range of advantages which basically stem from the number of alternate paths that are generated due to the extra stages. These alternate paths have been explored in the context of hot-spot reductions, fault-tolerant MINs, and congestion controlled traffic flow. Detailed discussion along this line is beyond the scope of this chapter. In this chapter, we exploit a similar advantage of the “alternate paths” in the context of generating multicast trees.

6.2.2 Design Choices

In principle, an ESMIN can be designed with any connection pattern between its stages. An $(h = n + m)$ -stage ESMIN can have the first n stages of the butterfly patterns and the remaining m stages of the shuffle pattern, or vice versa. For the sake of simplicity, we assume that one single connection pattern (e.g., either shuffle or butterfly) is followed over the $(n + m)$ stages. Thus, among the butterfly and PS patterns, we could either have a PS-ESMIN($N:n + m$) or a CU-ESMIN($N:n + m$). Two sample $(4+1)$ -stage ESMINs are shown in Figure 6.2(b).

Between a PS-ESMIN and a CU-ESMIN, we show that the latter has an added flexibility. Specifically, a CU-ESMIN can have *any* dimension at *any* of the ‘extra’ stages. However, the PS-ESMIN must retain a periodicity among its effective dimensions. Figure 6.2(a) shows a PS-MIN(16) and its equivalent CU-MIN(16). The cor-

respondence between the shuffle-exchange stages and butterfly dimensions are shown using double arrowed lines. Now, an extra stage is added to the 4-stage shuffle in Figure 6.2(b). The effect of the last stage is shown using an equivalent CU-ESMIN(16:5).

Normally, if an extra stage is added to a CU-MIN, the extra stage can be assigned to *anyone* of the $n - 1$ dimensions except dimension 0 (i.e., $\beta_{(0)} = I$). However, if extra stages are added to a PS-MIN, the effective *repeat* dimension occurs cyclically, i.e., a fixed dimension. In other words, for a CU-ESMIN(8:3+2), suppose its first three dimensions are 2,1,0. The other two dimensions can be any one of the two nonzero dimensions. For example, the 3+2-stage CU-ESMIN dimensions can be (2,2,2,1,0), (1,2,,2,1,0), (1,1,2,1,0), etc.

Thus, for an N -port and h -stage CU-ESMIN, it is represented as CU-ESMIN($N:C_{h-1}, C_{h-2}, \dots, C_0$) in order to show the butterfly dimension at each stage. Note that C_0 must be $\beta_{(0)}$; otherwise, not all destinations can be reached. On the contrary, a 3+2-stage PS-ESMIN would always be equivalent to CU-ESMIN(8:2,1,2,1,0). The periodic sequence (2,1) and fragments thereof would always be the ordering.

Hence, we observe that for ESMINs, the CU-ESMINs offer an added degree of flexibility over PS-ESMINs, because the *extra* stages in PS-MIN are apparently restricted in their dimensional roles. However, if similar extra stages are added to a CU-MIN, the extra stages can assume *any* dimension - which is an added degree of flexibility.

From Figure 6.2 and the discussion above, we observe that a PS-ESMIN must include a periodicity in its *effective dimensions*. Thus, a PS-ESMIN(16:12) can be considered equivalent to the CU-ESMIN(16:2,1,3,2,1,3,2,1,3,2,1,0). However, the 12-

stage CU-ESMIN(16:12) could have had other dimension choices. In other words, the 12-stage PS-ESMIN(16:12) is necessarily an instance of the 12-stage CU-ESMIN. While the latter could have any dimension pattern, a particular pattern of the latter equates that of the former.

In this chapter we consider the generic CU-ESMIN for most illustrations and proof techniques. Since the PS-ESMIN is a particular instance of CU-ESMIN, the PS-ESMIN shares an identical set of properties. Similarly, in the performance section we report the simulation results for various dimension patterns of the CU-ESMIN. *One of these dimension patterns, we have so chosen, that it fits the periodicity requirement of the PS-ESMIN.* In this way, we also report the simulation results for the PS-ESMIN (refer to “Pattern 1” in Section 6.5).

6.3 Multicast in Extra-Stage MIN

Unlike regular MIN, routing in ESMIN is no longer a trivial issue. A number of alternate routing styles may be adopted with varying tradeoffs. In this chapter we adopt one of these alternatives. For an ESMIN, the number of *alternate* multicast trees grows exponentially with the number of extra stages. Among these exponential numbers of multicast tree alternatives, an equally large (in order notation) number of multicast trees can be considered as traffic-optimal. A traffic-optimal multicast tree is a multicast tree with a minimum number of channels used. Therefore, enumeration of *all* traffic-optimum multicast trees requires an exponential complexity.

However, though there are exponential number of traffic optimal multicast trees,

a single traffic-optimum multicast tree can be derived in polynomial time. The next section develops an algorithm which can yield a traffic-optimum multicast tree in polynomial time.

6.3.1 Alternate Routing Styles in Extra-Stage MINs

Routing in a regular MIN is a trivial issue (since path unique) and requires little attention. However, routing in ESMIN no longer remains so simplistic. An example of conventional routing (source \oplus destination) is given in Figure 6.3. In this figure, part (a) shows the XOR routing on MIN(8) and part (b) shows the XOR on CU-ESMIN(8:PS¹,2,1,2,1,0), in which a source node 1 to destination node 2 routing is required. The XOR routing tag is $xx011$, where x can be either 0 or 1, since the first two stages are extra, but the conventional routing (source \oplus destination based) leads the message to destination 4, 5, 6, or 7 instead of 2 as shown in part (b). This is clearly a malfunctioning of the traditional routing (source \oplus destination), and requires further modifications.

We have identified a fair number of alternatives, among which two routing styles (namely, the “destination routing” (DR), and exclusive-OR with LSB fixing (XOR+LSB) can provide successful routings in ESMIN. In the remainder of this chapter, the DR approach (pseudo-code given in Section 6.3.2 below) is used.

¹The XOR works on MIN only if the rightmost connection is a perfect shuffle pattern and the destination routing has no such constraint.

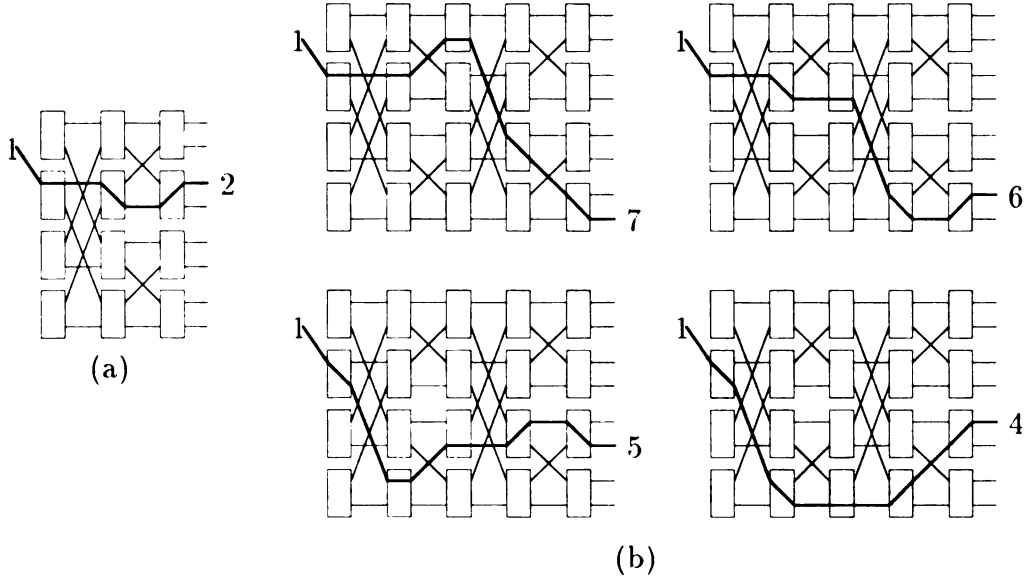


Figure 6.3: An example of the inadequacy of the traditional routing approach (source \oplus destination): A source node 1 to destination node 2 route does not lead to the right destination.

6.3.2 Distributed Routing and Multicast Implementation

A message multicast (which includes routing, as a special case) in an ESMIN is initiated by the source node. The message sent by a source node contains *a routing tag set* and *data*. Let σ be the number of routing tags for a message. It is k in the source node and will gradually decrease as the message progresses through the stages, i.e., in every split, towards the destination. Let $t_{i,j}$ be the routing tag for the destination D_i , $0 \leq i \leq k-1$ and $0 \leq j \leq h-1$ and t_i be the vector $\{t_{i,j} | 0 \leq j \leq h-1\}$. Note that $t_{i,j}$ bears the following interpretation to the intermediate stage switches:

$$t_{i,j} = \begin{cases} 0 & \text{go up (port 0)} \\ 1 & \text{go down (port 1)} \\ x & \text{go either up or down at random} \\ s & \text{go straight to the next switch in the same row} \end{cases}$$

The source node generates the $t_{i,j}$ tag values using the following destination-address based multicast algorithm (set $k = 1$ in this to obtain the DR algorithm).

Procedure *Destination-address-based Multicast*

1. Let $t_{i,j} = x$, $0 \leq j \leq h-1, 0 \leq i \leq k-1$
2. Let $\{\delta_j = g \mid C_{g-1} = \beta_j \text{ and } C_i \neq \beta_j \text{ for } 0 \leq i < g-1\}$, $0 \leq j \leq n-1$
3. Let $t_{i,\delta_j} = d_{i,j}$, $0 \leq j \leq n-1, 0 \leq i \leq k-1$

Once the $t_{i,j}$ tags are generated by the source node, these tags are used by the switches at the intermediate stages to derive the multicast (or routing) path. The routing algorithm for a switch on stage G_j , $h-1 \geq j \geq 0$ is specified in the following.

A final comment applies to the *distributed* nature of the above DR approach. Strictly speaking, the above approach is not completely distributed, since the source node pre-calculates the entire multicast (or routing) path. The very mechanism of calculating all the tags at the source node itself is an instance of non-local decision making (hence centralized). However, the intermediate stage switches operate in a distributed fashion, which makes the overall algorithm a mix of distributed and centralized fashions.

This apparently centralized decision making at the source node has other advan-

Input: A message contains a set of routing tags, $T = \{t_i | 0 \leq i \leq \sigma - 1\}$, and data.

Output: Data and two disjoint routing tag sets, L and U , for output port 0 and 1, respectively.

Procedure *switch*

Let $\sigma = |T|$, $U = \emptyset$ and $L = \emptyset$

if $t_{0,j} = s$ **then** /* all $t_{i,j}, 0 \leq i \leq \sigma - 1$, will be s */

if (port 0 connects to the next switch in the same row) $U = T$

else $L = T$

else if $t_{0,j} = x$ **then** /* all $t_{i,j}, 0 \leq i \leq \sigma - 1$, will be x */

for each $t_i, 0 \leq i \leq \sigma - 1$

 Append t_i into either U or L at random

else /* $t_{i,j}, 0 \leq i \leq \sigma - 1$ will be either 0 or 1 */

for each $t_{i,j}, 0 \leq i \leq \sigma - 1$

if $t_{i,j} = 0$ **then** $U = U \cup \{t_i\}$

else $L = L \cup \{t_i\}$.

endif

if $U \neq \emptyset$ **then** send U and data to output port 0.

if $L \neq \emptyset$ **then** send L and data to output port 1.

tages, e.g., eliminating complex processing (and/or possibly table lookup operations) at the intermediate stage switches. This issue, reflecting the traditional tradeoff between truly distributed algorithms vs the run-time processing complexity at intermediate nodes, is left for future studies.

6.3.3 Number of Multicast Trees: Upper Bound

Let CU-ESMIN($N : h$) have n -dimension, where $n = \log_2 N$ and $h \geq n$, and dimension i repeat c_i , $c_i \geq 1$ and $0 \leq i \leq n - 1$, times. For $h = n$, it is a CU-MIN and all c_i s equal 1. When $h > n$ on a PS-ESMIN, $c_i = \lfloor \frac{h}{n} \rfloor + x$, where $x = 1$ for $i \geq n - (h \bmod n)$ or $x = 0$ otherwise. Note that such closed form expressions cannot be derived for CU-ESMINs, due to the “dimension assignment flexibilities at the extra stages.”

Let us consider a particular destination D_j , among the k destinations ($0 \leq j \leq k-1$). Based on the destination D_j , a particular dimension i may be a 1 or a 0. We list these two cases in the following:

1: At the latest stage that corresponds to dimension i , the route must go *up*. At all other i^{th} dimensional stages, the route may go either way, i.e., up or down.

This leads to 2^{c_i-1} possible paths.

0: At the latest stage that corresponds to dimension i , the route must go *down*. At all other dimension i stages, the route may go either way. This also leads to 2^{c_i-1} possible paths.

Therefore, for dimension i , the route has 2^{c_i-1} flexibility. Hence, for all the n dimensions (regardless of whether they are ‘1’ or ‘0’), a maximum of $\prod_{i=0}^{n-1} 2^{c_i-1}$ ($= 2^{h-n}$) routes can be formed. For k destinations, an upper bound on the multicast trees is

$$(2^{h-n})^k$$

6.3.4 Multicast Tree Selection Criteria

An optimality criterion is proposed, in multicast tree selection, with primary focus on the ‘traffic’ and ‘time’ metrics. The ‘time’ metric, for an ESMIN(h), would always equal h . Thus, there is no opportunity for ‘time’ reduction (besides the fact that with cut-through switching the ‘time’ metric or hops-count largely loses its significance).

Therefore, we equate traffic minimization to the optimality criterion of multicast tree selection. This is stated as follows:

Given an ESMIN of $n + m$ stages, a source node S and k destinations $(D_j, 0 \leq j \leq k - 1)$ generate a multicast tree that minimizes the total number of tree edges.

6.3.5 Traffic Optimal Multicast Trees

The number of possible multicast trees grows exponentially with extra stages, and so do the number of *traffic-optimal* multicast trees. To verify that there are an exponential number of traffic-optimal multicast trees, Lemma 14 and Theorem 5 are given as follows:

Lemma 14 *There are 2^m alternative paths for every (source, destination) pair of nodes in an ESMIN with m extra stages.*

Proof: This property has also been proven in [46] and a brief description is given as follows. Let the routing code, with destination routing, be $x_0x_1x_2\dots x_{h-1}$, where x_i is 0 or 1 in a 2×2 switch. To achieve a certain destination, we know all those later dimensions d_i must be either 0 or 1, which match to the corresponding destination bit i , $0 \leq i \leq \log_2 N - 1$. Hence, $\log_2 N$ bits are decided by the destination node but not the other m bits. Hence, there are 2^m possible routing codes to achieve a destination. In other words, there are 2^m alternative paths for any (source, destination) pair. □

Theorem 5 *There are at least 2^m optimal multicast trees (OMTs) for any source/destination set pair on ESMIN with m extra stages.*

Proof: From the Lemma 14, there are 2^m alternative paths to achieve the first destination node from the source. By the latest branch algorithm (step 2 to 4), we can construct one OMT from every one of these paths. Hence, there are at least 2^m OMTs for any source/destination set pair on ESMIN with m extra stages. \square

The exact number of OMTs is dependent on the destination set and the location of the latest stages with different dimensions. For example, let the second stage be the latest stage for dimension j , $1 \leq j \leq n-1$, and there be two destinations x and y . Furthermore, let x and y be distinguished in the j^{th} bit. To achieve both x and y , the path must branch at the second stage. After stage 2, there are $m-1$ extra stages or 2^{m-1} alternative paths to achieve both destinations. However, these two alternative path sets are independent of each other. Hence, there are $2^{2(m-1)}$ combinations after the second stage. The first stage also has two alternative paths to achieve the second stage. Hence, there are $2 \times 2^{2(m-1)}$, or 2^{2m-1} , OMTs.

An example of CU-ESMIN(8:5) is given in Figure 6.4 and 6.5. The source is 0 and destination set is $\{1, 3\}$. Figure 6.4 gives a CU-ESMIN(8:1,2,1,2,0). The number of OMTs is 2^m which is 4 as shown in the figure. In Figure 6.5, the dimension pattern is (1122). The destination 1 and 3 are distinguished at bit 1. The multicast tree splits at the second stage. After this stage, there is one extra stage which implies two alternative paths for every path. Hence, there are 4 combinations. Since there are

two alternative paths at the first stage, there are 8 ($=2^{2 \times 2 - 1}$) OMTs.

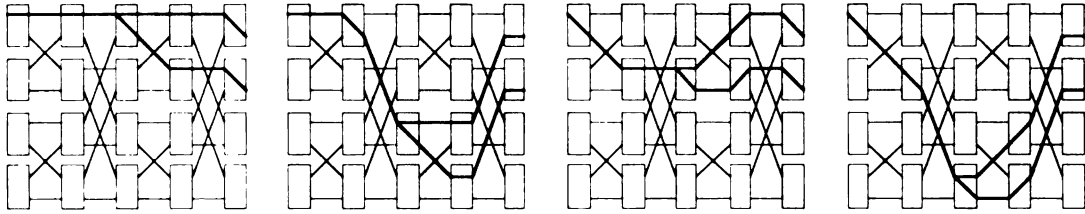


Figure 6.4: The four optimal multicast trees in an ESMIN(8:1,2,1,2,0)

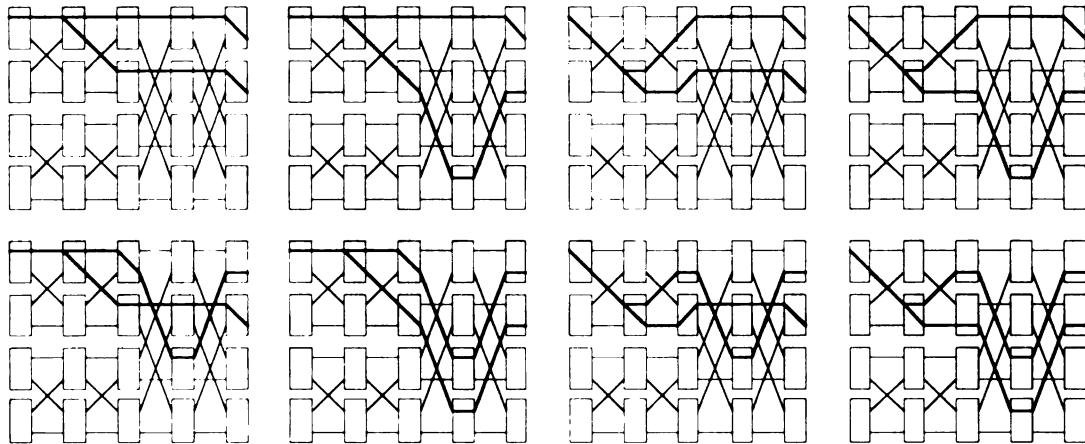


Figure 6.5: The eight optimal multicast trees in an ESMIN(8:1,1,2,2,0)

6.4 Multicast Algorithm

This section describes a multicast tree generation algorithm and shows that it can generate a traffic-optimum multicast tree in polynomial time. This algorithm applies generically to PS-ESMINs as well as to CU-ESMINs, and in our discussion we interchangeably use illustrations from both types of ESMINs. Note that the PS-ESMIN is merely a specific instance of the CU-ESMIN, and assertions for the latter imply the same for the former.

We also propose three other multicast heuristics that are non-optimal. These heuristics are compared with respect to the optimal algorithm in the simulation performance section.

6.4.1 Latest Branch Multicast Algorithm

The key point of this algorithm is to branch the message as late as possible. In a formal way, let us consider a CU-ESMIN($N : h$). The algorithm is specified as follows: when the destination id has a 0 (1) at dimension j , implement going up (down) at the latest (rightmost) dimensional j stage, where $0 \leq j \leq n - 1$. At all other stages, go straight to the next switch in the same row.

The latest branch multicast algorithm (LB) can be implemented in the following way:

Step 1: For each $c_i > 1$ ($0 \leq i \leq n - 1$) mark the non-latest stages concerning dimension i as ‘null’ stages. For every stage marked ‘null,’ eliminate the cross links and only keep the links that connect *within* a switch row.

Step 2: For every ‘null’ stage (also called ‘horizontal’) the message is simply forwarded by the *straight* link to the switch within the same row at the next stage. For every non-null stage, i.e., the latest stages (one per dimension), implement a going up if the corresponding destination bit is 0; otherwise, implement a going down.

Let the ESMIN be represented by enumerating its dimensions, and let a stage marked ‘null’ be labeled as ϕ . An example LB multicast in a CU-

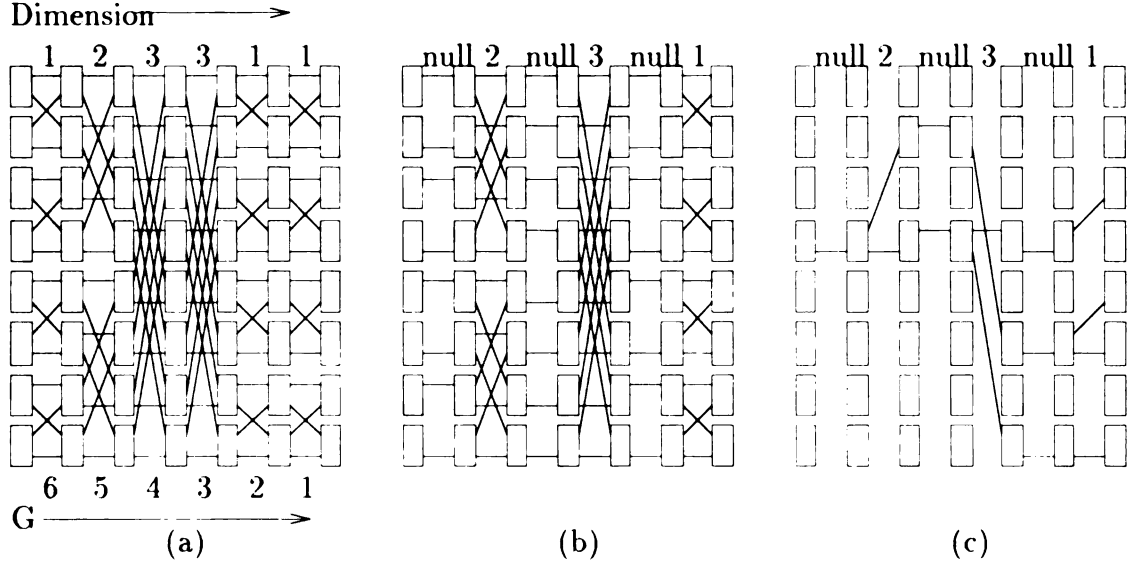


Figure 6.6: Latest-Branch multicast algorithm illustration: a) CU-ESMIN(16:1,2,3,3,1,1,0), b) Construction of the CU-ESMIN¹ by seeking the rightmost occurrence of each dimension and marking the remaining stages as ‘horizontal.’ Horizontal stages merely carry-forward the data. c) Example multicast in the CU-ESMIN¹.

ESMIN(16:1,2,3,3,1,1,0) is shown in Figure 6.6. The LB algorithm marks the non-latest positions for each dimension as ‘null’, i.e., converts the CU-ESMIN as $(null, 2, null, 3, null, 1, 0)$, as shown in Figure 6.6b. Then, the CU-ESMIN is merely a regular MIN (with a few extra ‘straight’ stages) in which multicast can be done trivially, e.g., Figure 6.6c.

Pseudo Code

A general LB algorithm to generate any one of those exponential optimal multicast trees is expressed as follows:

A restricted LB algorithm, to generate a unique multicast tree as discussed in the previous part of this section and to be referred to as LB in the following text, can be obtained by revising the first step to

Procedure *latest branch multicast*

1. Let $t_{0,j}$ be 0 or 1 at random, $0 \leq j \leq h-1$
/* This gives one of those 2^m possible paths for the source node to the first destination */
2. Let $t_{i,j} = t_{0,j}$, $1 \leq i \leq k-1$ and $0 \leq j \leq h-1$
3. Let $\{\ell_j = g \mid C_{g-1} = \beta_j \text{ and } C_i \neq \beta_j \text{ for } 0 \leq i < g-1\}$, $0 \leq j \leq n-1$
4. Let $t_{i,\ell_j} = d_{i,j}$, $0 \leq i \leq k-1$ and $0 \leq j \leq n-1$

Let $t_{0,j} = s$, $0 \leq j \leq h-1$.

6.4.2 Optimality Proof and Complexity

Theorem 6 *The latest branch algorithm generates the optimal traffic multicast tree.*

Proof: Let h be the number of total stages, $h = n + m$. The stages are numbers 1, 2, ..., h from left to right (source node to destination nodes), respectively. We prove the above theorem using induction.

Base case: $|D| = 1$ or 2: The MT constructed by the latest branch algorithm is optimal.

Inductive Hypothesis: Assume that $|D| = k-1$ is true and the optimal tree is $G_{OMT}(S, D', C')$.

Induction: $|D| = k$:

Let sw_1 be the latest branch switch that $G_{OMT}(S, D', C')$ can achieve the destination d_k and construct a MT $G_{MT}(S, D, C^*)$. Let sw_2 be any other branch switch by

which $G_{MT}(S, D', C')$ can reach d_k and use fewer channels. Name this multicast tree $G_{MT}(S, D, C'')$. Let sw_1 be at stage st_1 and sw_2 be at stage st_2 . Based on the latest branch algorithm, we have $st_1 \geq st_2$.

It is trivial that the path from st_2 to st_n has $n - st_2$ channels and the path from st_1 to st_n has $n - st_1$. Since $st_1 \geq st_2$, it is impossible that $n - st_1 > n - st_2$. This contradicts the assumption that $G_{MT}(S, D, C'')$ has fewer channels than $G_{MT}(S, D, C^*)$. Therefore, the $G_{MT}(S, D, C^*)$ is an optimal multicast tree. \square

Complexity: The first, second and last steps in the LB algorithm are repeated h , $h \times (k - 1)$ and $h \times k$ times, respectively. In other words, the complexity is $O(h \times k)$ for these three steps. To find the rightmost i^{th} dimensional stage, $0 \leq i \leq n - 1$, it takes at most h steps. Therefore, the complexity of the 3rd step is also $O(h \times k)$. Hence, the complexity of the LB algorithm is $O(h \times k)$, which is, indeed, a polynomial algorithm.

6.4.3 Other Sub-Optimal Multicast Heuristics

The LB multicast algorithm is traffic-optimal and its performance would be identical to the exhaustively generated optimum solutions. Hence, a question arises as to the basis of comparison for the proposed LB algorithm. Towards this, our idea is to compare the LB algorithm with what the current practices might be.

In the absence of our optimal multicast algorithm, and since there is no prior work on ESMIN multicast, current applications would use **either** a *random* dimension (RD)

selection approach **or** a *first-available* (i.e., greedy) approach to dimension selection. These lead to three heuristics, *First-Branch* (FB), *Random Branch* (RB) and *Random Mapping* (RM), which may not always generate traffic-optimal multicast trees. Their performance is compared the LB algorithm in the next section.

First-Branch

Instead of selecting the latest stage for each dimension, the FB approach selects the *earliest* stage for effecting each dimension. All non-earliest dimension stages are masked off and messages are simply forwarded to the next stage within the respective rows.

In other words, for each dimension i that has a 1 in the destination id, the FB approach would go up at the leftmost stage occurrence of dimension i . In all other non-leftmost occurrences of dimension i , the message would be forwarded straight. Alternately, if the destination node had a 0 bit at dimension i , then the FB approach would go down at the leftmost stage occurrences of dimension i . At all other non-leftmost occurrence of dimension i , the message would be forwarded straight. The tag generation mechanism for the FB approach is described below.

Procedure *first branch multicast*

1. Let $t_{i,j} = s$, $0 \leq i \leq k-1$ and $0 \leq j \leq h-1$
/* 's' indicates "go straight to next switch in the same row" */
2. Let $\{f_j = g \mid C_{g-1} = \beta_j \text{ and } C_i \neq \beta_j \text{ for } g < i \leq h-1\}$, $0 \leq j \leq n-1$
3. Let $t_{i,f_j} = d_{i,j}$, $0 \leq i \leq k-1$ and $0 \leq j \leq n-1$

Random Branch

This heuristic is a generalization of the LB algorithm. The LB algorithm enforces the latest stage, for each dimension, to account for going up (down) depending on whether the destination node had a 0 (1) bit. In addition, the LB algorithm also mandates going on the same path as the previous one, if any, at all the non-latest dimensional stages.

The RB approach generalizes the above second component of the LB approach. Thus, while it requires the latest stage to emulate the LB approach, the RB approach does not restrict the non-latest dimensional stages to ‘straight’ configurations. Each one of the non-latest switches can be in an up or down configuration and this configuration can be chosen randomly.

It can be shown that due to this random non-latest stage configuration, the RB approach can lead to blocking, i.e., unsuccessful multicast efforts. Figure 6.7 gives an example of a blocking situation. It is a CU-ESMIN(8:2,1,2,2,0) with source node 0 and destination $\{6, 7\}$. The path splits at G_4 and tries to compete with the others for the lower output port at G_1 . Since a switch has no ability to merge two messages, this multicast tree is blocked at the shadow switch.

Random Mapping

The *random mapping* (RM) approach can be viewed as a generalization of the FB and LB approaches. Each one of the FB or LB approaches implemented effects dimensional transfer at one and only one stage per dimension; so does the RM approach.

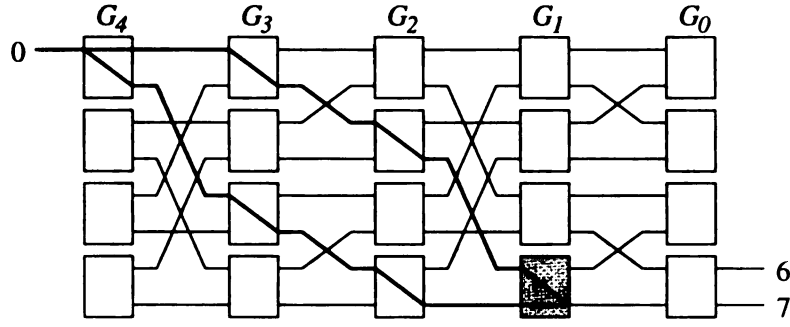


Figure 6.7: An example of blocking in multicast tree construction by the RB algorithm.

Procedure *random-branch multicast*

1. Let $t_{i,j} = x$, $0 \leq j \leq h-1, 0 \leq i \leq k-1$
/* 'x' indicates "go either up or down at random" */
2. Let $\{\delta_j = g \mid C_{g-1} = \beta_j \text{ and } C_i \neq \beta_j \text{ for } 0 \leq i < g-1\}$, $0 \leq j \leq n-1$
3. Let $t_{i,\delta_j} = d_{i,j}$, $0 \leq j \leq n-1, 0 \leq i \leq k-1$

All redundant stages for each dimension merely carry the message 'straight' within a switch row for FB, LB, as well as RM. The FB approach selects the dimensional-transfer stage at the *first* or leftmost stage for every dimension, while the LB approach selects the same at the *last* or rightmost stage for every dimension.

The RM approach does not make a deterministic selection decision either at the *first* stage or at the *last* stage. Instead it randomly selects any single stage per dimension. Thus, suppose that in a CU-ESMIN dimension 2 is involved at stages G_9 , G_6 , G_3 and G_1 . The LB approach would select the stage G_1 ; FB would select stage G_9 ; while RM can select *any* one of the four stages. The pseudo-code for the RM approach is given below. Note that two specific instances of the RM approach equate to the FB and LB approaches respectively.

Procedure *random mapping multicast*

1. Let $t_{i,j} = s$, $0 \leq i \leq h - 1$
/* 's' indicates "go straight to next switch in the same row" */
2. Let $G_j = \{g | C_{g-1} = \beta_j, 0 \leq g \leq h - 1 \text{ and } 0 \leq j \leq n - 1\}$
3. for each G_j , $0 \leq j \leq n - 1$, do
 - 3.1. Let r_j be the randomly chosen element in G_j
4. Let $t_{i,r_j} = d_{i,j}$, $0 \leq i \leq k - 1$ and $0 \leq j \leq n - 1$

6.5 Performance

This section presents the simulation performance of the proposed LB multicast algorithm. It is compared with the other three heuristic algorithms proposed in Section 6.4.3.

6.5.1 Simulation Description

This simulation considers a CU-ESMIN(64:10). The dimension assignment of each stage is varied following a number of patterns; in fact, one of these patterns "(5,4,3,2,1,5,4,3,2,1)" makes the CU-ESMIN equivalent to a PS-ESMIN - thereby also allowing us to effectively report performance of all algorithms over PS-ESMIN. The source node is node 0, without loss of generality. The number of destinations is varied between 2 and 64 in steps of 1 to capture the entire range of destination set size. Traffic is equated to the number of channels that carries a copy of the message. Both the LB algorithm and FB, RB, RM heuristics are operated with 500 randomly produced destination nodes *within each* destination set size.

6.5.2 Dimension Patterns and Output Parameters

Each one of the ten stages can be assigned to any one of the 5 butterfly dimensions (1 through 5), with the constraint that each dimension must be assigned at least once across the ten stages. Thus, the total number of different dimension patterns that can be generated is very large (somewhat less than 5^{10}) and cannot be exhaustively attempted. Instead, we selected six commonly occurring patterns, as listed in Table 6.1. Note that the first pattern configures the CU-ESMIN in a way that it is structurally-equivalent to a PS-ESMIN. Thus, the result generated for the CU-ESMIN with pattern 1 would also be applicable as the performance with respect to the PS-ESMIN.

Table 6.1: Patterns of this study

pattern	Butterfly pattern of each stage									
1	5	4	3	2	1	5	4	3	2	1
2	1	1	2	2	3	3	4	4	5	5
3	5	5	4	4	3	3	2	2	1	1
4	1	2	3	4	5	5	4	3	2	1
5	1	2	3	4	5	5	5	5	5	5
6	1	2	3	4	5	1	1	1	1	1

Output Parameters: We report the ‘traffic’ parameter (T) as the average number of channels used. T is used to report comparative performances between the optimum LB algorithm and two other heuristics: FB and RM. However, the third heuristic (RB) can block, i.e., lead to unsuccessful multicasts. Hence, we report the blocking rate of the RB approach only. Blocking rate, B , is defined as the probability that RB will be blocked while constructing a multicast tree or

$$B = 1 - \frac{\text{number of trials with success MT construction}}{\text{total number of trials}}$$

6.5.3 Plots and Observations

Plots for the T and B metrics are shown in the following. As mentioned before, the Pattern 1 also includes the case for the PS-ESMIN and thus the curves in the plots corresponding to “pattern 1” should be viewed separately to sense the performance for the PS-ESMIN.

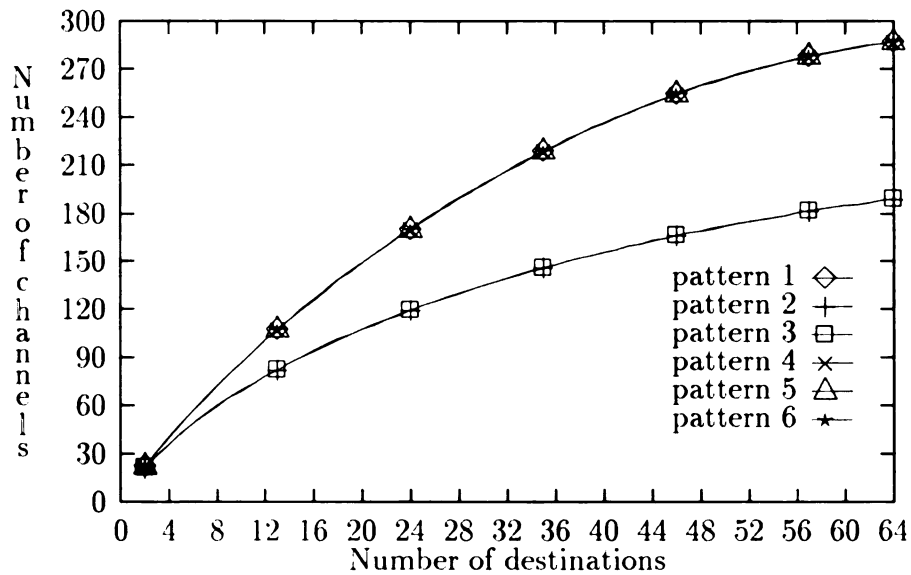


Figure 6.8: The number of channels used in the first branch algorithm.

Figures 6.8, 6.9 and 6.10 show the T values for the FB heuristic, the LB algorithm and the RM heuristic, respectively. For pictorial clarity, not all the x-axis points have been highlighted *on the respective curves using pattern identifiers*, though the data values from those points have been followed in the curves.

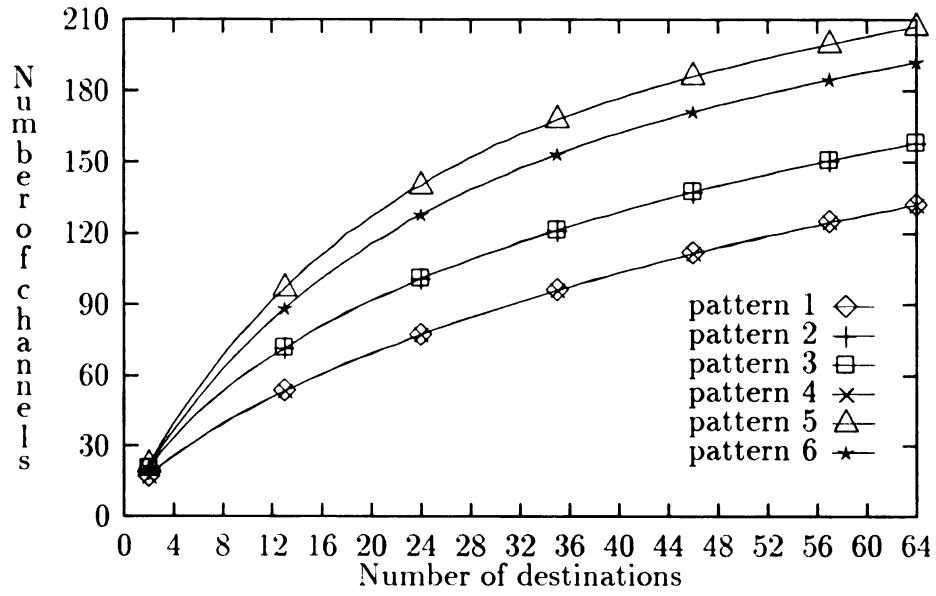


Figure 6.9: The number of channels used in the latest branch algorithm.

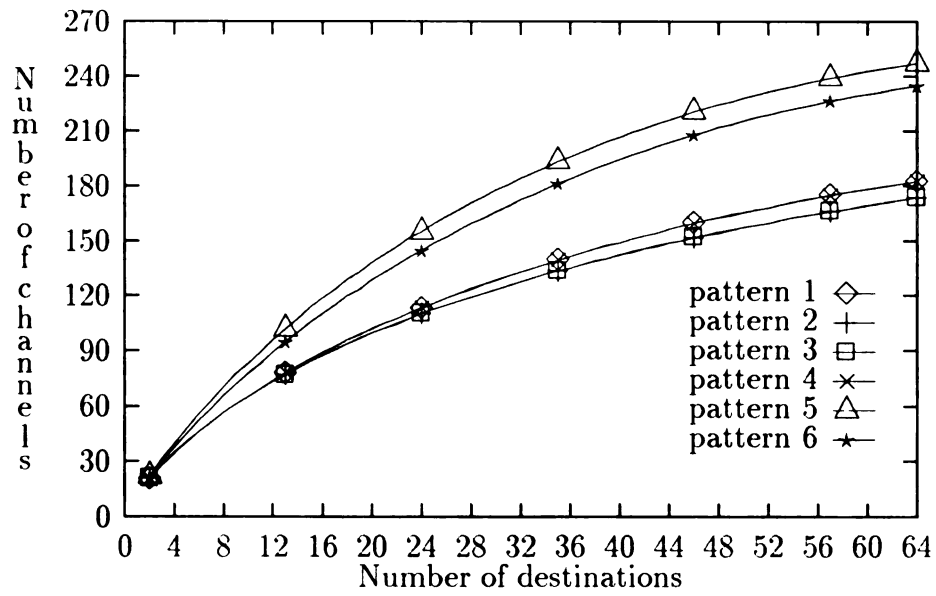


Figure 6.10: The number of channels used in the random mapping algorithm.

As shown in Figure 6.8, patterns 2 and 3 have similar performance. Patterns 1, 4, 5 and 6 also have closed performance on the FB heuristic. Patterns 2 and 3 have fewer channels used since the other patterns split the message in the first 5 stages. For the LB, patterns 1 and 4 as well as patterns 2 and 3 have the same performance because of the similarity of these patterns. For patterns 1 and 4, the LB only uses 6 channels in the first 5 stages — 5 to connect the stages and the other one to connect the source node to the first stage. At the rightmost 5 stages, pattern 1 is the same as pattern 4. Therefore, they have the same T metrics. The even distribution of destinations made patterns 2 and 3 have similar performance. However, this result does not consider the locality of processing nodes. Depending on the locality, pattern 3 and pattern 4 may have entirely different performances. In the RM, the performance of patterns (1,4) is like that of patterns (2,3). Pattern 5 and pattern 6 are slightly different due to the location of extra stages. In general, the later the message splits, the fewer the number of channels used.

Figure 6.11 shows the B metric for the RB heuristic. The blocking rate is over 20% when there are more than 10 destinations and 40% when there are 20 or more destinations for every pattern. This result suggests that the RB algorithm shall not be used even though the number of destinations is small. Otherwise, an intelligence switch, which is able to merge two messages with different routing tags but identical data, is needed to improve the blocking rate. The T metric for the RB heuristic is not reported since the RB heuristic does not always lead to successful multicast.

T and also B increase with an increasing number of destinations in all the cases, and this can be trivially expected. However, due to the channel saturation effect, the

rate of increase gradually decreases with an increasing number of destinations.

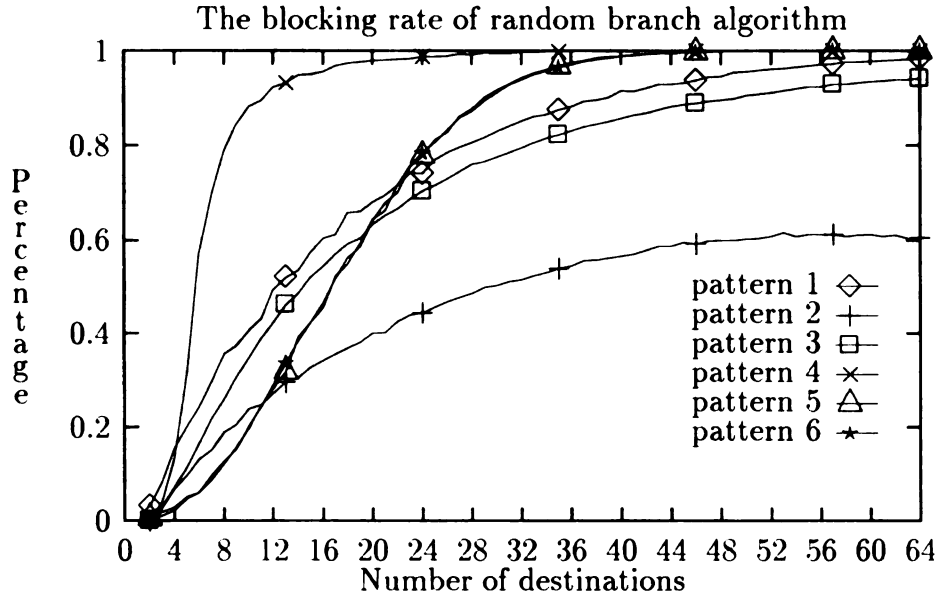


Figure 6.11: The probability of blocking in random branch algorithm.

Figure 6.12 compares the T metric for the optimal LB algorithm and FB, RM heuristics. It is easy to observe that the LB leads to the least traffic; RM leads to the next least; and FB gives the highest traffic, which is due to the fact that LB is optimum and RM is an intermediate generalization between LB and FB. Not all patterns have been reported in Figure 6.12, since some of the patterns lead to overlapping results. At this moment we are not reporting inter-pattern traffic variation analysis; however, we believe that the traffic generated has strong correlation with the dimension pattern.

6.6 Summary

In this chapter, we discussed the establishment of a multicast tree in the ESMIN, which is an extension design of regular MINs. The ESMIN provides several advantages, such as a flexible routing path, fault tolerance, *etc.* The ESMIN multicast

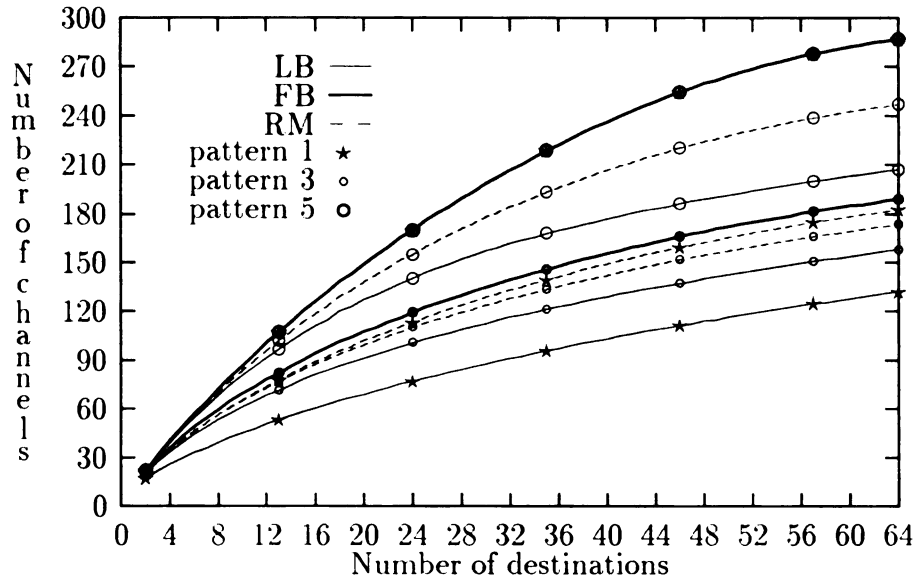


Figure 6.12: The number of channels used in three non-blocking algorithms.

problem is formulated and an optimality criterion is defined. An upper bound on the number of multicast trees is estimated, and we have shown that the total number of traffic-optimal multicast trees may itself be exponential. However, a traffic-optimal multicast tree can be generated in polynomial time. The proposed latest branch multicast algorithm can achieve this.

When multiple multicast communications occur simultaneously, deadlock becomes possible as shown in Chapter 4. Hence, some systems rely on unicast-based multicast, or software-based multicast. How to support an efficient software-based multicast on such systems will be discussed in the next chapter.

CHAPTER 7

Software-based Multicast

Most existing message-based SPCs support only unicast communication, that is, single-destination message passing, in hardware. In this environment, multicast must be implemented in software by sending unicast messages. One way to implement multicast in such systems is *separate addressing* in which a separate copy of the message is sent from the source to every destination. As the number of destinations increases, this separate addressing may require excessive time. An alternative approach is to use *multicast tree* in which the source sends the message to only a subset of the destinations. Each recipient of the message forwards it to some subset of the destinations that have not yet received it. The focus of this chapter is on such multicast tree implementation, also known as *unicast-based* multicast implementation.

7.1 Issues in Multicast Communication

Although hardware implementations of multicast communication would intuitively offer better performance than software implementations, many such implementations

exhibit either undesirable properties or are restricted in their use. The NEC Cenju-3 claims to provide restricted multicast in hardware with the limitation that all destination addresses must be consecutive. This restricted multicast is called *single region broadcast*. Unfortunately, single region broadcast is not deadlock free unless only one does multicast at a time. A deadlock example is illustrated in Figure 7.1, where a shadow switch indicates that a message is blocked by the other in that switch. In the initial instance, given in Figure 7.1(a), nodes 0 and 5 initiate multicasts while two unicasts, node 1 to node 5 and node 4 to node 2, are transmitted in the network. Hence, both source nodes grab partial destinations. After both unicasts are completed, as shown in Figure 7.1(b), each source node grabs half the number of nodes and waits for the other half to become available, deadlock occurs.

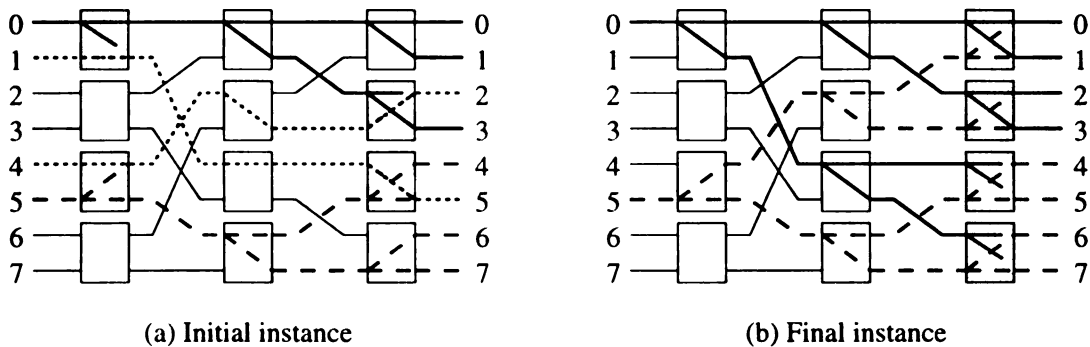


Figure 7.1: An example of deadlock in single region broadcast.

Although some deadlock-free multicast algorithms were proposed [25, 47, 48, 49], most existing wormhole-switched SPCs support only unicast communication in hardware. In these environments, all communication operations must be implemented in software by sending one or more unicast messages. One main issue is to develop an efficient multicast tree. Which type of multicast trees to use depends on the underlying switching strategy and unicast routing algorithm. An efficient multicast tree

should involve no local processors other than the source and destination processors, should exploit the distance-insensitivity of wormhole switching, and should maintain a minimum height, specifically, height $k = \lceil \log_2(m) \rceil$ for $m - 1$ destination nodes. Another desirable property is that there be no channel contention among the constituent messages of the multicast. That is, the unicast messages involved should not simultaneously require the same channel.

How to achieve these goals depends on the network topology, switching strategy, and unicast routing algorithm of the SPC. The issues and difficulties involved in implementing efficient multicast communication in unidirectional MINs is illustrated in the following (small-scale) example. Let's consider an 8-node multistage cube network built with 2×2 switches. Suppose a multicast message is sent from source 100 to all other nodes, $\{000, 001, 010, 011, 101, 110, 111\}$. Figure 7.2(a) shows a binary multicast tree. At step 1, the source sends the message to node 000. At step 2, nodes 100 and 000 inform nodes 110 and 010, respectively. Continuing in this fashion, this implementation requires 4 steps to reach all destinations. Taking advantage of the distance insensitivity of wormhole routing, the duration of each step must be approximately equal to the duration of a single unicast transmission of the message. In other words, it should be correct to assume that each step requires unit time as long as there exists no channel contention among the messages transmitted during each step. For this reason, the multicast latency in Figure 7.2(a) is 4 time steps. In Figure 7.2(b), the shape of the tree is rearranged in such a way that the number of steps to complete the tree can be reduced to 3. However, closer inspection reveals that the message sent from node 100 to node 001 and the message sent from node 000

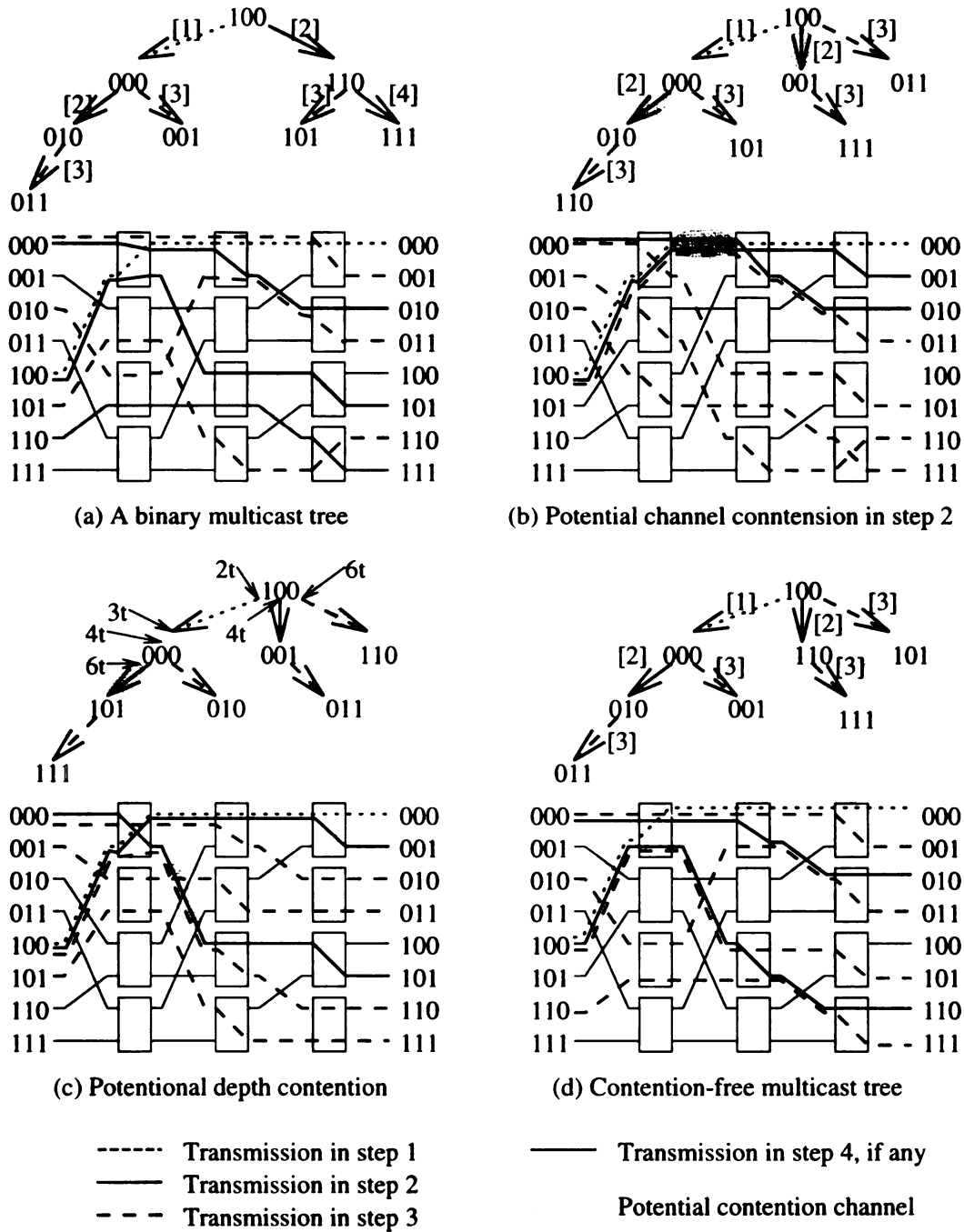


Figure 7.2: Unicast-based software multicast trees

to node 010 in step 2 use two common channels. The contention for those channels would force one message to block the other from using the channel. Consequently, these two unicasts cannot take place during a single time step. As a result, the multicast latency in Figure 7.2(b) is actually larger than 3 time steps.

This situation is rectified in Figure 7.2(c), where the messages sent within a particular time step do not contend for common channels. Contention among messages sent in *different* steps may still arise, however, if the message length is small and the sending latency is large. For ease of explanation, we model the sending latency $2t$ and the receiving latency t . The length of the multicast message is chosen in a way that its network latency is t . As shown in Figure 7.2(a), the message transmission from node 100 to node 110 and the message transmission from node 000 to node 010 take place concurrently during the time period between $6t$ and $7t$, and contention occurs for the shadowed areas. The multicast tree in Figure 7.2(d), which is based on the methods presented in next section, is contention-free regardless of message length or startup latency. In the next section, we will show that such contention-free multicast may not always exist.

7.2 Multicast Algorithm

This section defines an optimal multicast as well as an algorithm to implement it in certain network topologies. We also give a proof of some other network topologies whose optimal multicast tree may not exist even if the source and destination nodes form a cube.

The theoretical model for unicast-based multicast communication has been addressed in [50]. The underlying topology of the network is represented by a directed graph, $G(V, E)$ with the vertex set $V(G)$ and the arc set $E(G)$. A vertex u in $V(G)$ represents a switch box or a processor node. An arc (u, v) in $E(G)$ represents the unidirectional channel from switch box u to switch box v , processor node u to switch box v , or switch box u to processor node v . An alternating sequence $ue_1v_1e_2\ldots v_{k-1}e_kv$ of distinct vertices and distinct arcs, starting with vertex u and ending with vertex v is called a *directed path*, or *path* for short.

A unicast operation can be defined as an ordered quadruple $(u, v, P(u, v), t)$, where u and v are the source and destination vertices respectively, $P(u, v)$ is a given path in G over which the message will traverse, and t is the step at which the unicast is to take place. Each unicast is assumed to take a unit of time whose duration is independent of the path length. As a result, a unicast that begins with time step t should terminate at time step $t+1$, provided that no encountering channel is blocked by other messages. This assumption is consistent with the wormhole switching strategy which diminishes the effect of the path length in communication latency.

In a one-port architecture, in which a processor node may send (receive) only one message at a time, two unicasts $(u, v, P(u, v), t)$ and $(x, y, P(x, y), \tau)$, with $t = \tau$, are called *feasible* if vertices u , v , x , and y are all distinct processors. A set of unicasts $U_t = \{(u_1, v_1, P(u_1, v_1), t), (u_2, v_2, P(u_2, v_2), t), \dots, (u_k, v_k, P(u_k, v_k), t)\}$, whose members are pairwise feasible, is called a *feasible unicast set*. A multicast group can be represented by a set $M = \{d_0, d_1, d_2, \dots, d_{m-1}\}$, where vertex d_0 represents the source and vertices d_1, d_2, \dots, d_{m-1} represent the destinations. An *implementation*

$I(M)$ of a unicast-based multicast request M is a sequence of feasible unicast sets U_1, U_2, \dots, U_k satisfying the following conditions:

1. For each j , $1 \leq j \leq k$, if $(u, v, P(u, v), j) \in U_j$ then both u and v belong to M .
2. The set $U_1 = \{(d_0, u, P(d_0, u), 1)\}$, where $u = d_i$ for some i , $1 \leq i \leq m - 1$.
3. For every unicast $(u, v, P(u, v), t) \in U_t$, $1 < t \leq k$, there must be a set U_τ with $\tau < t$ which has $(w, u, P(w, u), \tau)$ as a member.
4. For every destination d_i , $1 \leq i \leq m - 1$, there exists one and only one integer t such that $1 \leq t \leq k$ and $(w, d_i, P(w, d_i), t)$ appears in U_t for some vertex w .

The first condition guarantees that only the destination processors of the given message are involved in the implementation. The second condition states that the first step of the implementation involves a single unicast from the source to one of the destinations. The third condition ensures that a destination processor has received the message before it may forward it to another destination processor. Finally, the fourth condition guarantees that every destination processor receives the message exactly once.

Condition 3 above also implies that the total number of destinations receiving the message can at most double during each step. Therefore, $\lceil \log_2 m \rceil$ is the greatest lower bound of the number of steps required by an implementation. An implementation requiring exactly $\lceil \log_2 m \rceil$ steps is referred to as a *minimum-step* implementation.

Definition 10 Two feasible unicast operations $(u, v, P(u, v), t)$ and $(x, y, P(x, y), t)$ are called stepwise contention-free if $P(u, v)$ and $P(x, y)$ are arc-disjoint. An imple-

mentation is called *stepwise contention-free* if the elements in each unicast set U_i are pairwise stepwise contention-free.

Stepwise contention-freeness guarantees that the minimum multicast latency can be achieved by a minimum-step implementation when the message size is large and startup latency is neglected.

Definition 11 *A vertex v is in the reachable set R_u of a vertex u if and only if there exists a t , $1 \leq t \leq k$, such that either $\{(u, v, P(u, v), t)\} \in U_t$ or $\{(w, v, P(w, v), t)\} \in U_t$ for some node $w \in R_u$.*

The reachable set of a vertex u contains those vertices in M to which the message is sent after having been handled by vertex u . If the implementation $I(M)$ is viewed as a tree of unicast messages, then the reachable set of a vertex u is the set of vertices in the subtree rooted at u . For example, in Figure 7.2(b), $R_{000} = \{010, 101, 110\}$. Using this definition, the characteristics of an implementation necessary to avoid contention between messages sent in different steps, called *depth contention*, can be formally defined as follows.

Definition 12 *An implementation $I(M)$ is depth contention-free if and only if $P(u, v)$ and $P(x, y)$ are arc-disjoint for any two unicasts $(u, v, P(u, v), t)$ and $(x, y, P(x, y), \tau)$ in $I(M)$ such that $u \neq x$, $u \notin R_y$, and $x \notin R_v$.*

Depth contention includes all possible types of channel contention among concurrently transmitted messages without consideration of message size or startup latency. Since each processor is characterized by one-port communication architecture, node

u has to send messages to any two nodes, v and y , sequentially. Thus, the message transmitted from u to v and the message transmitted from u to y will never contend for a common channel. In addition, no node in R_u can begin to send or receive a copy of the message before u completes receiving its copy of the message. This implies that the message transmitted from u to v and the message transmitted from x to y will never contend for a common channel if $x \in R_v$ or $u \in R_y$. An implementation must be stepwise contention-free if it is depth contention-free.

Depth contention-freeness guarantees that the minimum multicast latency can be achieved by a minimum-step implementation when the message size is small and the unicast communication latency is dominated by startup latency. An implementation that attains the minimum multicast latency is said to execute in *minimum-time*. Such an implementation is also called the *optimal multicast* in what follows.

7.3 Non-Optimal Multicast in Baseline and Butterfly Networks

This section shows that an optimal multicast may not exist for either baseline nor butterfly networks. The following lemmas prove this.

Lemma 15 *In an N -node baseline network, where $N = k^n$ and $n \geq 3$ if $k \geq 4$ or $n \geq 5$ if $k = 2$, an optimal unicast-based multicast may not exist when source and destination nodes form a cube.*

Proof: We shall disprove that there exists a stepwise contention-free multicast. As shown in Lemma 10, the channel used in connection C_i is $d_{n-2} \cdots d_{i+1} s_{n-2} \cdots s_{n-i-1} d_i$. Therefore, this channel is used by those source nodes, $x_{n-1} s_{n-2} \cdots s_{n-i-1} x_{n-i-2} \cdots x_0$, and those destination nodes, $d_{n-1} \cdots d_i y_{i-1} \cdots y_0$, where x_i and y_i are variables within the range from 0 to $k-1$. Since there are at least four nodes in a cube to have two node-disjoint unicast operations, $n-i-1 \geq 2$ and $i \geq 2$ for $k=2$ or $n-i-1 > 0$ and $i > 0$ for $k \geq 4$. By solving these inequality, we get $n \geq 5$ when $k=2$ or $n \geq 3$ while $k \geq 4$.

Let $d_j = s_j$ for $\ell \leq j \leq n-1$, where ℓ is the minimum value of $\{i, n-i-1\}$. These nodes form a cube, C , with addressing code $d_{n-1} \cdots d_\ell x_{\ell-1} \cdots x_0$. This implies that $P(s_1, d_1)$ and $P(s_2, d_2)$ are not arc-disjoint for $\{s_1, d_1, s_2, d_2\} \in C$. Thus, the unicast operations $(s_1, d_1, P(s_1, d_1), t)$ and $(s_2, d_2, P(s_2, d_2), t)$ are not feasible. Therefore, the minimum-time unicast-based multicast may not exist in the baseline network. \square

For example, let the cube be 000XX in a 32-node baseline network constructed by 2×2 switches. As shown in Lemma 10, channels 00XX0 are used in C_4 , 000X0 are used in C_3 , 00000 is used in C_2 , 0000X are used in C_1 , and 000XX are used in C_0 . In the connection C_2 , all nodes share the same channel to reach any other one. This implies that any two node-disjoint unicast operations are not feasible, i.e., stepwise contention-free, in this cube.

Lemma 16 *In an N -node butterfly network, the optimal unicast-based multicast may not exist when source and destination nodes form a cube.*

The proof of this lemma is similar to that of Lemma 15. Since there is no optimal multicast in both baseline and butterfly networks, we concentrate our efforts in cube and omega networks. However, the algorithm proposed below may apply to modified baseline and butterfly networks directly since they have the same network partitionability as cube and omega networks.

7.4 The C-min Algorithm

The C-min, (Corresponding MIN), algorithm is proposed not only to achieve the optimal unicast-based multicast but also to reduce the probability of contention among different unicast/multicast communications. The basic idea of the C-min algorithm is to divide the lexicography-ordered chain [51] into two even chains, the upper chain and the lower chain. Then, the source node delivers the message to the *corresponding* position of the other half. Such a process is recursively executed until every node receives a copy of the message. The C-min algorithm is formally specified in Figure 7.3.

In Figure 7.3, the C-min algorithm first sorts source and destination addresses in lexicographic order, known as *lexicography-ordered chain* and is denoted by Φ . The source successively divides Φ in half. If a source node is in the lower or upper half, then it sends a copy of the message to the *corresponding node* in the upper or lower half, respectively. A corresponding node is the node which has same position as source node when Φ is divided into two halves. That destination will serve as source for the upper (lower) half if it is located in the upper (lower) half, using the C-min

Algorithm: C-min Algorithm

Inputs: Φ : lexicography-ordered chain $\{d_\ell, d_{\ell+1}, \dots, d_r\}$ for source and destination addresses

d_s : the address of source nodes

Procedure:

while $\ell < r$ **do**

$c = \frac{\ell+r+1}{2}$;

if $s < c$ **then** /* send to upper cube */

$D = \{d_c, d_{c+1}, \dots, d_r\}$;

$t = s + \frac{r-\ell+1}{2}$;

$r = c - 1$;

else /* send to lower cube */

$D = \{d_\ell, d_{\ell+1}, \dots, d_{c-1}\}$;

$t = s - \frac{r-\ell+1}{2}$;

$\ell = c$;

endif

Send a message to node d_t with the address field D ;

endwhile

Figure 7.3: The C-min algorithm

algorithm. A source node continues this procedure until Φ contains only its own address. Such a divide-and-conquer property makes the C-min algorithm successful in attaining the minimum-step multicast implementation.

Figure 7.4 shows how to obtain the optimal multicast implementation (Figure 7.2) by using the C-min algorithm. Source, node 100, begins with a lexicography-ordered chain $\Phi = \{000, 001, 010, 011, 100, 101, 110, 111\}$. As shown in Figure 7.4, the source node first sends a copy of the message to node 000, the node with the corresponding position in the lower half of Φ . The lower half of Φ is deleted, and therefore the nodes remaining in Φ are $\{100, 101, 110, 111\}$. Since source node 100 is the first node in the new Φ , it sends a copy to the first node, node 110, in the new upper half. Eventually, source 100 sends a copy of message to node 101. Each of the receiving

nodes is likewise responsible for delivering the message to the nodes in its subtree using the same algorithm. As shown in this figure, this multicast implementation requires 3 steps. Note that the associated diagram of such an implementation on a multistage cube network is shown in Figure 7.2(d).

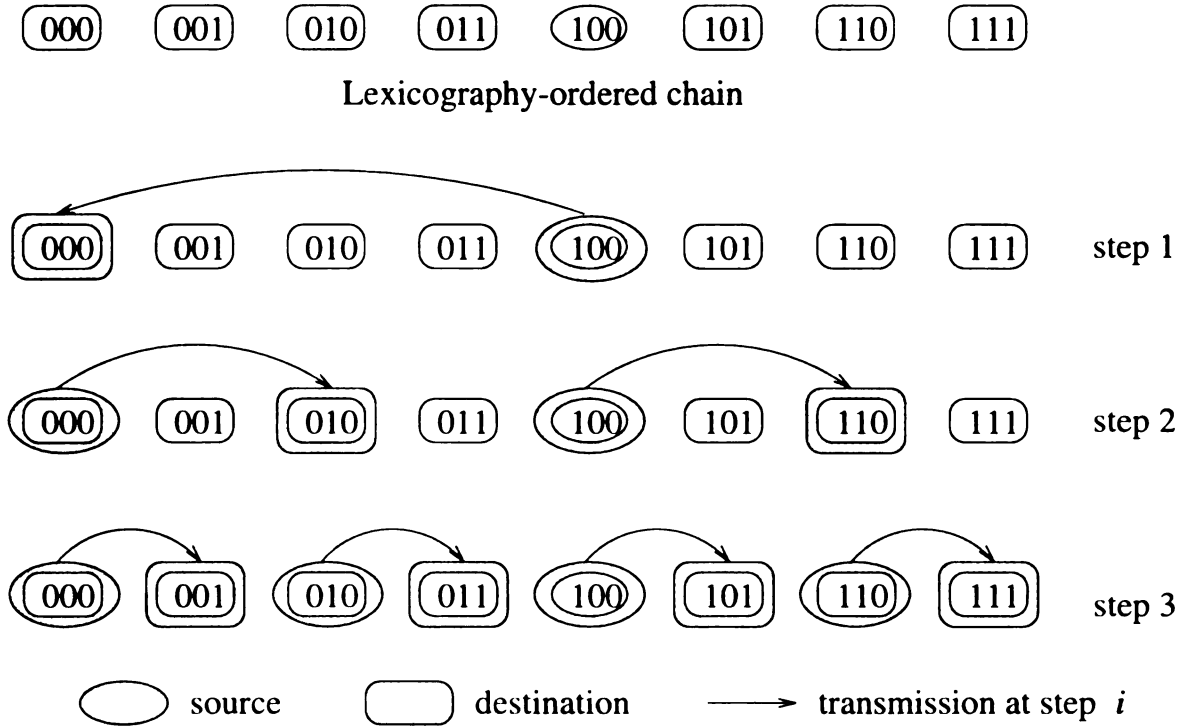


Figure 7.4: An example of software multicast tree implemented by C-min algorithm.

To show that the C-min algorithm can achieve the minimum-time multicast as well as minimum contention possibility, we will show that all transmissions are located in disjoint *s-cube*'s, which is formally defined as follows:

Definition 13 An *s-cube* is the smallest k -ary cube in a k -ary m -cube.

An *s-cube*, which is a k -ary 1-cube, contains only k nodes, in which addresses are varied in the same single position. From Lemmas 8 and 9, which imply that two disjoint *s-cube* won't have any communication interference, we can obtain the

following Lemma.

Lemma 17 *Communications in disjoint s-cubes are contention-free in cube and omega networks.*

Another important property of an s-cube is that the communications within an s-cube are contention-free as long as those communications are node-disjoint. When $k = 2$, an s-cube contains only 2 nodes which implies that there is at most one communication. Hence, the communication is stepwise and depth contention-free. In the following, we will consider the case where $k \geq 4$ and k is power of 2.

Lemma 18 *Node-disjoint communications within an s-cube are contention-free in cube and omega networks.*

Proof:

Let's assume that there is a contention between two node-disjoint unicast operations, $(s_1, d_1, P(s_1, d_1), t)$ and $(s_2, d_2, P(s_2, d_2), t)$, where $\{s_1, s_2, d_1, d_2\}$ belongs to an s-cube with address code $a_{n-1} \cdots a_{j+1} x a_{j-1} \cdots a_0$. Based on destination routing, the channels $a_{n-1} \cdots a_x \cdots a_{i+1} a_{i-1} \cdots a_0 a_i$, $a_{n-1} \cdots a_{i+1} a_{i-1} \cdots x \cdots a_0 a_i$ and $a_{n-1} \cdots a_{i+1} a_{i-1} \cdots a_0 x$ are used by the unicast operation $(s, d, P(s, d), t)$ while $j > i$, $j < i$ and $j = i$, respectively. Since the x exists during the address change, sharing a common channel between two unicast operations implies that either $x_{s_1} = x_{s_2}$ or $x_{d_1} = x_{d_2}$, a contradiction. \square

The proof for omega networks is similar and is omitted. Note that as long as the source nodes are distinct and destination nodes are distinct, the unicast transmissions

within an s-cube are stepwise contention-free. To show that the C-min algorithm can achieve optimal multicast, we need to show that multicast in an s-cube is depth contention-free.

Lemma 19 *The multicast tree generated by C-min algorithm in an s-cube is depth contention-free.*

Proof: Let (s_1, d_1) and (s_2, d_2) be two source destination pairs in different steps and $s_1 \neq s_2$. Let's assume that there are common channels used in these two source destination pairs. From the C-min algorithm, we have $d_1 \neq d_2$. From Lemma 18, s_1 must equal to s_2 in order to share a common channel, a contradiction. \square

Since the C-min algorithm divides the k -ary m -cube into k disjoint k -ary $(m-1)$ -cube in every $\log_2 k$ steps, we have the following theorem from Lemmas 17, 18, and 19.

Theorem 7 *The implementation constituting a C-min tree is stepwise contention-free, depth contention-free, and a minimum-time implementation.*

7.5 Performance Evaluation

Our performance evaluation is based on a simulator which simulates different unidirectional MINs, including cube, baseline (which is used in the NEC Cenju-3), butterfly, and omega networks. The message length is uniformly distributed between 32 and 96 flits. The number of nodes is 64, and different size of switches constituting the

network is considered in the simulation. Latency is reported in the 95% confidence interval. Multiple multicasts, where each multicast is in a separate cube, are allowed simultaneously since the unicast-based multicast is deadlock-free. Based on previous discussions, it is easy to observe that the cube and omega networks should have identical performance. Our simulation results did confirm this observation; therefore, only cube networks are considered in the following context.

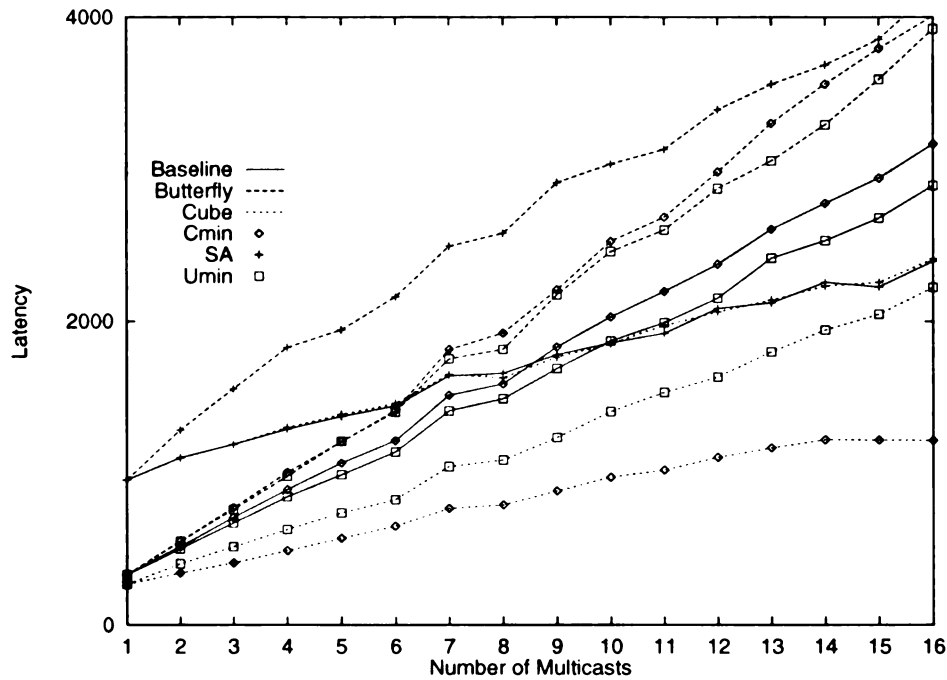


Figure 7.5: The latency in blocking multicasts.

Figure 7.5 gives the latency when multiple-blocking multicast communications are initiated simultaneously. Every cluster contains 16 nodes but only a limited number of nodes can initiate multicast communications. By carefully examining the networks, the number of channels is reduced to 4 from 16 in connection C_1 of the butterfly network. This implies that latency on the butterfly will be the worse than the other two networks. Such phenomena is confirmed by Figure 7.5. Although the number of

channels is 16 between any two stages on the baseline network, there are potential channel contentions when node-disjoint communications are conducted within an s -cube. For example, sources 0 and 2 compete channel c_0 at connection C_1 when they are forwarding messages to destinations 1 and 3, respectively. Since such unicast patterns are the base for C-min algorithms, the performance of C-min on a baseline network should be degraded.

For a cube network, the C-min can fully utilize the network. Hence, it has the shortest latency in a cube network as shown in Figure 7.5. Unlike cube network, the U-min has a smaller latency than the C-min algorithm on the baseline and butterfly networks due to batch-like message forwarding in those middle nodes. The performance of separate addressing on the cube and baseline networks are similar since the separate addressing can fully use all 16 channels.

Figure 7.6 shows the latency on various networks for non-blocking multicast communications with 16 nodes in each cluster. The system load is defined as the average number of flits expected on an output port per time unit. Such a definition is identical to the definition of load in Chapters 4 and 5.

Unlike network independent of hardware multicast, it's easy to observe from Figure 7.6 that the network topology is a very important factor. The cube network always has the lowest latency in all traffic loads comparing with the two other network topologies because of network partitionability and traffic localization. The size of switches is another important factor. Larger switches provide lower latency in cube and baseline networks but not in butterfly networks. This contrary result in butterfly is due to the identity connection in the leftmost stage. Note that 64-node cube and

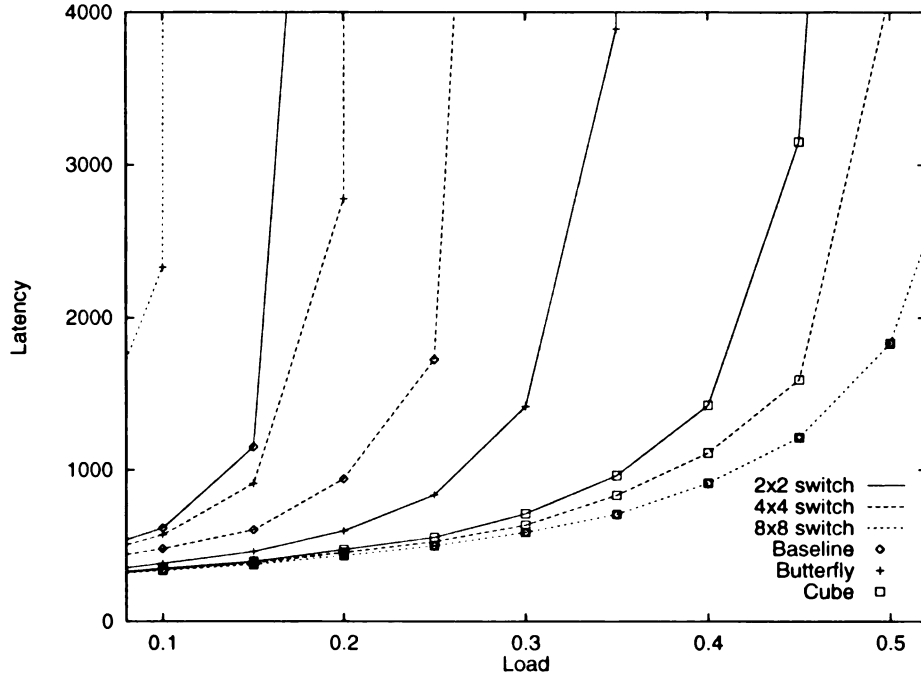


Figure 7.6: The latency of C-min algorithm on various networks and various sized switches.

baseline networks with 8×8 switches are identical.

The comparison among three different unicast-based multicast algorithms: C-min, U-min, and separate addressing (SA), is given in Figure 7.7. Three different processor clusters are considered: 16 clusters with 4 nodes per cluster, 4 clusters with 16 nodes per cluster, and a 64-node cluster. Note that the U-min algorithm was designed for bidirectional MINs [51], not for unidirectional MINs. The performances given here are only for comparison purpose. In a light load environment, both C-min and U-min algorithms provide a lower latency. Since the U-min always sends messages to middle nodes first, the queue in these nodes become saturated quickly. The latency shows such an effect even when the number of nodes in a cluster is small. The separate addressing is similar to a batch process. Therefore, it should have a higher throughput in a heavily loaded environment which also implies a lower latency. Such phenomena

is confirmed in Figure 7.7.

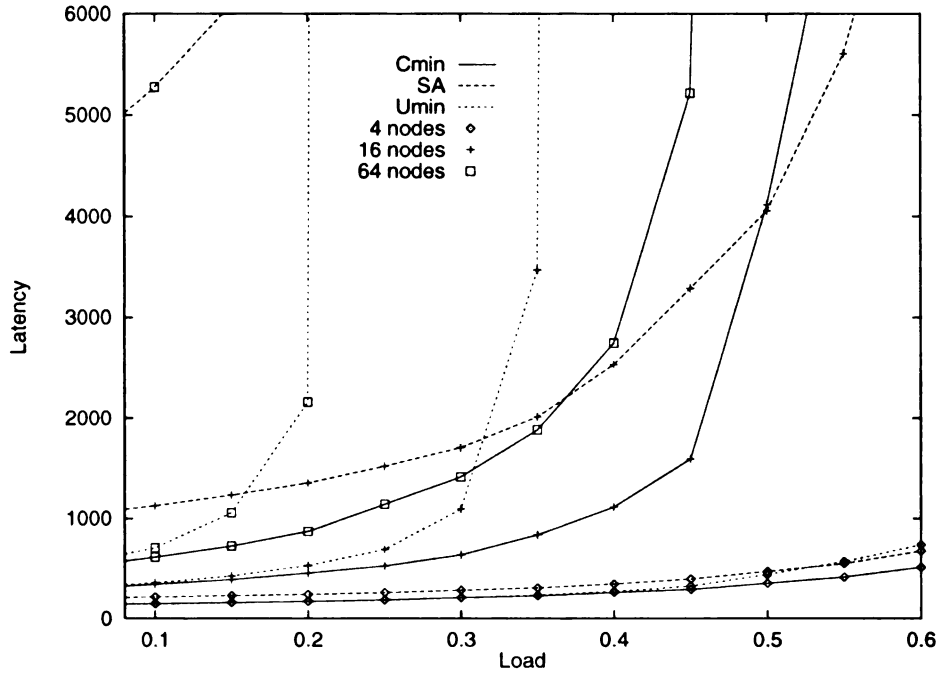


Figure 7.7: The latency of various algorithms on a cube network constructed by 4×4 switches.

Figure 7.8 gives the throughput of different networks with different multicast algorithms. The results also verify previously mentioned phenomena. Due to saturation in the U-min algorithm, its throughput decreases rapidly as the number of nodes in a cluster increases. The peak throughput of the C-min is better than separate addressing during certain ranges since the C-min evenly distributes its traffic.

7.6 Summary

In this chapter, our original intention was to propose an optimal unicast-based multicast algorithm for unidirectional MINs. The first finding in this work is that known topologically equivalent delta-class networks have quite different capabilities in sup-

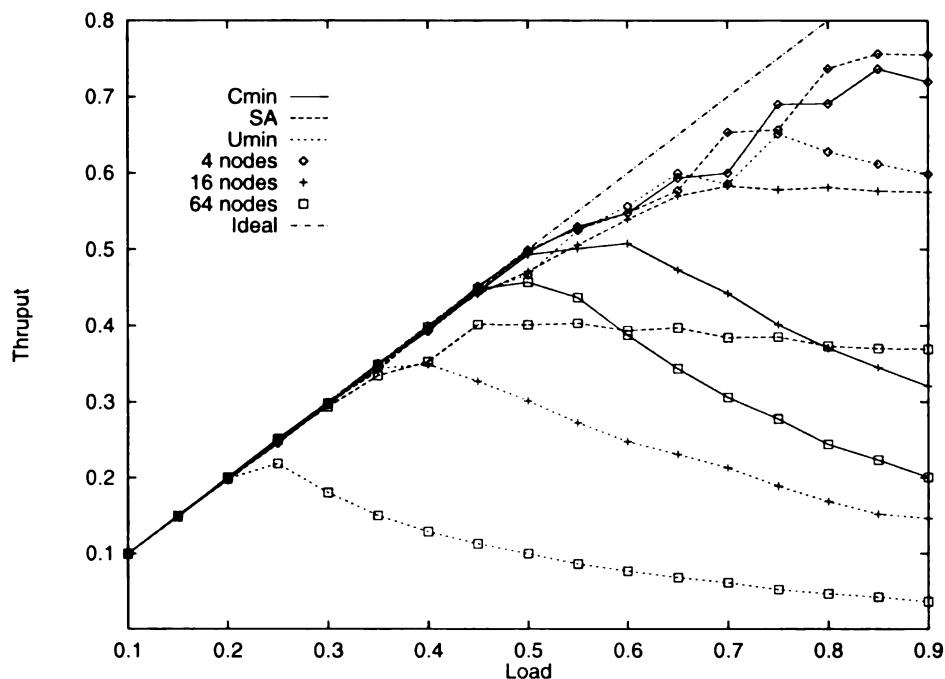


Figure 7.8: The throughput of various algorithms on cube networks with 4×4 switches.

porting multicast. We have shown that cube and omega networks can always support optimal software multicast, while baseline and butterfly networks may not. This suggests that either the cube or the omega network is a better choice for constructing unidirectional MINs for SPC's among other MINs. The proposed C-min algorithm provides a minimum unicast-based multicast for such networks even when there are multiple multicasts. Although the modified baseline and butterfly networks are not discussed in this chapter, the C-min algorithm may apply to both networks directly.

CHAPTER 8

Related Work

As discussed earlier, multicast communication is a frequently used communication pattern. Some low-level languages based on message-passing interfaces or libraries, such as MPI and PVM, explicitly define a number of collective communications in which the performance of many of them can benefit from efficient multicast communication [1, 52]. In those high-level languages, such as HPF, the multicast communication pattern is implicitly specified, such as aligning a low-dimensional array to a high-dimensional array [8]. Multicast communication is also demanded in high-speed networks, such as various ATM switches and the DEC GIGAswitch, for various applications [3, 4, 5]. Multicasting has been studied for distributed systems, such as reliable multicast (*e.g.*, [53]) and ordered multicast (*e.g.*, [54]), and for wide area networks, such as MBONE [55] and Internet multicast [56, 57].

Multicast communication has been extensively studied for distributed-memory multicomputers based on direct network architectures, such as mesh and hypercube topologies. Hardware multicast support for hypercube machines based on virtual cut-

through switching was proposed in [58, 22], while a prototype VLSI multicast router was designed in [26]. Performance evaluation of various wormhole multicast routing algorithms was reported in [59]. Since then, many researchers have contributed to this important area (*e.g.*, [47, 60, 48, 61, 49, 62]). A theoretical study of multicast communication for 2D mesh and hypercube topologies can be found in [63]. Some parallel computer vendors have also tried to directly support multicast communication in their machines. The nCUBE-2 was the first hypercube machine supporting a restricted multicast, in which each multicast is actually a broadcast with a subcube. However, as pointed out in [59], deadlock is possible in the nCUBE-2 when there are two or more simultaneously multicasts within a subcube. Consequently, nCUBE decided to disable their hardware multicast support and replaced it with software multicast. The Cray T3D has dedicated hardware to support a special case of multicast, namely barrier synchronization [42]. As pointed out in [64], barrier synchronization can be efficiently implemented by utilizing the underlying multicast support.

Multicast communication for MIN-based parallel architecture was studied in [32]. However, their multicast support is restricted to a multistage cube network and to destination nodes forming a subcube. The TMC CM-5 has a dedicated control network to support multicast communication [65]. The multicast is restricted to one message at a time and must form a subtree. The NEC Cenju-3 also supports multicast [18]. The multicast destination must be in consecutive addresses (not necessary a power of 2). However, deadlock is still possible if there are two or more simultaneously multicast communications as shown in this thesis. Subsequently, NEC was decided to disable their hardware multicast support. Note that both the CM-5 and

Cenju-3 support wormhole switching.

Multicasting has been studied in MIN-based ATM switch design, Liew proposed a multicast algorithm for ATM switches using the Clos network in [66, 67] and gave performance evaluation in [68]. Multicast support in [27] involved three segments: a *copy* network, a *distributed* network and a *routing* network. In both cases, an excessive number of MIN stages are needed.

In this chapter, we concentrate on those works related to multicast communication in wormhole-switched networks, including ATM-based, hardware, and software approaches.

8.1 MIN-based ATM Switches

ATM (Asynchronous Transfer Mode) switches which were originally designed for broadband ISDNs have emerged as a promising platform to support high-performance computing, multimedia, and teleconferencing in local area network. Supporting multicast communication within an ATM switch has been studied by many researchers. Many ATM switch designs are based on MINs [67, 69, 12]. A notable design is by J. Turner [27] which has three MINs concatenated together in an ATM switch: the copy network replicates the multicast message, the distribution network randomizes the traffic, and the routing network delivers individual messages to corresponding destinations as shown in Figure 8.1. Each network is a 64×64 multistage baseline network constructed with 2×2 switches¹.

¹In such network, switch s_{ij} can reach 2^{i+1} outgoing ports or reach 2^i outgoing ports through a single link, where $0 \leq i < 6$ and $0 \leq j < 32$.

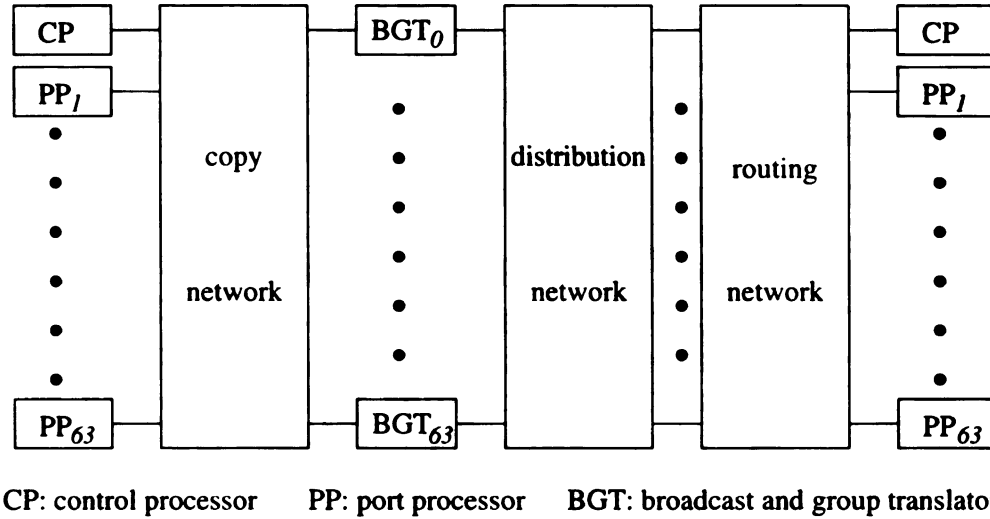


Figure 8.1: Switch fabric.

When a multicast cell² enters the copy network, the number of copies required, m , and the broadcast channel identity, c , are carried in the header. Depending on the number of reachable nodes of a switch, the switch forwards the broadcast cell to a single outgoing link or both outgoing links. If the number of reachable nodes via a single outgoing link is larger than or equal to the number of copies required, m , the cell is forwarded to either the upper outgoing link or the lower outgoing link arbitrarily. Otherwise, the cell is replicated and forwarded to both outgoing links. The number of copies required are revised to m_u and m_ℓ for the upper outgoing link and the lower outgoing link, respectively. The value of m_u and m_ℓ are obtained based on the following rules:

1. If c is even, $m_u = \lfloor (m + 1)/2 \rfloor$ and $m_\ell = \lfloor m/2 \rfloor$.
2. If c is odd, $m_u = \lfloor m/2 \rfloor$ and $m_\ell = \lfloor (m + 1)/2 \rfloor$.

A small version of a 16×16 copy network is shown in Figure 8.2. Considering an

²In ATM, a message is divided into small segments called *cells*. Each cell consists of 53 bytes.

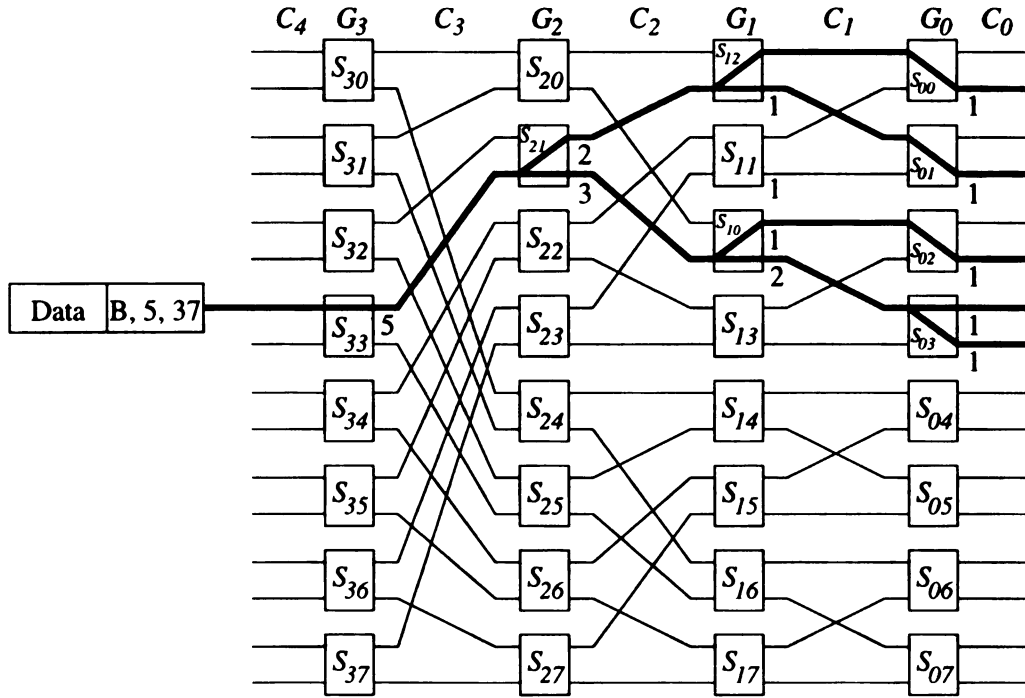


Figure 8.2: A 16×16 copy network.

incoming cell with a header $(B, 5, 37)$ which indicates that it is a broadcast message, the number of copies required m is 5, and the broadcast channel identity c is 37. When switch s_{33} receives this cell, it forwards the cell to only the upper outgoing link since the number of reachable ports via each outgoing link is 8, which is greater than the number of copies required, 5. Note that the upper outgoing link is arbitrarily chosen by the switch for this cell. For the next broadcast cell, this switch may choose the lower outgoing link. Such an arbitrary scheme is used to randomize the traffic for the routing network.

Since switch s_{21} has fewer reachable ports from either outgoing link, it replicates the cell and forwards them to both the upper and lower outgoing links, while the number of copies required is modified to 2 and 3, respectively. The modification of m is based on the second rule specified earlier since c is an odd number. Such process

is repeated in each switch which receives the cell. The value m associated with each forwarding and replicating switch is indicated beside the outgoing link in the figure. Eventually, an exactly number of m copies will reach m distinct outgoing ports.

After the copy network, the real addresses of destinations are taped to packets by table lookup, which is handled by the BGT (broadcast and group translator). A switch in the distributed network ignores the destination address of an incoming cell and forwards the incoming cell to one of its two outgoing links alternately. In other words, a switch forwards an incoming cell to the upper (lower) outgoing link if it delivers the previous incoming cell to the lower (upper) outgoing link no matter what the destination of the incoming cell is. When an outgoing link or both outgoing links are blocked, the switch sends the cell to the first available outgoing link. The purpose of such a process is to randomize the traffic since most researches have shown that uniform traffic has a better performance than other traffic model. The actual routing is performed in the routing network. In general, such network requires $3n$ stages and can handle one multicast message at a time for $N = 2^n$ nodes.

The work by Lee [28] demonstrates that it is possible to deliver multiple multicasts in a synchronous manner which involves a concentrator, a multistage cube network, and a point-to-point network. The concentrator synchronizes the distinct multicast messages and arranges them in consecutive locations. The latter one is achieved by an adder which counts the total number of copies required by all multicast messages in front of each node which initiates a multicast message at this time slot. The routing address of the i^{th} multicast message after the adder becomes $(b_i, b_i + m_i - 1)$ where b_i is the value from the counter and m_i is the number of copies required by the i^{th} multicast

message. Note that when the $b_i + m_i - 1$ is greater than the number of outgoing ports in the copy network, all multicast message after and including the i^{th} one are blocked and will be re-initiated in the next time slot. The middle MIN replicates each of the multicast message and forwards them to associated outgoing ports — all ports between port b_i and port $(b_i + m_i - 1)$ for the i^{th} multicast message. The last point-to-point network delivers the individual messages to associated destinations.

An example is given in Figure 8.3. The source nodes are 4, 7, and 10 and the associated destination sets are $\{5, 8\}$, $\{0, 2, 3, 6\}$, and $\{1, 11, 13, 14\}$, respectively. Since there is no multicast message in front of node 4, the header after the adder is $(0, 1)$. The counter is 2 for the second multicast message initiated by node 7. Hence, the header of the second multicast message becomes $(2, 5)$. Similarly, the header for the third multicast message (initiated by node 10) becomes $(6, 9)$. The concentrator puts these three multicast messages into the first three input ports of the copy network. The copy network replicates and forwards the first, the second, and the third messages to output ports 0 to 1, 2 to 5, and 6 to 10, respectively. After the copy network, the real destination is tagged to each message. The point-to-point switch then delivers these messages to the actual destinations.

This design requires an excessive number of stages, and all multicasts must be synchronized. When the second field of a header for a multicast message is larger than the number of outgoing ports in the copy network, all subsequent multicast messages as well as itself are blocked. Thus, the priority is implicitly given to upper nodes or lower nodes and starvation becomes possible.

In order to avoid the implicit priority, Chen *et al.* extended Lee's work by propos-

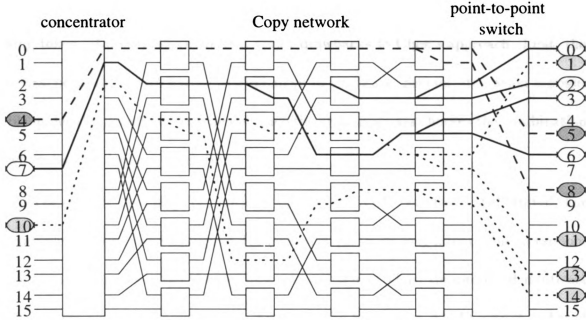


Figure 8.3: An example of synchronous region broadcast.

ing two different approaches: the cyclic priority input access method and the neural network method [70]. The cyclic priority input access method is derived from the ring reservation method. Rather than giving tokens to nodes in ascending or descending order, the tokens are given to the first node which is blocked in a previous time slot due to a lack of enough tokens. The extras tokens are assigned to subsequent nodes until it reaches a node which requires more than the number of tokens left. This approach is similar to round-robin scheduling where there are N sources which can be used in each time slot. The neural network is introduced to increase the network utilization and to decrease the network contention. Instead of forwarding the first cell at each incoming port, this approach allows the network to choose a cell from k cells, $0 \leq k \leq t$, at each incoming port to maximize the network utilization, where k is the number of available cells and t is a predefined threshold. Increasing the threshold would increase the network utilization but also the communication latency. As

expected, the neural network method has slightly better utilization than the cyclic ring token. However, there are two major drawbacks to Chen's approach. First, the neural network is much harder to implement in hardware. Second, starvation is possible under the neural network model since the system always chooses the combination which has the highest utilization.

The current knowledge of multicast support for ATM switches is still limited. For example, a major disadvantage of Turner's work [27] is its long latency due to random message transmission and the need for re-sequencing. Thus, the proposed multicast ability has not been implemented. Although the SynOptics ATM switch is based on Turner's work, the multicast capability is implemented by a high-speed bus [19].

8.2 Hardware Multicast

8.2.1 Path-based Multicast

The *path-based* hardware multicast was proposed to provide efficient multicast while avoiding the potential deadlock for direct networks [25, 71]. In this approach, each processor node is labeled with a relative address based on a Hamiltonian path. A source node divides the destination set into several disjoint subsets to avoid deadlock. Then, the source node sends an individual copy of the message to each subset based on the Hamiltonian order. To reduce the length of a path, the *shortest path* confirming to the Hamiltonian path is proposed. It skips those non-destination nodes in the Hamiltonian path if the skip will not cause deadlock. The router must be capable of

replicating an incoming flit and forwarding one copy to the associated processor and the other to the outgoing channel immediately.

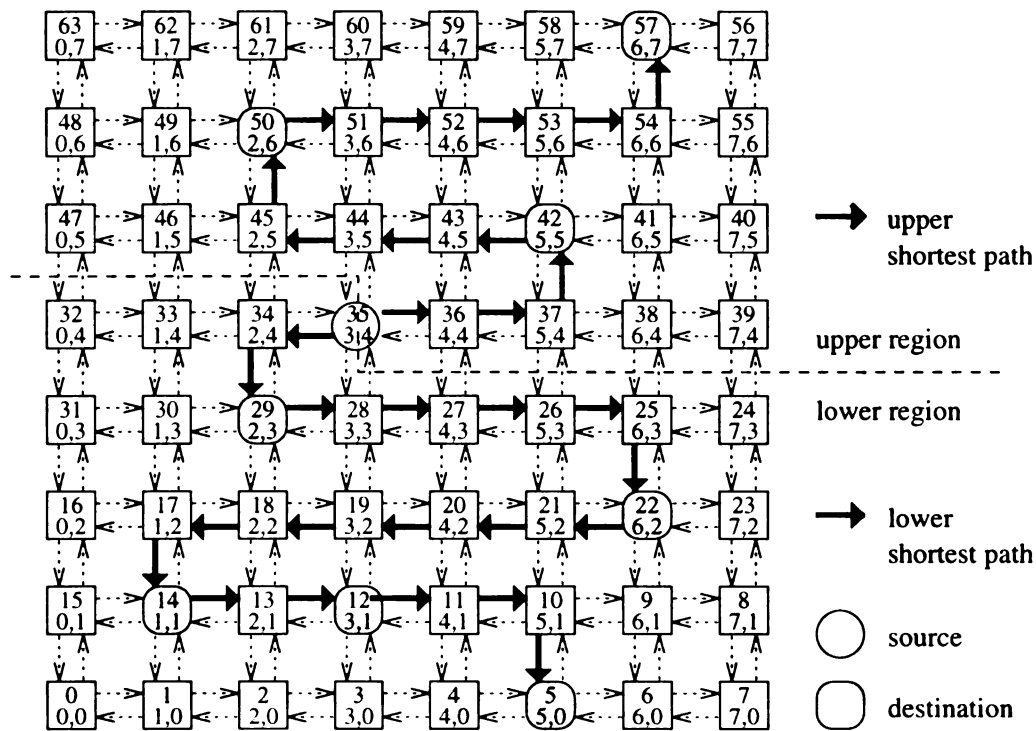


Figure 8.4: An example of a path-based multicast in a 2-D mesh. The lower pair of numbers is the absolute address of a node and the upper number is the relative address of the same node based on a Hamiltonian path.

Figure 8.4 gives an example of path-based multicast in a 2-D 8×8 mesh. The source node is node (3,4), or 35 in relative address, and the destinations are nodes (5,0), (1,1), (3,1), (6,2), (2,3), (5,5), (2,6) and (6,7), or 5, 14, 12, 22, 29, 47, 50, and 57 in relative addresses, respectively. The source node divides the destination set into two disjoint subsets: one subset contains all destination nodes whose relative addresses are smaller than the source node and the other consists of all destination nodes of which relative addresses are larger than the source node. These two subsets are sorted into descending and ascending orders, respectively. As shown in the figure, source node 35 sends one copy of the message followed the upper shortest path to

reach nodes {47, 50, 57} and the other copy followed the lower shortest path to reach nodes {29, 22, 14, 12, 5}, sequentially. Nodes 38 to 41, 46 to 49, and 55 to 56 are skipped by the upper shortest path and nodes 30 to 33, 23 to 24, 15 to 16, and 6 to 9 are skipped by the lower path since such skips will not cause deadlock.

It was shown in [25] that in the intuitive tree-based multicast it is difficult to avoid deadlock in direct networks when there are multiple multicasts. In this thesis we have shown that the *path-based* multicast suffers a potential deadlock in MINs due to self blocking, which is difficult to avoid. The tree-based multicast becomes the natural choice for MINs.

8.2.2 Trip-Based Multicast

Tseng and Panda extended the study in path-based multicast and proposed a *skirt-based* trip multicasting in wormhole-switched networks [72]. A trip in any graph is defined to be a path which visits each node of the graph at least once. The Hamiltonian path, which visits each node once and only once, is a special case of a trip. The deadlock is avoided by using virtual channels, high-channel and low-channel, and by imposing certain restrictions.

After constructing a spanning tree of a multicast, let the tree root be at r . A skirt-based trip is defined as the path that starts from r 's leftmost descendant, ends with r 's rightmost descendant, and wraps around the boundary of T by visiting each node one by one. Figure 8.5 gives an example of a skirt-based trip multicast. There are 10 nodes in the network and their connections are shown in part (a). Part (b)

gives the corresponding spanning tree with skirt. Hence, the trip is $\{p_4, p_1, p_5, p_1, p_0, p_2, p_6, p_9, p_6, p_2, p_0, p_3, p_7, p_3, p_8\}$.

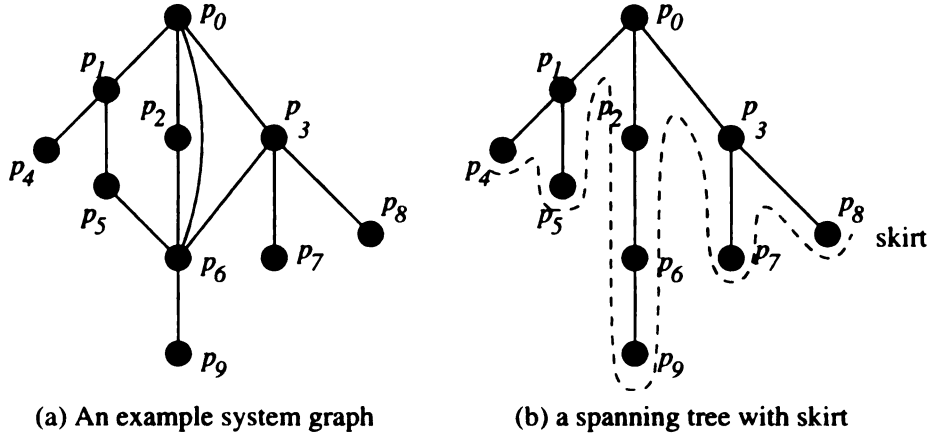


Figure 8.5: An example of a trip-based multicast.

The all-destination encoding/decoding scheme is used in their work implicitly. Like the path-based approach, this work concentrates in direct networks such as hypercube or meshes but not in MINs. A trip-based multicast faces the same difficulty as the path-based multicast in MIN.

8.2.3 A Multidestination Worm Conforming to Base Routing Schemes

To co-exist with unicast routing algorithms such as e-cube, planar adaptive and fully adaptive routing schemes, a multicast approach conforming to base routing schemes, named as *multidestination worm*, was proposed in [73]. A multicast message is divided into several multidestination worms. Each multidestination worm consists of one or more destinations. All destinations in a multidestination worm must be located in a base routing path. Two examples are given in Figure 8.6 where e-cube routing is

used. The first one, shown in Figure 8.6(a), is the same as the example given in

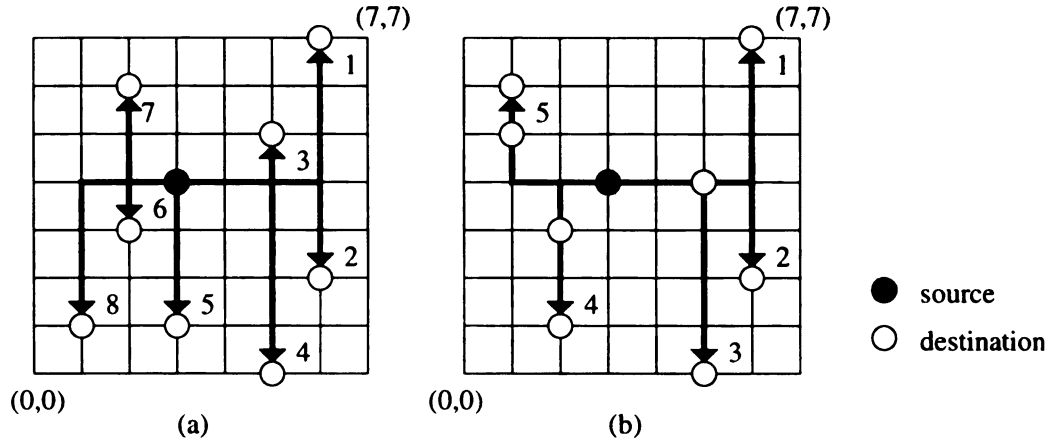


Figure 8.6: Examples of a multidestination worm conforming to base routing schemes.

Figure 8.4. Since every destination is located in a different base routing path, eight multidestination worms are needed to finish the multicast. This is the worst case which is the same as separate addressing. Part (b) of this figure shows a different example, where source is node (3,4) and destinations are nodes (1,5), (1,6), (2,1), (2,3), (5,0), (5,4), (6,2) and (6,7). There are five destination worms. The first two consist of a single destination. The third, fourth, and fifth multidestination worms carry the multicast message for destinations $\{(5,0), (5,4)\}$, $\{(2,1), (2,3)\}$ and $\{(1,5), (1,6)\}$, respectively.

8.2.4 Synchronous Receiver Initiated Multicast

Instead of sender-initiated multicast, Al-Hajery and Batcher proposed a synchronous receiver initiated multicast [74, 75]. In this approach, all nodes intended to receive messages must inform their associated senders. Those nodes without communication requests have to ask themselves to send a dummy message. Otherwise, the underlying

sorted network will fail to deliver messages to correct destinations. Because of the synchronization requirement, it is only suitable for short messages. Other than that, there are two major disadvantages of such a design. First, the setup time in every time slot may cause longer latency. Second, the sorted network requires more switches than a Banyan network.

8.3 Software Multicast

Optimal software multicast for wormhole-switched meshes and hypercubes were first proposed in [39]. The first optimal software multicast for switch-based networks, bidirectional MINs, was given in [51]. The proposed U-min algorithm guarantees contention-free multicast communication for different message sizes and for different start-up latencies. The U-min algorithm was implemented in a 64-node IBM SP-1 and its performance is superior to Chameleon broadcast [76] and MPI-F broadcast [77] operations and is consistently about 30% better than Chameleon application-level broadcast. The U-min algorithm can be easily adapted to the Meiko CS-2 [17].

As shown in [51], the first step of the U-min algorithm is to sort the destinations in lexicographic order. Then it divides the destination nodes into equal halves, in which the first half is one node less than the second half. The source node forwards the message to the closest node in the other half. The last two steps are repeated until all destinations receive the message. An example of source node 2 sending a message to all other nodes in an 8-node MIN is given in Figure 8.7. Source node 2 first sends the message to node 4, then to node 1, and finally node 3 as shown in the

figure. After receiving the message, node 4 forwards a copy to node 6 and then node 5.

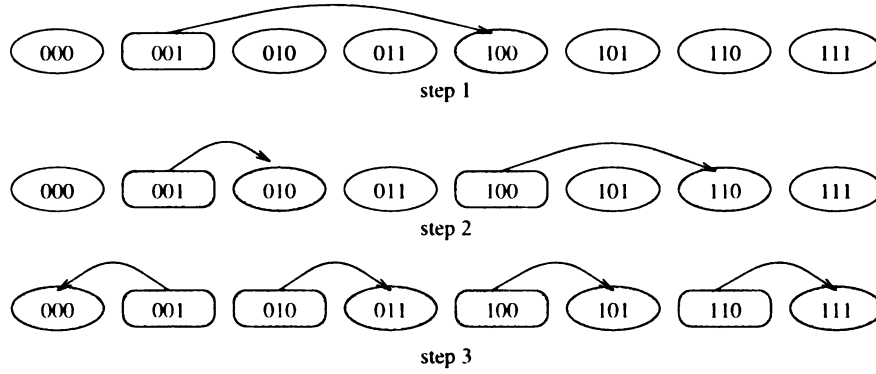


Figure 8.7: An example of a U-min multicast.

The U-min provides an efficient software-based multicast. However, it suffers in a longer queue at the middle nodes when multiple multicasts are allowed simultaneously. Since U-min forwards the message to the center nodes from all source nodes, those center nodes have a much longer queue than the other nodes. Hence, the performance degrades rapidly as the load of multicasts increases.

The C-min avoids such centralized traffic by evenly distributing the senders and the receivers. For example, let there be two multicasts initiated simultaneously in a binary m -cube³. These two multicasts will block each other in some steps unless one is initiated from the leftmost node and the other is started from the rightmost node based on the U-min algorithm. Hence, the blocking probability p_u is

$$p_u = 1 - \frac{1}{C(m, 2)} = \frac{(m-2)(m+1)}{m(m-1)}.$$

³Since a k -ary p -cube may be interpreted as a binary $(k \times p)$ -cube, the following property is also suitable for all k -ary p -cubes

Based on the C-min algorithm, there is no destination contention when these two multicasts are initiated by an odd node and an even node. Therefore, the blocking probability p_c is

$$p_c = 1 - \frac{C(m/2, 1) \times (C(m/2, 1))}{C(m, 2)} = \frac{m-2}{2(m-1)}.$$

Let us use a binary 3-cube as an example. The blocking probabilities p_u and p_c are 0.9643 and 0.4286, respectively. Furthermore, let the source nodes be 0 and 5. The steps of both the U-min and the C-min algorithms are given in Figure 8.8. As shown in part (a), source node 4, which is the destination of source node 0 in the first step, and source node 5 send their messages to node 6 in the second step. In the third step, node 6 needs to send both messages to node 7. Such node contention degrades the performance of the U-min when multiple multicast communications are allowed simultaneously. The corresponding multicast steps of the C-min algorithm are given in Figure 8.8(b). All three steps are contention-free.

Unlike hypercubes, meshes or bidirectional MINs, a contention-free multicast is very complicated and is not always possible for a non-restricted source and destination list in a unidirectional MIN, such as the NEC Cenju-3. In this work, we present a minimum step software multicast in such an environment. It has been proven to be intratask contention-free and to offer better performance than other approaches when the system is not heavily loaded.

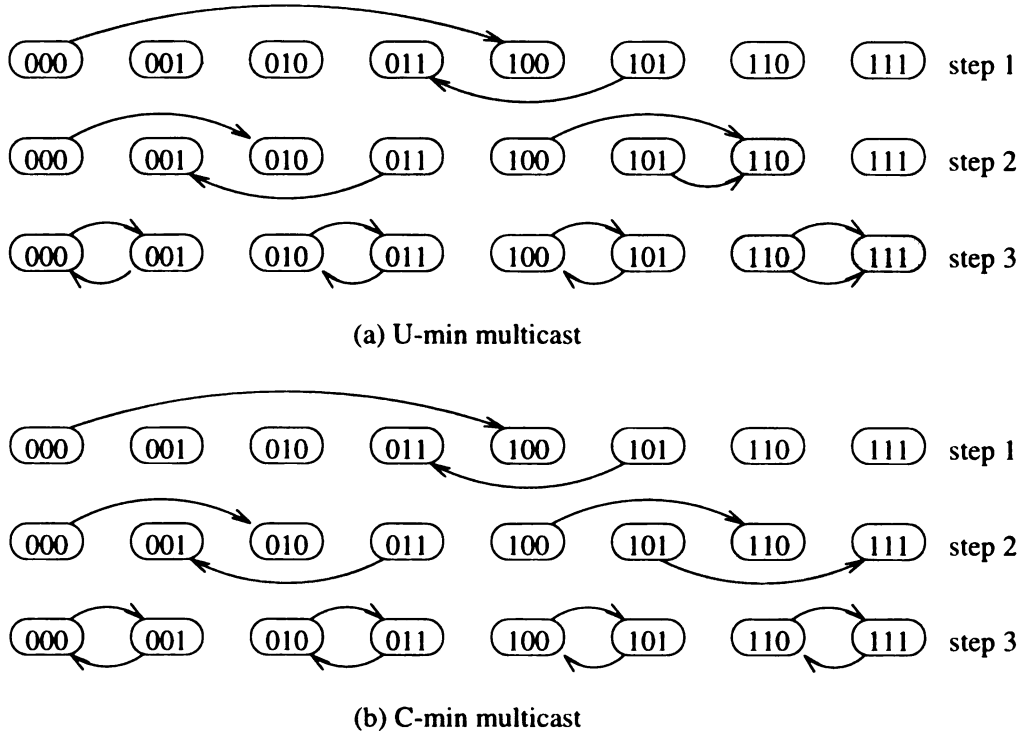


Figure 8.8: An example of two multicasts based on U-min and C-min algorithms.

8.4 Summary

In this chapter, we reviewed several related works and presented the difference between our work and theirs. Since cell size is fixed to 53 bytes and cells are forwarded synchronously in ATM, to support efficient multicast in ATM network is easier than the model we have.

Several hardware approaches were proposed for different network topologies. Although some restrictions are required, none of them serve properly for unidirectional MINs. For example, Turner's design allows one multicast at a time, which is relatively expensive. The nCUBE-2 and NEC Cenju-3 are both forced to disable the hardware support due to the deadlock issue.

For those systems without hardware multicast capability, the efficient software-

based multicast becomes important to applications. The U-min and U-mesh provide an efficient solution for turn-around MIN and mesh, respectively. They fall behind in unidirectional MIN when multiple multicasts are allowed simultaneously.

We have proposed both hardware implementation and a software approach to reduce the latency of multicast communication in unidirectional MINs. Basically, considering the practical intertask contention, allowing multiple multicasts and asynchronous message initiating distinguish our work from others. The contribution of our work as well as some possible future research directions are given in the next chapter.

CHAPTER 9

Conclusions and Future Work

The increasing requirement of multicast communications in various applications necessitates the development of efficient multicast support. This thesis proposes a set of efficient multicast communications, including multi-address header optimization, hardware implementation, and software approaches for unidirectional MINs. In this chapter, we summarize the salient contributions made by this research and present interesting avenues for possible future research.

9.1 Research Contributions

The run-time overhead of communication on variant applications can significantly limit the amount of system parallelism that can be exploited. As more nodes are joined, the communication cost of systems increases. To support efficient and scalable multicast communication becomes critical to the success of various applications, such as parallel programs, teleconferencing, *etc.*

Unlike a unicast message, a multicast message needs to carry all destination in-

formation. However, such header encoding and decoding is overlooked by most researchers. As shown in this thesis, all-destination is the most intuitive approach and is adapted by most works. Few other works refer to single region broadcast or single region mask. Such approaches restrict the destinations of a multicast communication to a contiguous region or to a complete cube, respectively. We have proposed and studied several multi-address encoding and decoding schemes to eliminate such restriction, to give flexibility of destination distribution, and to minimize the header overhead.

Although supporting deadlock-free hardware multicast is difficult, we have shown that hardware multicast implementation has much better performance than the software approach. To avoid deadlock, we have developed a synchronous multi-head worm. We also have shown that it is better than other approaches in different aspects. The NEC Cenju-3 uses an asynchronous multi-head worm whose switches are less complicated than in the synchronous multi-head worm. However, the restriction of single region broadcast is not enough to avoid deadlock. The nCUBE-2 has a similar approach to the synchronous multi-head worm and requires that destinations must form a subcube. It still fails to prevent deadlock. Both restrictions are caused by the multi-address encoding and decoding schemes. We have eliminated these restrictions by proposing flexible multi-address encoding and decoding schemes.

Starvation is another potential problem in the synchronous multi-head worm. We have studied several priority schemes and show that one of them can provide starvation-free multicast communications. When multi-address encoding and decoding is considered with the synchronous multi-head worm, deadlock becomes possible

due to the distance between different headers. A pseudo multi-address encoding scheme is proposed to guarantee a deadlock-free network.

Although processor allocation has been studied extensively, none of that research has studied the corresponding issue when multicast communication is involved. However, in our work, a network partitioning to enforce traffic localization and eliminate intertask contention has been developed. As expected, a system with a systematic network partition outperforms those without such partitionability. Although delta class MINs are topology equivalent, not every one of them has such partitionability. We have shown that both the multistage cube and omega networks have such property and offer better performance. In addition to show that the baseline and butterfly networks have no such property, we have given minimum modification schemes to reconfigure both networks. An existing system with either network topology can offer the same network partitionability as the cube and omega networks after such modification.

There are lots of variant MINs to provide different features. For example, the extra stage MIN provides routing flexibility and fault tolerance. The multiple butterfly network gives multiple paths for any pair of source and destination nodes. The turn-around MIN gives fewer hops for local traffic. Among these variants, we have chosen the extra stage MIN to extend our work. We have shown that to find all traffic-optimal multicast trees on ESMIN is an NP problem, but to find one of them is not. We have developed a polynomial time algorithm to find such a traffic-optimal multicast tree.

Many existing systems only support unicast communication. We have developed

an algorithm to implement efficient software-based multicast for those systems built with unidirectional MINs. The proposed C-min algorithm outperforms the separate addressing as expected. However, when the system is heavily loaded, separate addressing has better performance due to its batch-like multicast.

9.2 Directions for Future Research

In this thesis, we have concentrated our effort on wormhole-switched unidirectional MINs. The performance models and experimental results presented in this work establish the foundation for future study but need to be extended in several ways.

The multi-address encoding and decoding schemes can be easily extended and optimized to networks with different switching techniques or network topologies. We have shown such optimization on unidirectional MINs. More research can be done on other network topologies to evaluate the performance and cost tradeoffs among different encoding schemes and to design efficient and deadlock-free multicast routing algorithms.

The synchronous multi-head worm may be extended to different network topologies, especially to bidirectional MINs, dilated MINs, and virtual-channel MINs. The bidirectional MINs have been adopted in IBM SP1/SP2 and Meiko CS2. Although the optimal software multicast has been studied for the bidirectional MINs, the hardware implementation has not been addressed. As shown in this thesis, the performance of hardware multicast implementation is superior to that of the optimal software approaches in a unidirectional MIN. Hence, the hardware multicast implementation

on these networks will provide better performance. How to establish a multi-head worm on a bidirectional MIN is not obvious since the number of hops among different source and destination pairs may be different. Without careful multi-address encoding and routing, such distance difference may cause deadlock. The dilated MINs provide extra physical links between adjacent stages and the virtual-channel MINs offer channel sharing among different communications. Since both are unidirectional MINs, the multicast algorithms proposed in this research may be applied to them directly. However, how to fully utilize both MINs and how to minimize the communication latency become challenge issues. Furthermore, how to find the optimal number of dilated links and the optimal number of virtual channels on different traffic models are another challenges.

Finally, a particularly challenging direction is to extend the multi-head worm to ATM switches. The cell size in ATM is fixed to 53 bytes, of which five are header and 48 are data. Although there is no deadlock issue, there are some other potential problems, such as out-of-order cell handling, lost data, *etc.* There are several challenges. The first one is to establish efficiently the virtual channel and virtual path for a multicast. The second one is to replicate cells and deliver cells to output port synchronously. Currently, the virtual channel and virtual path establishment is receiver initiated. This approach is the natural way for some applications, such as teleconferencing, but not for parallel programs. To embed the active message becomes another challenging issue.

BIBLIOGRAPHY

Bibliography

- [1] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard," tech. rep., University of Tennessee, Mar. 1994.
- [2] H. Xu, E. T. Kalns, P. K. McKinley, and L. M. Ni, "ComPaSS: A communication package for scalable software design," *Journal of Parallel and Distributed Computing*, vol. 22, pp. 449–461, Sept. 1994.
- [3] H. T. Kung, "Gigabit local area networks: A systems perspective," *IEEE Communications Magazine*, pp. 79–89, Apr. 1992.
- [4] J. Hui, "Switching integrated broadband services by Sort-Banyan networks," *Proceedings of the IEEE*, pp. 145–154, Feb. 1991.
- [5] V. Kompella, J. Pasquale, and G. Polyzos, "Multicasting for multimedia applications," in *Proceeding of the IEEE INFOCOM'92*, Mar. 1992.
- [6] High Performance Fortran Forum, "HPF-2 Scope of Activities and Motivating Applications," Nov. 1994.
- [7] C.-M. Chiang, Q. Du, M. W. Mutka, and R. Sass, "An empirical study of scalable domain decomposition methods for a 2-d parabolic equation solver," in *Proceeding of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, (Norfolk, Virginia), pp. 687–690, SIAM, Mar. 1993.
- [8] Northeast Parallel Architectures Center at Syracuse University, "HPF/Fortran-D Benchmarking Suite (release 2.01)," 1992. (Available at Public Domain at Syracuse University).
- [9] Fore Systems, Inc., *ForeRunnerTM ASX-100 ATM Switch Architecture Manual*, 1992.
- [10] Adaptive Corporation, Network Equipment Technologies, I NC., *ATMX Broadband Switch, Release 1.2*, 1993.
- [11] W. J. Dally and C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.
- [12] J. S. Turner, "Design of local ATM networks." Tutorial Notes at INFOCOM'92, 1992.

- [13] BBN Advanced Computers Inc., Cambridge, Massachusetts, *Inside the GP1000*, 1989.
- [14] BBN Advanced Computers Inc., Cambridge, Massachusetts, *Inside the TC2000 Computer*, 1990.
- [15] W. Gropp, E. Lusk, and S. Pieper, "Users Guide for the ANL IBM SP-1 DRAFT," Tech. Rep. ANL/MCS-TM-00, Argonne National Laboratory, Feb. 1994.
- [16] Thinking Machines Corporation, Cambridge, MA, *The Connection Machine CM-5 Technical Summary*, October 1991.
- [17] Meiko Limited, Waltham, MA., *Computing Surface: CS-2 Communications Networks*, 1993.
- [18] N. Koike, "NEC Cenju-3: A microprocessor-based parallel computer," in *Proc. of the 8th International Parallel Processing Symposium*, pp. 396–401, Apr. 1994.
- [19] J. C. Jerome R., M. R. Gaddis, and J. S. Turner, "Project Zeus," *IEEE Network*, vol. 7, pp. 20–30, Mar. 1993.
- [20] R. J. Souza, P. G. Krishnakumar, C. M. Özveren, R. J. Simcoe, B. A. Spinney, R. E. Thomas, and R. J. Walsh, "The GIGAswitch system: A high-performance packet switching platform," *Digital Technical Journal*, vol. 6, Jan. 1994.
- [21] L. M. Ni, Y. Gui, and S. Moore, "Performance evaluation of switch-based wormhole networks," in *Proc. of the 1995 International Conference on Parallel Processing*, vol. I, Aug. 1995. (accepted to appear, also available as Technical Report MSU-CPS-ACS-96, Dept. of Computer Science, Michigan State University, July 1994).
- [22] Y. Lan, A. H. Esfahanian, and L. M. Ni, "Multicast in hypercube multiprocessors," *Journal of Parallel and Distributed Computing*, pp. 30–41, Jan. 1990.
- [23] NCUBE Company, *NCUBE 6400 Processor Manual*, 1990.
- [24] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, vol. 26, pp. 62 – 76, Feb. 1993.
- [25] X. Lin and L. M. Ni, "Deadlock-free multicast wormhole routing in multicomputer networks," in *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pp. 116–125, May 1991.
- [26] Y. Lan, L. M. Ni, and A. H. Esfahanian, "A VLSI router design for hypercube multiprocessors," *Integration: The VLSI Journal*, vol. 7, pp. 103–125, 1989.
- [27] J. S. Turner, "Design of a broadcast packet switching network," *IEEE Transactions on Communications*, vol. 36, pp. 734–743, June 1988.

- [28] T. T. Lee, "Nonblocking copy networks for multicast packet switching," *IEEE Journal on Selected Areas in Communications*, vol. 6, Dec. 1988.
- [29] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," in *Proc. of the First International Symposium on Computer Architecture*, pp. 21–28, 1973.
- [30] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Transactions on Computers*, vol. C-30, pp. 771–780, Oct. 1981.
- [31] C. L. Wu and T.-Y. Feng, "On a class of multistage interconnection networks," *IEEE Transactions on Computers*, vol. C-29, pp. 694–702, Aug. 1980.
- [32] H. J. Siegel, W. G. Nation, C. P. Kruskal, and L. M. Napolitano Jr., "Using the multistage cube network topology in parallel supercomputers," *Proceedings of the IEEE*, vol. 77, pp. 1932–1953, Dec. 1989.
- [33] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Transactions on Computers*, vol. C-24, pp. 1145–1155, Dec. 1975.
- [34] D. J. Kuck, E. S. Davidson, D. H. Lawrie, and A. H. Sameh, "Parallel supercomputing today and the Cedar approach," *Science*, vol. 231, pp. 967–974, Feb. 1986.
- [35] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Samlley, Jr., and S. D. Smith, "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Transactions on Computers*, vol. C-30, pp. 934–947, Dec. 1981.
- [36] G. F. Pfister *et al.*, "An introduction to the IBM research parallel processor prototype (RP3)," in *Experimental Parallel Computing Architectures* (J. J. Dongarra, Ed.), pp. 123 – 140, Elsevier Science Publishers B.V., Amsterdam, 1987.
- [37] A. Gottlieb, "An overview of the NYU ultracomputer project," in *Experimental Parallel Computing Architectures* (J. Dongarra, Ed.), pp. 25 – 95, North Holland, 1987.
- [38] R. D. Rettberg, W. R. Crowther, P. P. Carvey, and R. S. Tomlinson, "The Monarch parallel processor hardware design," *IEEE Computer*, vol. 23, pp. 18 – 30, Apr. 1990.
- [39] P. K. McKinley, H. Xu, A. H. Esfahanian, and L. M. Ni, "Unicast-based multicast communication in wormhole-routed networks," in *Proceedings of the 1992 International Conference on Parallel Processing*, vol. II, pp. 10–19, Aug. 1992.
- [40] C.-M. Chiang and L. M. Ni, "Multi-address encoding for multicast," in *Proc. of the First International Workshop on Parallel Computer Routing and Communication (PCRCW'94)* (K. Bolding and L. Snyder, Eds.), pp. 146–160, Springer-Verlag, May 1994.

- [41] C.-M. Chiang and L. M. Ni, "Encoding and decoding of address information in multicast messages," in *Proceeding of the 1994 International Computer Symposium*, (HsinChu, Taiwan), pp. 1092–1097, Dec. 1994.
- [42] Cray Research, Inc., Chippewa Falls, Wisconsin, *CRAY T3D System Architecture Overview*, 1993.
- [43] C.-M. Chiang and L. M. Ni, "Network partitioning and unicast-based multicast on multistage networks," Tech. Rep. MSU-CPS-ACS-102, Dept. of Computer Science, Michigan State University, East Lansing, Michigan, Mar. 1995.
- [44] W. Lin and C. L. Wu, "A distributed resource management mechanism for a partitionable multiprocessor system," *IEEE Transactions on Computers*, vol. 37, pp. 201–210, Feb. 1988.
- [45] M. G. Hluchyj and M. J. Karol, "ShuffleNet: An application of generalized perfect shuffles to multihop lightwave networks," *Journal of Lightwave Technology*, vol. 9, 1991.
- [46] C. Y. Chin and K. Hwang, "Packet switching networks for multiprocessors and data flow computers," *IEEE Transactions on Computers*, vol. C-33, pp. 991–1003, Nov. 1984.
- [47] D. K. Panda, S. Singal, and P. Prabhakaran, "Multidestination message passing mechanism conforming to base wormhole routing scheme," in *Proc. of the First International Workshop on Parallel Computer Routing and Communication (PCRCW'94)* (K. Bolding and L. Snyder, Eds.), pp. 131–145, Springer-Verlag, May 1994.
- [48] R. V. Boppana and S. Chalasani, "On multicast wormhole routing in multi-computer networks," in *Proc. of the Sixth IEEE Symposium on Parallel and Distributed Processing*, pp. 722–729, Oct. 1994.
- [49] Y. Lan, "Adaptive fault-tolerant multicast in hypercube multicomputers," *Journal of Parallel and Distributed Computing*, vol. 23, Oct. 1994.
- [50] P. K. McKinley, H. Xu, A. H. Esfahanian, and L. M. Ni, "Unicast-based multicast communication in wormhole-routed networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, pp. 1252–1265, Dec. 1994.
- [51] H. Xu, Y. Gui, and L. M. Ni, "Optimal software multicast in wormhole-routed multistage networks," in *Proceedings of Supercomputing'94*, pp. 703–712, Nov. 1994.
- [52] V. S. Sunderam, "PVM: A framework for parallel distributed computing," *Currency: Practice and Experience*, vol. 2(4), pp. 315–339, Dec. 1990.
- [53] B. Rajagopalan, "Reliability and scaling issues in multicast communication," in *Proceeding of the ACM SIGCOMM*, pp. 188–198, Aug. 1992.

- [54] H. Garcia-Molina and A. M. Spaulster, "Message ordering in a multicast environment," in *Proceedings of the 9th International Conference on Distributed Computing Systems*, pp. 354–361, June 1989.
- [55] H. Eriksson, "MBONE: The multicast backbone," *Communications of the ACM*, vol. 37, pp. 54–60, Aug. 1994.
- [56] D. R. Cheriton and S. E. Deering, "Host groups: A multicast extension to the Internet protocol," Tech. Rep. RFC-966, SRI Network Information Center, Dec. 1985.
- [57] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *ACM Transactions on Computer Systems*, vol. 8, pp. 85–110, May 1990.
- [58] Y. Lan, A. H. Esfahanian, and L. M. Ni, "Distributed multi-destination routing in hypercube multiprocessors," in *Proceedings of the Third Conference on Hypercube Computers and Concurrent Applications*, pp. 631–639, Jan. 1988.
- [59] X. Lin, P. McKinley, and L. M. Ni, "Performance evaluation of multicast wormhole routing in 2D-mesh multicomputers," in *Proc. of the 1991 International Conference on Parallel Processing*, vol. I, pp. 435–442, Aug. 1991.
- [60] D. K. Panda, "Fast synchronization in wormhole k-ary n-cube networks with multidestination worms," in *Proc. of the First High Performance Computer Architecture Symposium*, Jan. 1995. (accepted to appear).
- [61] S. Bhattacharya, G. Elsesser, W.-T. Tsai, and D.-Z. Du, "Multicasting in generalized multistage interconnection networks," *Journal of Parallel and Distributed Computing*, vol. 22, July 1994.
- [62] J. Wu and Z. Li, "A multidestination routing scheme for hypercube multiprocessors," in *Proc. of the 1991 International Conference on Parallel Processing*, vol. III, pp. 290–291, Aug. 1991.
- [63] X. Lin and L. M. Ni, "Multicast communication in multicomputers networks," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1105–1117, Oct. 1993.
- [64] H. Xu, P. K. McKinley, and L. M. Ni, "Efficient implementation of barrier synchronization in wormhole-routed hypercube multicomputers," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 172 – 184, October 1992.
- [65] C. E. Leiserson *et al.*, "The network architecture of the Connection Machine CM-5," in *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, (San Diego, CA.), pp. 272–285, Association for Computing Machinery, 1992.

- [66] S. C. Liew, "Multicast routing algorithms for 3-stage CLOS ATM switching networks," in *Proceedings of the 1991 Globecom*, pp. 1619–1625, 1991.
- [67] S. C. Liew, "Multicast routing in 3-stage Clos ATM switching networks," *IEEE Transactions on Communications*, vol. 42, pp. 1380–1390, February/March/April 1994.
- [68] S. C. Liew, "Performance of various input-buffered and output-buffered atm switch design principles under bursty traffic: Simulation study," *IEEE Transactions on Communications*, vol. 42, pp. 1371–1379, February/March/April 1994.
- [69] J. Y. Hui, *Switching and Traffic Theory for Integrated Broadband Networks*. Norwell, Mass.: Kluwer Academic Pub., 1990.
- [70] X. Chen, J. F. Hayes, and M. K. Megnet-Ali, "Performance comparison of two input access methods for a multicast switch," *IEEE Transactions on Communications*, pp. 2174–2178, May 1994.
- [71] X. Lin, P. K. McKinley, and L. M. Ni, "Deadlock-free multicast wormhole routing in 2-D mesh multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, pp. 793–804, Aug. 1994.
- [72] Y.-C. Tseng and D. K. Panda, "Trip-based multicasting in wormhole-routed networks," Tech. Rep. OSU-CISRC-1/93-TR3, Department of Computer and Information Science, The Ohio State University, 1993.
- [73] D. K. Panda and P. Prabhakaran, "Multicasting using multidestination-worms conforming to base routing schemes," Tech. Rep. Technique Report 37, Department of Computer and Information Science, Ohio State University, Sept. 1993.
- [74] M. Z. Al-Hajery and K. E. Batcher, "Multicast bitonic network," in *Proceedings of the Fifth IEEE Symposium on Parallel and Distributed Processing*, (Dallas, Texas), pp. 320–326, Dec. 1993.
- [75] M. Z. Al-Hajery and K. E. Batcher, "Low cost complexity of a general multicast network," in *Proceedings of the 1994 International Parallel Processing Symposium*, pp. 23–29, Apr. 1994.
- [76] W. Gropp and B. Smith, "Users manual for the Chameleon parallel programming tools," Tech. Rep. ANL-93/23, Argonne National Laboratory, June 1993.
- [77] H. Franke, "MPI-F: An MPI implementation for IBM SP-1," Feb. 1994. Available on anonymous ftp from info.mcs.anl.gov.

MICHIGAN STATE UNIV. LIBRARIES



31293014172534