Deflection Routing
in
Buffered Binary Hypercube Switches

A Thesis Submitted to the College of
Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the Degree of
Ph.D.
in the
Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan

by
Utpal Kanti Mukhopadhyaya

Fall 1998

© Copyright Utpal Kanti Mukhopadhyaya, 1998. All rights reserved.
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
University of Saskatchewan
College of Graduate Studies and Research
SUMMARY OF DISSERTATION
Submitted in partial fulfillment
Of the requirements for the

DEGREE OF DOCTOR OF PHILOSOPHY
By
Utpal Mukhopadhyaya

Department of Computer Science
University of Saskatchewan

Fall 1998

Examing Committee:

Dr. R.J. Bolton    Dean's Designate, College of Graduate Studies & Research
Dr Jim Greer      Graduate Chairman, Department of Computer Science
Dr. Carl McCrosky Supervisor, Department of Computer Science
Dr. Derek Eager   Department of Computer Science
Dr Carey Williamson Department of Computer Science
Dr. Raj Srinivasan Department of Mathematics

External Examiner:

Dr. P. Gburzynski
Professor and Associate Chair
Department of Computing Science
615 General Services Building
University of Alberta
Edmonton, Alberta T6G 2H1
Deflection Routing in Buffered Binary Hypercube Switches

The growing acceptance of B-ISDN (Broadband Integrated Services Digital Network) requires entirely new switching support in a wide range of service demands including voice, video and data. At the same time, advances in the field of VLSI have enabled new principles to the design and architecture of high-performance switching fabrics. Direct binary switch fabrics are a suitable candidate for future B-switches.

Binary hypercubes have regular topology, are highly fault tolerant and have multiple paths for routing cells which help avoid performance penalties due to congestion and faults. In addition, these switches can adopt the novel, distributed, and adaptive routing scheme called 'deflection routing'. In normal routing, cells are routed along shortest paths to their destinations. In case of multiple cells contending for a single outgoing channel, the rest of the contending cells are either buffered or dropped to avoid congestion. In the case of deflection routing, cells can be routed along non-shortest paths. As a result, deflection routing helps avoid dropping cells. The scheme may be implemented with and without queuing buffers at the routers.

In order to properly provision, control, and design these hypercube switches, it is essential that their performance capabilities be completely understood. Researchers have used both analytical model and simulation to evaluate performance of hypercube switches. The presence of distributed logic, multi-path routing, deflection routing, and queuing buffers make modeling tasks highly challenging. Building a reasonably accurate model of a hypercube switch with queuing buffers and deflection routing and using that model to gain practical insights into some of the important design parameters of the switch has been the major motivation of this thesis. An approximate Markov model of a single switching element is built to capture the behavior of a d-dimension switch. The numerical model is solved iteratively. Accuracy of the model is established by validating against simulation results.

One disadvantage of having multiple paths, queuing buffers, and deflection motion of cells in hypercube switches is that the cells belonging to a particular traffic stream may not be delivered at their destinations in sequence. This phenomenon is known as 'out-of-orderness' of cells. An additional goal of this thesis has been development of a model to capture out-of-orderness phenomenon. The model is validated by comparing model results against simulation.

Results show that the model is accurate and reveals significant insight into switch's behavior that can be used to design and engineer d-dimension hypercube switches.
PERMISSION TO USE

Permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5A9
Deflection Routing in Buffered Binary Hypercube Switches

Candidate: Utpal Kanti Mukhopadhyaya
Supervisor: Dr. Carl D. McCrosky

Doctor of Philosophy Thesis
Presented to the College of Graduate Studies and Research
University of Saskatchewan
Fall 1998

Abstract

B-ISDN (Broadband Integrated Services Digital Network) switches can be built using hypercube fabrics which have regular topology. They are highly fault tolerant and have multiple paths for routing cells. Multiple paths help avoid performance penalties due to congestion and faults. These switches may be implemented with and without queuing buffers and may use deflection routing.

In order to properly provision, control, and design these hypercube switches, it is essential that their performance capabilities be completely understood. Researchers have used both analytical models and simulations to evaluate the performance of hypercube switches. The presence of distributed logic, multi-path routing, deflection routing, and queuing buffers make modeling tasks highly challenging. Building an accurate model of a hypercube switch with queuing buffers and deflection routing, and using that model to gain practical insights into some of the important design parameters of the switch has been the major motivation of this thesis. An approximate Markov model of a single switching element is built to capture the behavior of a d-dimension switch. The numerical model is solved iteratively. Accuracy of the model is established by validation against simulation results.

One disadvantage of having multiple paths, queuing buffers, and deflection motion of cells in hypercube switches is that the cells belonging to a particular traffic stream may not be delivered at their destinations in sequence. This phenomenon is known as
out-of-orderness of cells. An additional goal of this thesis has been development of a model to capture out-of-orderness phenomenon. The model is validated by comparing model results against simulation.

The numerical model and the simulation results are compared for the selected performance metrics. Results indicate good agreement between the model and the simulation data and establish accuracy of the model for the assumed uniform traffic with Bernoulli injection. However, significant computation time and memory requirements prevented us from extending the validation of the model for switches with dimension greater than four. Limitations of the model with a restricted number of delivery channels is also pointed out.
ACKNOWLEDGMENTS

The dream came true! I got my Doctorate degree which I had been dreaming for years. It's just that great exciting moment in my life when I would like to thank all those people who have given their thoughtful and helpful guidance over the years. have stood beside me during those frustrating moments of research. and have always inspired me to reach this goal.

I express my sincere gratitude to my supervisor Dr. Carl D. McCrosky for his personal caring. continued guidance. support. and encouragement throughout the research of this thesis. I am fortunate to have Carl as my supervisor. If this thesis has been a reality. it is due in large part to Carl. Thanks. Carl. I am really grateful for all you have done for me.

I would like to thank my thesis committee members Dr. Derek Eager. Dr. Carey Williamson. Dr. Raj Srinivasan. and my external examiner Dr. P. Gburzynski for providing thoughtful and useful criticisms and insights about this work.

I also appreciate the support and co-operation of other faculty and staff members in the Department of Computer Science. Special thanks go to Maureen Desjardins who always has been extremely helpful whenever I asked for it. Special thanks go to Gail Walker also for her kindly encouragement and help during all these years.

Thanks to Canadian Commonwealth Scholarship and Fellowship Committee and Telecommunication Research Labs (TR Labs) for providing scholarship support for my research. Special thanks go to Mr. Cliff Klein at TR Labs. Saskatoon who has always provided constant support and encouragement for the research.

I would like to thank my special friend Pankaj Sud for devoting his precious time and relentless effort for making this thesis a reality.

Thanks go to my daughter Cyonee for the renewal and pleasure I receive from her company and to my wife Chhaya. for not only allowing me the luxury of studying later in life. but also supporting and caring for me.

Thanks go to my parents without whose blessings I could never have realized my dream into reality. Finally. I would like to thank my sister Urmila for her support.
Contents

1 Introduction .................................................. 1

2 Switches .......................................................... 7
   2.1 Shared-Medium Architectures .................................. 9
   2.2 Shared-Memory Architectures .................................. 11
   2.3 Space-Division Architectures .................................. 12
       2.3.1 Crossbar Switches ..................................... 13
       2.3.2 Multi-Stage Switches .................................. 14
   2.4 Distributed Switches ......................................... 21
       2.4.1 Toroidal Switches .................................... 21
       2.4.2 Shuffle Network Switches ............................... 23
   2.5 Direct Binary Hypercube Switches ............................ 25
   2.6 Routing Algorithms for Direct Switches ...................... 28
       2.6.1 The Saturated Constant Shuffle (SCS) Algorithm ........ 31
       2.6.2 Out-of-Orderness .................................... 35
   2.7 Summary .................................................. 36

3 Switch Models .................................................. 37
   3.1 Models of Shared-Medium Switch Architectures ............... 40
   3.2 Models of Shared-Memory Switch Architectures ............... 42
   3.3 Models of Space-Division Switch Architectures ............... 44
   3.4 Models of Distributed Switches ................................ 46
       3.4.1 Greenberg and Goodman’s MSN model ................... 47
3.4.2 One Node Model ........................................... 49
3.4.3 One Cell Model ........................................... 51
3.4.4 Choudhury and Li's Model of 2-connected Networks ........ 52
3.4.5 Abraham and Padmanabhan's Hypercube Model .............. 56
3.4.6 Greenberg and Hajek's Hypercube Model ................. 58
3.5 Szymanski's Hypercube Model ................................ 62
3.6 Motivation of the Thesis .................................... 68

4 An Approximate Model for a Buffered Deflecting Hypercube Switch 70
4.1 Single Node Model ........................................... 72
4.2 Assumptions .................................................. 75
4.3 Analysis ...................................................... 76
  4.3.1 Approximate Algorithm .................................. 81
  4.3.2 Tree Transformation ...................................... 83
  4.3.3 Computation of Marginal Distributions ................... 87
  4.3.4 Computations of Link Utilization Vector ............... 88
  4.3.5 Convergence Criterion .................................... 89
4.4 Summary ..................................................... 90

5 Model for Out-of-Orderness .................................. 91
5.1 Previous Models ............................................. 92
  5.1.1 Gogate and Panwar's Model ............................... 93
  5.1.2 Jean-Marie's Model ....................................... 97
  5.1.3 Choudhury's Model ....................................... 99
5.2 Proposed Model ............................................. 100
  5.2.1 Delay Distributions ...................................... 103
5.3 Summary ..................................................... 112

6 Validation of the Single Node Model ......................... 114
6.1 Experimentation with the Model ............................. 115
6.2 Simulation Results ......................................... 125
6.2.1 Model Validation using Simulation Results .................. 127
6.3 Steady-State Distributions of Vector Spaces ..................... 132
6.4 Why Buffered Deflecting Switches? .............................. 136
6.5 Delivery Channels ............................................. 141
   6.5.1 The Optimum Number of Delivery Channels ............. 146
6.6 Summary ...................................................... 147

7 Validation of Out-of-orderness Model ......................... 153
   7.1 Model Results ............................................. 154
   7.2 Simulation Results and Model Validation .................. 156
   7.3 Summary .................................................. 160

8 Conclusions and Future Work .................................. 162
   8.1 Thesis Summary ........................................... 162
   8.2 Conclusions ............................................... 164
   8.3 Contributions ............................................. 165
   8.4 Future Work ................................................ 165

9 References ..................................................... 168

A Simulation Run Length ........................................... 175
   A.1 Determining Sample Size .................................. 177
   A.2 Transient Removal ......................................... 178

B The Vector Spaces ............................................. 180
List of Figures

2.1 NEC Shared-Medium Switch ........................................... 10
2.2 Hitachi’s Shared-Buffer Switch ....................................... 12
2.3 Crossbar Switches ..................................................... 13
2.4 Crossbar Switch with Buffers at Crosspoints ..................... 14
2.5 A 1 × 8 Multistage Binary Tree ...................................... 14
2.6 A 3 Stage $N \times N$ Multistage Switch ............................. 15
2.7 8 × 8 Delta-2 Network ............................................... 16
2.8 Types of Blocking: (a) Internal Link Blocking (b) Output Port Blocking ...................................................... 17
2.9 Three Types of Internal Buffering: (a) Input Buffering (b) Output Buffering and (c) Shared Memory ............................... 19
2.10 Two-stage Banyan Networks with: (a) 2-Dilation (b) 2-Replication .......... 20
2.11 Torus Network ......................................................... 22
2.12 n-Dimensional Meshes and k-ary n Cubes: (a) 2-Dimensional Mesh (b) 3-Dimensional Mesh (c) 4-ary 1-Cube or Ring (d) 2-ary 3 Cube ........ 23
2.13 4 × 4 Manhattan Street Network .................................... 24
2.14 A ShuffleNet with $k = 3$. ........................................... 25
2.15 Hypercubes for Various Dimensions: (a) Dimension 0. (b) Dimension 1. (c) Dimension 2. (d) Dimension 3. and (e) Dimension 4. ........... 27
2.16 Single Switching Element of a Hypercube .......................... 27
2.17 Details of a Single Switching Element of a Hypercube ................ 32

3.1 Model of a Nonblocking Shared-Medium Switch ..................... 41
3.2 Model of a Blocking Shared-Medium Switching Element ............. 42
3.3 The Queueing Structure of Subsystem \( i \) ........................................ 44
3.4 Queueing Network Models of a \( 4 \times 4 \) Banyan Network with I/O Buffers:
   (a) \( 4 \times 4 \) Banyan Network: (b) Input Buffering: (c) Output Buffering:
   (d) Input/Output Buffering ................................. 45
3.5 \( 4 \times 4 \) MSN Network ........................................ 47
3.6 MSN Grid ...................................................... 48
3.7 The Structure of a Node ........................................ 53
3.8 State Transition Diagram for Buffer Utilization with 3 Buffers per Out-
   going Link ................................................. 54
3.9 Iterative Evaluation of \( \pi_t \) and \( m_t \) .......................... 55
4.1 Single Switching Element of a Hypercube ..................... 72
4.2 Probability Tree Showing the Generation of States ............ 85
5.1 Re-sequencing Model Using System of Parallel \( M/M/1 \) Queues with
   Delay Lines .................................................. 94
5.2 Re-sequencing Time of a Customer ............................ 98
5.3 Routing Outcomes of a Cell in Injection Buffer ............... 103
5.4 Routing Outcomes of a Cell in Queue-in Buffer ............... 104
5.5 Routing Outcomes of a Cell in Forward-in Buffer ............. 104
5.6 An Example of Computing Routing Outcomes of a Tagged Cell in a
   Queue-in Buffer ............................................ 106
5.7 Another Example of Computing Routing Outcomes of a Tagged Cell
   in a Queue-in Buffer ....................................... 107
5.8 Markov Chain Showing the Walk of a Tagged Cell ............. 111
5.9 Computing the Delay Distributions of a Cell with a Probability Tree
   Analysis ..................................................... 112
6.1 Transient Behavior of a Dimension 3 Hypercube Switch with 4 Delivery
   Channels at an Injection Load of 0.9 ........................ 118
6.2 Transient Behavior of a Dimension 4 Hypercube Switch with 5 Delivery Channels at an Injection Load of 0.9 ............................................... 119

6.3 Transient Behavior of a Dimension 3 Hypercube Switch with 4 Delivery Channels as a Function of the Queue Buffer Size at an Injection Load of 0.9 ............................................... 119

6.4 Model Data for Average Number of Cells in Queue-In Buffers of a Dimension 3 Hypercube Switch with Queueing Buffers (Queue Size = 1), and Retrograding ......................................................... 120

6.5 Model Data for Average Number of Cells Retrograded in a Dimension 3 Hypercube Switch with Queueing Buffers (Queue Size = 1), and Retrograding ......................................................... 121

6.6 Retrograde Performance of Cells in a Dimension 3 Hypercube Switch with 4 Delivery Channels with Variation in Injection Load for Three Different Queue Buffer Sizes (Queue Size = 1, Queue Size = 2, and Queue Size = 3) ......................................................... 122

6.7 Cell Loss Probability of a Dimension 3 Hypercube Switch with 4 Delivery Channels with Variation in Injection Load for Three Different Queue Buffer Sizes ......................................................... 123

6.8 Model Data for Average Number of Cells in Queue-In Buffers of a Dimension 4 Hypercube Switch with Queueing Buffers (Queue Size = 1), and Retrograding ......................................................... 124

6.9 Model Data for Average Number of Cells Retrograded in a Dimension 4 Hypercube Switch with Queueing Buffers (Queue Size = 1), and Retrograding ......................................................... 125

6.10 Retrograde Performance of Cells in a Dimension 4 Hypercube Switch with 5 Delivery Channels with Variation in Injection Load for Four Different Queue Buffer Sizes (Queue Size = 1, Queue Size = 2, Queue Size = 3, and Queue Size = 4) ......................................................... 126

6.11 Cell Loss Probability of a Dimension 4 Switch with 5 Delivery Channels with Variation in Injection Load for Four Different Queue Buffer Sizes 127
6.12 Average Number of Cells in Queue-In Buffers of a Dimension 3 Hypercube Switch with Queuing Buffers (Queue Size = 1), and Retrograding

6.13 Average Number of Cells Retrograded in a Dimension 3 Hypercube Switch with Queuing Buffers (Queue Size = 1), and Retrograding

6.14 Average Number of Cells in Queue-In Buffers of a Dimension 4 Hypercube Switch with Queuing Buffers (Queue Size = 1), and Retrograding

6.15 Average Number of Cells Retrograded in a Dimension 4 Hypercube Switch with Queuing Buffers (Queue Size = 1), and Retrograding

6.16 Average Number of Retrograded Cells per CEC for a Dimension 3 Deflection Switch

6.17 Average Number of Cells in Queue-In Buffers of a Dimension 3 Hypercube Switch with Queuing Buffers (Queue Size = 1) Only and No Retrograding

6.18 Probability Mass Functions of the Queue-In Vector Space of a Dimension 3 Hypercube Switch with 4 Delivery Channels and Queuing Buffers (Queue Size = 3)

6.19 Probability Mass Functions of the Forward-In Vector Space of a Dimension 3 Hypercube Switch with 4 Delivery Channels and 3 Queue Buffers

6.20 Cell Loss Probability of a Dimension 3 Hypercube Switch with $d - 1$ Delivery Buffers under Various Architecture Scenarios

6.21 Comparison of Cell Loss Probabilities of Buffered Switches with Two Different Queue Buffer Sizes (Queue Size = 1 and Queue Size = 3) with Deflection and Buffered Deflection Switches

6.22 Comparison of Cell Loss Probabilities of a Dimension 3 Hypercube Switch with 4 Delivery Channels and Two Different Queue Buffer Sizes (Queue Size = 1 and Queue Size = 3)

6.23 Comparison of Average Latency for a Dimension 3 Hypercube Switch with 4 Delivery Channels for Three Different Variants of Switches at Various Injection Loads
6.24 Transient Behavior of a Dimension 3 Hypercube Switch with Queue Buffers (Queue Size = 3) as a Function of Delivery Channels at an Injection Load of 0.9 .................................................. 142
6.25 Average Number of Cells in Queue Buffer for a Dimension 3 Hypercube Switch under Single Delivery Buffer Assumptions. Queue Size = 1 .... 150
6.26 Average Number of Cells Retrograded for a Dimension 3 Hypercube Switch under Single Delivery Assumptions. Queue Size = 1 .... 150
6.27 Probability Mass Functions of the Queue-In Vector Space of a Dimension 3 Switch with 3 Queue Buffers for Two Different Numbers of Delivery Channels .................................................. 151
6.28 Probability Mass Functions of the Forward-In Vector Space of a Dimension 3 Switch with 3 Queue Buffers for Two Different Numbers of Delivery Channels .................................................. 151
6.29 Average Number of Cells in Queue Buffer for a Dimension 3 Hypercube Switch under Two Delivery Buffer Assumptions. Queue Size = 1 .... 152
6.30 Average Number of Cells Retrograded for a Dimension 3 Hypercube Switch under Two Delivery Buffer Assumptions. Queue Size = 1 .... 152

7.1 Model Data for Delay Distributions of Cells with Hopcount 1 for a Dimension 3 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0 .................................................. 155
7.2 Model Data for Delay Distributions of Cells with Hopcount 2 for a Dimension 3 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0 .................................................. 155
7.3 Model Data for Delay Distributions of Cells with Hopcount 3 for a Dimension 3 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0 .................................................. 156
7.4 Model Data for Delay Distributions of Cells with Hopcount 1 for a Dimension 4 Hypercube Switch with d+1 Delivery Buffers, Queue Size = 1. and Load = 1.0 .................................................. 156
7.5 Model Data for Delay Distributions of Cells with Hopcount 2 for a Dimension 4 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size $= 1.$ and Load $= 1.0$ ........................................ 157

7.6 Model Data for Delay Distributions of Cells with Hopcount 3 for a Dimension 4 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size $= 1.$ and Load $= 1.0$ ........................................ 157

7.7 Model Data for Delay Distributions of Cells with Hopcount 4 for a Dimension 4 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size $= 1.$ and Load $= 1.0$ ........................................ 158

7.8 Variation of Total Latency of Cells in a Dimension 3 Hypercube ($d+1$ Delivery Buffers. Queue Size $= 1.$ and Load $= 0.9$) with Injection Load 158

7.9 Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 1 for a Dimension 3 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size $= 1.$ and Load $= 1.0$ ........ 159

7.10 Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 2 for a Dimension 3 Hypercube Switch $d+1$ Delivery Buffers. Queue Size $= 1.$ and Load $= 1.0$ ........ 159

7.11 Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 3 for a Dimension 3 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size $= 1.$ and Load $= 1.0$ ........ 160

7.12 Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 1 for a Dimension 4 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size $= 1.$ and Load $= 1.0$ ........ 160

7.13 Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 2 for a Dimension 4 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size $= 1.$ and Load $= 1.0$ ........ 161

7.14 Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 3 for a Dimension 4 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size $= 1.$, and Load $= 1.0$ ........ 161
7.15 Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 4 for a Dimension 4 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size $= 1$, and Load $= 1.0$ . . . . . 161
List of Tables

6.1 Model Data for Average Number of Cells in Forward-In Buffers of a Dimension 3 Hypercube Switch with Queueing Buffers (Queue Size = 1) and Retrograding ............................................. 120
6.2 Model Data for Average Number of Cells in Forward-In Buffers of a Dimension 4 Hypercube Switch with Queueing Buffers (Queue Size = 1) and Retrograding ............................................. 121
6.3 Link Utilization Vectors at Various Loads for 4 Delivery Channels ............................................. 123
6.4 Comparison of the Hopcount Distribution of the Cells Generated in the Simulator with the Theoretically Computed Value at a Load of 0.9 ............................................. 128
6.5 Average Number of Cells in Forward-In Buffers of a Dimension 3 Hypercube Switch with Queueing Buffers (Queue Size = 1) and Retrograding ............................................. 129
6.6 Average Number of Forward-In Cells in a Dimension 4 Hypercube Switch with Queueing Buffers (Queue Size = 1) and Retrograding ............................................. 130
6.7 Average Number of Forward-In Cells for a Dimension 3 Hypercube Switch with Retrograding Features Only ............................................. 131
6.8 Average Number of Cells in Forward-In Buffers of a Dimension 3 Hypercube Switch with Queueing Buffers (Queue Size = 1) Only and No Retrograding ............................................. 132
6.9 Steady-State Probability Distributions of the Queue-In Vectors of a Dimension 3 Hypercube Switch with 4 Delivery Channels and a Queue Buffer (Queue Size = 3) ............................................. 133
6.10 Average Number of Forward-In Cells for a Dimension 3 Hypercube Switch under Single Delivery Assumptions. Queue Size = 1 .......................... 142
6.11 Link Utilization Vectors at Various Loads in a Dimension 3 Hypercube Switch with Single Delivery Channel .................................................. 143
6.12 Comparison of Retrograde Performance of a Dimension 3 Hypercube Switch at Various Loads for 4 Delivery Channels and Single Delivery Channel with Same Queue Buffer Size (Queue Size = 1) .................. 143
6.13 Average Number of Forward-In Cells for a Dimension 3 Hypercube Switch under Two Delivery Buffer Assumptions. Queue Size = 1 .... 144
6.14 Table Comparing the Performance Figures of Model with 4 Delivery Channels and 2 Delivery Channels with Queue Size = 1 .................. 144
B.1 Mapping between the Vector Space Components and the Index Space 182
Chapter 1

Introduction

Existing telecommunications networks have independent networks operating in parallel for different applications like voice, video and data. Each of these networks is efficient for the application for which it is designed, but not suitable for supporting other applications [4]. A unified integrated communications system has the advantage that not only can it accommodate a variety of services with different bandwidth and quality of service requirements, but also it allows unified network management, operation, and maintenance [59, 60]. B-ISDN (Broadband Integrated Services Digital Network) is one such integrated communications system that can support many different types of services in a flexible and cost-effective manner [50].

B-ISDN networks will require new switch architectures. Rapid advances in the area of transmission systems and fiber optics, and in the field of VLSI technology have brought about completely new principles in the design and architecture of high-performance switching fabrics. The two most common switching techniques employed in high-performance switches have been circuit switching and packet switching. The traditional circuit switching concept which evolved primarily to transport stream-type traffic like voice and video provides fixed throughput and constant delay at a fixed rate of transmission, but makes poor utilization of the resources when sources are bursty. Packet switching on the other hand, is characterized by buffering, statistical multiplexing, and variable throughput and delay. It provides better utilization of the communication resources for bursty sources because of dynamic sharing.
The International Telegraph and Telephone Consultative Committee (CCITT) proposed Asynchronous Transfer Mode (ATM) as the "target transfer mode solution for implementing a B-ISDN" [13, 50]. ATM was recommended as a world-wide standard to create a broad-band (high-speed) switching network capable of transporting a wide variety of services like voice, video, and data in an integrated manner. Under ATM, a single-class high performance cell switching network would carry voice, video and data, in the form of short, fixed-size "cells" that would reduce packetization delay, reduce potential queueing delay and simplify engineering. Packets belonging to various traffic types are decomposed into cells at the source end, multiplexed, and then transmitted. Cell switches receive the cells on their input ports, carry out some form of address translation of the received cells and retransmit them on their outgoing links. At the destination end, cells are re-assembled into packets before they are delivered to the appropriate protocol layer. Cell switching has emerged as one of the most popular switching techniques to support a wide range of services.

Many different cell switching systems have been proposed over the past few years. Based on the fabric architectures, these switches have been classified as shared-medium switches, shared-memory switches, space-division switches, and distributed switches. The design of some of the switches exploits a high degree of parallelism and distributed control in order to achieve high performance. One such class of high-performance cell switches that has attracted attention is the binary $n$ cubes, or hypercubes. These switches form a subset of a larger class that includes $k$-ary $n$-cubes. Binary hypercubes have regular topology, are potentially fault tolerant, and have multiple paths for routing cells which help avoid performance penalties due to congestion and faults. Also, the diameter (maximum distance between two switching elements or nodes) of a hypercube switch increases only logarithmically with the number of switching nodes.

Nodes or switching elements in a hypercube switch are symmetric, i.e., the number of incoming links is equal to the number of outgoing links. Switching and transmission of cells are slotted, and synchronized. Nodes attempt to route cells along shortest paths to their destinations. Congestion occurs when multiple cells contend for a
single outgoing link. Based on the information carried in cell headers, if an output link happens to be the only choice for two or more cells, only one of them is forwarded and the rest are dropped. This is true in switches with no queueing buffers and running a very simple cell routing algorithms. However, by providing queueing buffers at the nodes, the loss of these transit cells (i.e., cells already accepted into the switch for delivery) can be reduced. Switches with queueing buffers at their nodes are known as buffered hypercube switches. Provision of queueing buffers at switching elements achieve lower cell loss at the cost of increased cell latency. The increased latency is due to the queue latency of cells. To achieve zero cell loss in buffered hypercube switches, buffer sizes need to be infinite.

An interesting alternative to achieve zero cell loss without having to use queueing buffers at the nodes, is to deflect the cells losing contention to non-optimal links. These links are non-shortest paths for the cells. Originally proposed as hot-potato routing [8] in switches that have nodes with an equal number of incoming and outgoing links. Deflection routing helps avoid queueing buffers and dropping packets. The scheme has been used to build very high-speed networks in the areas of multi-processor interconnection networks [27], metropolitan area networks (MAN) [14], all-optical networks [6] and also in hypercube networks [24, 46, 57].

Hypercube switches employing deflection routing [47] achieve superior performance (zero cell loss), reliability, and robustness, in terms of choice of routing strategies and fluctuating traffic, but at the cost of increased cell latency. Whenever a cell is deflected to a non-optimal link, it travels a number of extra hops called the deflection index. In the case of hypercubes, the value of the deflection index is 2. This causes extra latency for the deflected cells, called the retrograde latency. The total latency of cells in switches without queue buffers and employing deflection routing is larger than that in switches with queue buffers and no deflection routing.

A small amount of buffering inside the switching elements of a hypercube switch that employs deflection routing reduces the probability of deflection which in turn reduces the latency figures of the cells. In the event of congestion, when the buffer becomes full, cells that lose contention are deflected. This mixed approach avoids
loss of cells, avoids large buffer size, and keeps latency figures of the cells lower than either pure queueing or pure retrograd ing solution. The amount of buffering is a tradeoff between node complexity and improvement of switch performance [11]. Buffers within nodes can be provided in a number of ways. The two most widely used buffering schemes are: output buffering, in which buffers are dedicated to each output link of a switching element; and shared buffering, in which buffers are shared among all output links. Shared buffering needs substantially lower buffer capacity compared to output buffering in order to achieve similar switch performance [28].

Our focus in this thesis is on a performance study of hypercube switches with shared queueing buffers and deflection routing. In order to properly design, provision, and control these switches it is important to understand the performance of these switches. Performance studies have been carried out for a variety of related interconnection networks like Manhattan Street Networks [25, 44] the Shuffle and Shuffle Exchange Networks [16, 39, 65], and the Hypercube Networks [24, 57]. These studies have shown that performance depends on a number of parameters, such as the switch architectures, the switch diameters, the node connectivity (i.e., the number of inlets and outlets of each node), the queueing buffers (if any), the deflection index (if deflection is used), and also the routing algorithm running in the switch. We assume, in our performance study, a deflection routing algorithm called the SCS (Saturated Constant Shuffle) algorithm developed by McCrosky [46] that works well with buffered hypercube switches.

Researchers have used both analytical models and simulations to evaluate performance of hypercube switches. The presence of distributed logic, multi-path routing, deflection routing, and queueing buffers make the modeling task highly challenging. Some researchers have provided analytical models for buffered hypercubes but did not consider deflection routing. Some have developed models for hypercubes using deflection routing, but avoided storage buffers. No single analytic model in the literature studies buffered deflection routing in hypercube switches. Building a reasonably accurate model of a hypercube switch with queueing buffers and deflection routing and using that model to gain practical insights into some of the important design
parameters of the switch is major motivation of this thesis work. We build an approximate Markov model of the switch to capture switch behavior. The numerical results of the model are validated against simulation data obtained from the switch simulator we build.

Hypercube switches suffer from one major disadvantage. The cells belonging to a particular traffic stream may not be delivered at their destinations in sequence. This phenomenon, known as out-of-orderness of cells, happens because of the presence of multiple paths, queueing buffers, and retrograde motions of the cells in the switch. Re-assembly buffers are needed at the switching nodes to sequence the cells before they are delivered to the destination hosts connected to the nodes. However, studies show that the probability of cells getting severely out-of-order is very low. We build a model that captures this out-of-orderness phenomenon and validate the model against simulation results. An additional goal of this thesis is to build this out-of-orderness model.

The rest of the thesis is organized as follows. Chapter 2 presents background information on switch architectures with particular emphasis on hypercube switches and a particular routing algorithm called the Saturated Constant Shuffle (SCS) algorithm. Chapter 3 provides a brief survey of the models used for performance study of hypercubes and other direct networks like the Manhattan Street Network. Motivation for the present research is further developed there. A new analytical model of a buffered hypercube employing deflection routing is developed in Chapter 4.

A potential drawback of using queueing buffers and retrograding in hypercube switches is out-of-order arrival of cells at the switching elements. Chapter 5 provides a brief overview of the models developed by previous researchers for capturing the out-of-order delivery problems. The same chapter also develops a model based on a single tagged cell to capture the out-of-orderness phenomenon.

Validation of the numerical model developed in Chapter 4 is carried out in Chapter 6. in which results obtained from both the model and the simulation are compared. A design insight regarding the optimal number of delivery channels that the switching elements should have is obtained from computation of throughput bounds. Using
both numerical model data and simulation results. Chapter 6 shows the superior performance of buffered deflection hypercube switches compared to buffered switches or deflection switches.

The out-of-orderness model developed in Chapter 5 is validated in Chapter 7. The delay distributions of a tagged cell are compared with simulation results. It is shown that the probability of a tagged cell arriving before a cell transmitted three or more time units earlier is vanishingly small.

Chapter 8 summarizes the thesis work, points out the contributions made and the future research that can be further pursued.
Chapter 2

Switches

Switching systems provide simultaneous interconnection of arbitrary pairs of network users without requiring a completely connected network. A switch consists of three major components: input ports, output ports, and switching fabrics. An $N \times N$ switch can be viewed as a black box, with $N$ input links and $N$ output links, that transports cells from any incoming link to any outgoing link. Incoming links are connected to the switching fabric through input ports. After the cell header is processed to determine its outgoing link, it is passed to the switching fabric to be delivered to its outgoing link. The interface between the fabric and the outgoing link is referred to as the output port. Switching fabrics are patterned collections of simple switching building blocks that serve to move cells. These fabrics may be centralized or distributed. Distributed fabrics consist of collections of switching elements. Distributed fabrics have been used for applications such as metropolitan area networks or MANs.

When more than one cell attempts to access an output port simultaneously, output conflict occurs, in which case only one of the contending cells is transmitted by the output port and the remaining cells are stored in a buffer until they can be transmitted or else dropped. If output buffering is provided these cells are stored between the switch fabric and the output port. If input buffering is provided these cells are kept in storage between the incoming link and the input port. There is one more type of buffering, called internal buffering which is provided within the fabric itself. When cells contend for the same resource simultaneously within the switch fabric, internal
blocking arises and similar to the case of output port contention, all but one of the cells may be temporarily stored at buffers within the fabric.

Switches are characterized by their capability of providing multiple concurrent connections between disjoint input-output pairs. A switch is said to be non-blocking if any desired connection between an unused input port and an unused output port can be established immediately without interference from existing connection. A blocking switch may prevent a desired connection from being established between unused ports, because of already existing connections.

The two most common methods of transferring information between the source and the destination ports of a switch are: circuit switching and cell switching. In circuit switching, a physical path is established from the requesting source to the desired destination and the connection is usually maintained for the duration of a "call". The fixed path ensures that the data from the source arrives at its destination in the correct sequence and with minimal delay. As a result, circuit switching provides fixed throughput and constant delay at a fixed rate of transmission. However, circuit switching may result in poor utilization of the resources, because they are reserved according to the peak requirements of the source. If the source is bursty, these exclusively reserved resources will not be fully utilized during the time when the transmission rate falls below the peak rate. In cell switching, no physical paths are set up between the source and the destination ports. The source sends fixed-length cells of information, each carrying the destination address. The switches forward them one by one on the appropriate outgoing links, based on the destination information. Cell switching may or may not provide storage for cells. Intermediate switching elements without storage may either forward or drop cells. Cell switches with storage - also known as store-and-forward switches - store cells temporarily until appropriate paths are available for routing. The data transfer rate often depends on the system's traffic load, since cells may be held at switching buffers waiting for a clear path when traffic is heavy. Cell switching does not exclusively reserve the switching resources and thereby offers better utilization of those switching resources under variable loads. However, cell switching suffers from variable throughput and variable delay.
In order to meet diverse service requirements of existing and future B-ISDN applications, a need for high-performance switching systems has been recognized. Many different switching systems have been proposed and developed over the past few years. Most current approaches of high performance switching fabrics employ a high degree of parallelism and distributed control [4]. Cell switching has been chosen as the underlying switching technique to support a wide range of services with different bit rates. Our emphasis in this chapter will be on switches which incorporate the cell switching concept. Based on the fabric architectures, high-performance cell switches are classified into the following four categories [51):

- Shared-Medium architectures.
- Shared-Memory architectures.
- Space-Division architectures.
- Distributed architectures

Sections 2.1, 2.2, and 2.3 review the first three switching architectures. A new class of switching architectures based on distributed fabrics or switching elements is discussed in Section 2.4. Distributed switches like Toroidal switches, ShuffleNet, and Shuffle Exchanges - all have been based on the concepts of interconnection network topologies used in connection with MAN or WAN. A special type of switch, known as the direct binary hypercube switch and a particular routing algorithm, known as the SCS algorithm, are of particular interest to us. Section 2.5 gives the switch configuration and Section 2.6 discusses the algorithm. Finally, we summarize at the end of this chapter.

2.1 Shared-Medium Architectures

In a shared-medium architecture, incoming cells are multiplexed into a common medium, typically a bus or a ring. The medium speed, in general, is greater than or equal to the sum of the transmission rates of incoming links connected to it. A small
FIFO with a capacity to hold only a few cells is sufficient to store incoming cells until they can access the medium. The number of links that can be provided is limited by the bus speed. There is no fabric output contention in shared-medium architecture, because two or more cells cannot arrive simultaneously at an output port. However, the arrival rate of cells at a particular outgoing link may exceed the link bandwidth for a short while, in which case output buffers may be used to store cells that arrive at a faster rate than they can be served. An example of a shared-medium switch developed by NEC [56] is shown in Figure 2.1. As shown in the figure, each switching element consists of a bus and output buffers.

![NEC Shared-Medium Switch](image)

Figure 2.1: NEC Shared-Medium Switch

Another experimental high-speed cell switch (PARIS) [23], designed to transport voice, video, and data all in cell form, has a bus whose bandwidth is greater than the aggregate capacity of all incoming links and has both input and output buffers.

One major problem with shared-medium architectures is that they do not scale well, and can support only a small number of ports. As the number of links attached to the switch and the speed of the links increase, the medium speed becomes a bottleneck. However, they can be used as the building blocks of large switches in which these units are connected to each other according to some larger topology.
2.2 Shared-Memory Architectures

A shared-memory switch consists of a single dual-ported memory module shared by all input and output ports. All arriving cells are multiplexed into a single stream and are written to the shared memory. The memory is organized into logical queues, one for each output port. Cells at the output queues are also multiplexed into a single stream, read out, demultiplexed, and transmitted on the output lines [51].

The memory can be organized logically as either fully shared or as fully partitioned. In fully shared memory, the entire memory is shared by all output ports. An incoming cell is dropped only when the memory becomes full. In fully partitioned memory an upper limit is imposed on the number of cells waiting in each output port queue, and when this limit is exceeded a cell is dropped, even though there may be space available in the memory. Full sharing allows efficient utilization of memory resources and achieves better cell loss probability, but suffers from fairness problems. Whenever a burst of cells arrives at a particular output port, reducing the memory space available for other ports.

Two important shared-memory switches are the Prelude switch developed by CNET [18] and Hitachi's shared-buffer switch [41]. In the prototype PRELUDE switch, the cell size is 16 bytes, of which 1 byte is header. The number of input and output ports is equal to the number of bytes in a cell (i.e., 16). In order to achieve high memory speeds, 16 memory modules are used such that each byte of a cell is stored in a different memory module.

A high-level view of Hitachi's shared-buffer architecture is shown in Figure 2.2. The memory is fully shared by the output ports. Output queues are formed using linked lists. The serial-to-parallel (S/P) modules perform serial to parallel conversion and the header conversion module (HD CNN) determines the link the cell should be put in. There are three types of circuitry: the switching chip which consists of the memory block, a multiplexer, and a demultiplexer, the control chip which consists of the read and write registers, one pair for each buffer, and input address buffers which keep track of the status of unused memory locations [51]. The write address register
is accessed to get a memory location for an incoming cell and the cell is written to the memory. The memory address is queued to the corresponding outgoing link as well. Simultaneously, a new empty buffer address is supplied from input address buffers to the control unit, if one is available. Similarly, during each time slot, one cell buffer is identified by the control unit read registers, retrieved, and transmitted by the outgoing link.

![Figure 2.2: Hitachi's Shared-Buffer Switch](image)

Memory switches are suitable either for small switches or as building blocks of larger switches. A switch of size $N$, where $N$ is the number of input/output ports, must be capable of sequentially processing $N$ incoming cells and selecting $N$ outgoing cells each cycle. As a result, a high memory bandwidth of the order of $2NV^t$ is required, where $V^t$ is the speed of the ports. To give an example, a 32 port switch running at 150 Mb/s requires a memory bandwidth of 9.6 Gb/s. The memory bandwidth becomes the bottleneck of shared-memory architectures.

### 2.3 Space-Division Architectures

Both shared-medium and shared-memory architectures suffer from two major drawbacks. First, multiplexing is required at the input side and demultiplexing is required at the output side of the switch, which means the switch does not scale well to support a large number of ports. Second, centralized control functions and buffer management increase the complexity of the switching nodes [51].

In space-division switches, multiple cells from different input ports are transferred concurrently on multiple output links. They also allow distributed control, and therefore reduce node design complexity. Space-division switches can be classified into two
important categories:

- Crossbar switches.

- Multi-stage switches

Both categories are explained in the following subsections.

2.3.1 Crossbar Switches

Crossbar switches have been attractive to switch designers because of their simple architectures, as shown in Figure 2.3. Every input can be connected to any non-busy output through a path that has only one switching element when connected. Cells at switch inputs reach their outputs with small constant delay.

![Crossbar Switches Diagram]

Figure 2.3: Crossbar Switches

These switches are nonblocking when handling permutation loads. But, when cells at two or more inputs contend for the same output port, output port blocking arises. Output port blocking can be relieved by adding queueing buffers at the input ports of the switch, at the crosspoints, or at the output ports. Figure 2.4 shows a crossbar switch with queueing at the crosspoints. The simple on/off switch of a crosspoint is replaced by a FIFO queue preceded by an address filter (AF). Input cells are sent directly into the matrix. Cells whose destination address match the address of the filter can pass through. A typical crossbar-based switch with concentration, that has become well known, is the Knockout switch [21, 63].
In general, an $N \times N$ crossbar switch uses $N^2$ crosspoints. The main disadvantage of crossbar switches is that switch complexity grows with $N^2$, of which at most $N$ are used at any time. As a result, crossbar switches become uneconomical for large switches. Also, since there is a unique path between any input and output port, the loss of a crosspoint will prevent connection of the two ports involved.

### 2.3.2 Multi-Stage Switches

Unlike crossbar switches, which are based on a matrix topology, multi-stage switches are based on a tree topology, as shown in Figure 2.5 for a $1 \times 8$ multistage binary tree.

![Figure 2.5: A 1 x 8 Multistage Binary Tree](image)

Multi-stage switches consist of a number of switching elements arranged in a number of stages. The input links connected to the switching elements in the first stage are called the input ports and the output links of the switching elements of
the last stage are called the output ports. Each input port is the root of a tree that branches over a number of intermediate switching elements and terminates at the leaves of the tree which become the output ports. Therefore, an $N \times N$ multi-stage switch generates a forest of $N$ trees sharing all the links and switching elements except the roots. The different ways of forming such trees results in different topologies of multi-stage switches.

The number of crosspoints in a multistage switch is less than a single-stage crossbar matrix. Each switching element can be a $n \times k$ crossbar. The most commonly used switching elements are $2 \times 2$ crossbars. A generalized 3 stage $N \times N$ multi-stage switch is shown in Figure 2.6, where $N$ is the number of input/output ports of the switch. If we assume the switching elements in the first stage to be of size $n \times k$, then there will be $(N/n)$ elements in the first stage. The second stage has an array of elements of size $(N/n \times N/n)$ and there are $k$ of them. The last stage has $N/n k \times n$ switching elements. For each connection between the source and the destination, there are $k$ different paths in the 3-stage interconnection network. Since the input ports of a multi-stage switch are connected to the output ports through a series of intermediate switching elements, they are sometimes referred to as indirect switches.

![Figure 2.6: A 3 Stage $N \times N$ Multistage Switch](image)

Many well-known interconnection networks such as Banyan, Delta, Omega, Flip, and Cube have been proposed as switching fabrics for high-performance cell switches. Banyan networks have the property that there is exactly one path from any input to any output. There are several classes of Banyan networks [49] out of which the regular, rectangular SW-banyans constructed from identical switching elements are of practical interest. Delta networks as defined by Patel are a subclass of Banyan
networks. A rectangular \( N \times N \) delta network, constructed from identical \( b \times b \) switching elements in \( k \) stages where \( N = b^k \), is called a Delta-\( b \) switch. Figure 2.7 shows an \( 8 \times 8 \) Delta-2 network.

![Figure 2.7: 8 x 8 Delta-2 Network](image)

All these networks have different interconnection patterns, but they perform much the same in a cell-switching environment [4]. Multi-stage switches are self-routing switches. Cells are routed from any input to any output using a \( k \)-digit, base \( b \) destination address. For example, consider the 3-stage Delta-2 network in Figure 2.7. Each switching element can connect its input and output links directly (i.e., top input link to top output link and bottom input link to bottom output link) or crossed (top input link to bottom output link and bottom input link to top output link). Suppose, the cell at input link number 5 (assuming the links are numbered from the top starting at 0) needs to be routed to the output destination number 1 (binary address 001). The destination address itself serves as a routing code for the cell. The switching element in the first stage connected to the input link number 5 decides on the current output link setting (direct or crossed) based on the first digit of the routing code, which is 0 in this case. Assuming that a bit 0 will connect to the upper link of the switching element and a bit 1 to the bottom link, the path of the cell from input link number 5 to output link number 1 will be as shown in the figure. This type of cell movement from input to the output of a switch is known as
**digit-controlled** routing or **self-routing**. The advantage of this type of routing is that each switching element makes decisions independently, based on a single bit in the routing tag. There is no need for centralized control. The figure also shows another cell at input link number 1 trying to go to output link 6.

While multistage switches are capable of switching cells simultaneously and in parallel, there may be blocking inside the switching fabric as a result of which cells collide with each other and are lost on their way to the output destinations. This type of blocking due to contention for a particular link inside the switch is called internal link blocking as shown in Figure 2.8(a). There is another form of blocking called output port blocking which arises when two or more cells contend for the same output port (Figure 2.8(b)).

![Internal Link Contention](image)

Figure 2.8: Types of Blocking: (a) Internal Link Blocking (b) Output Port Blocking

Blocking results in the reduction of the maximum throughput of the switch. Several suggestions have been made to reduce blocking and to increase the throughput of the switch:

- Place buffers in every switching element.

- Use a handshaking mechanism or a back-pressure mechanism between stages so that transfer of blocked cells may be delayed.

- Increase the speed of the internal links (i.e., links within the switch) compared to that of the external links (i.e., input or output links).
- Create multiple paths from any input to any output either by having multiple links for each switch connection or by using multiple switches in parallel.

- Distribute the network load evenly by using a distribution network in front of the cell switching network.

The idea of using buffers in multi-stage networks was first introduced and analyzed by Dias [20]. Turner extended the idea further to develop a general-purpose cell switching system - a buffered banyan switch [60]. Buffers can be placed at the input ports of the switch, output ports of the switch, or even inside the switch fabric. Input buffering results in a form of blocking called head-of-line (HOL) blocking which takes place when the blocked cell at the head of the queue prevents the subsequent cell from being delivered to the free output port. HOL blocking results in a reduction of switch throughput.

By providing buffering at the output port of the switch, output port contention can be resolved. Output buffering eliminates the HOL blocking present in the input buffering. Therefore, higher throughput is obtained with output buffering. However, output buffering increases the hardware complexity of the switch. The output buffer must be able to receive and manage cells arriving simultaneously. As the link and switching speeds increase, this becomes a difficult task and may constrain scalability of Banyan switches.

Contention inside the switch fabric, i.e., internal blocking can be resolved with internal buffering. Cells contending for the same output port of a switching element can be stored temporarily inside buffers provided at each internal link. Again, the buffers can be placed at the input or output ports of the switching element or in a shared memory, as shown in Figure 2.9 for a $2 \times 2$ switching element.

As discussed earlier, input buffering causes HOL blocking whereas output buffering requires more connectivity complexity. Shared memory eliminates HOL blocking and also is less complex than output buffering. However, shared memory can not be just simple FIFOs, and needs buffer management logic to be implemented inside the switching element, which adds complexity.
Figure 2.9: Three Types of Internal Buffering: (a) Input Buffering (b) Output Buffering and (c) Shared Memory

Although internal buffering reduces internal blocking it suffers from certain disadvantages. Internal buffers introduce delays within the switch fabric that may cause undesired cell delay variation. Also, internal buffers are most effective when the traffic is uniformly distributed at the output ports, which may not be the case most of the time.

Flow control between switch fabric elements using a back-pressure mechanism or a hand-shaking mechanism can also reduce internal link blocking. A back-pressure mechanism ensures that a cell can only leave its buffer if the corresponding destination buffer at the next stage is able to accept it. The next stage buffer will accept a cell only when the buffer is empty or if any cell in the next stage buffer can move forward. In switches using a back-pressure mechanism, control signals are passed from the last stage of the switch toward the first stage, so that every cell knows whether it should move forward one stage or remain in its current buffer.

By operating the switch fabric links at a speed higher than the speed of the input source, blocking can be reduced. If the switching fabric of an $N \times N$ switch runs $N$ times faster than the input cell arrival rate, queueing of cells at the input ports can be avoided, but there may be queueing at the output ports. If there are $N$ cells arriving at the $N$ input ports and all of them want to go to a common output, then they can enter the output buffers simultaneously. If the internal links joining the switching elements operate at, say, 8 times the speed of the external links, a 100% load on the external links will result in only 12.5 percent load on the internal links. However, as $N$ becomes larger, it becomes harder to achieve these speedups.

Internal blocking can also be avoided if the cells are first sorted according to their destination addresses and then switched. The Batcher-Banyan fabric is based on
this concept [30]. There are two segments in these switches: the first segment is a sorting network that sorts the incoming cells according to their destination addresses. The second segment is a banyan network which routes the sorted cells. One of the switch fabric implementations which is based on sort-banyan self-routing structure is the Starlite switch [30]. When cells with identical destination addresses arrive at the switch, output port blocking may still occur. Hui proposed another switch architecture [31] based on the Batcher-Banyan structure that overcame the output port contention problem.

Nonuniform traffic distribution may result in congestion and therefore, blocking in a switch. By using a distribution network in front of the switching network, congestion can be reduced. The distribution network distributes the incoming cells evenly throughout the succeeding switching network.

Yet another approach to reduce blocking in a multistage network is to provide multiple paths from any input to any output. Two strategies to have multiple paths in a switch without sacrificing much of the switch's structure were suggested by Kruskal and Snir [40]:

1. Replace each link within the switch by, say, $d$ distinct links as shown in Figure 2.10(a). This is defined as $d$-dilation of the switch. In a dilated switch a cell in a switching element can move to the next element using any of the $d$ links.

2. Replicate the switch itself, creating $d$ identical distinct copies as shown in Figure 2.10(b). This is called $d$-replication.

![Figure 2.10: Two-stage Banyan Networks with: (a) 2-Dilation (b) 2-Replication](image)
2.4 Distributed Switches

The very early theoretical work on interconnection networks was done for the purpose of interconnecting multiple processors and memories in parallel computer systems. Subsequently, several types of these networks have been proposed as telecommunication switches. One such example is Banyan switches, discussed earlier. Originally developed by Goke and Lipovsky. Banyan switches belong to the class of multi-stage interconnection networks. Interconnection networks proposed for use as a metropolitan area network (MAN) have also been suggested as distributed switch fabrics. Examples of distributed switch fabrics are Toroidal switches. ShuffleNet. and Shuffle Exchange switches. discussed in the following subsections. Binary hypercube switches which belong to the category of Toroidal switches are of special interest to us. We discuss this switch in detail in a separate section.

2.4.1 Toroidal Switches

Multi-stage switches are sometimes referred to as indirect switches because of the fact that the input ports of a multi-stage switch are connected to the output ports through a series of intermediate switching elements. A different class of switches, called direct switches. does not have intermediate stages between the switch inputs and outputs. Each switching element or node of the switch system has its own local source and sink. and is connected to other elements over a set of point-to-point links. Toroidal switches [53] and the Shuffle Network switches [39] (discussed in the next section) belong to the category of direct switching networks.

As in crossbars and multistage switches, the problem of output port contention may also arise in direct switches whenever more than one cell contends for a single outgoing link. To avoid losing cells the classical approach of placing buffers in the switching elements can be adopted. However, there is another option, called deflection or retrograding which is useful in direct switching networks. Deflection ensures that if - due to contention at a switching node - not all cells can be sent out along a shortest path, some cells are routed along non-shortest links. As a result, those cells travel a
greater distance before they reach their destinations.

Toroidal switches get their name from the fact that they can be embedded inside the surface of a torus (doughnut-type shape) [53]. When they are placed in a rectangular area most of the links map onto a rectangular grid layout. Some links are used to connect directly nodes on the left boundaries of the grid with nodes on the right boundaries and nodes on the top boundaries with those on the bottom as shown in Figure 2.11. These are called wrap-around links.

![Figure 2.11: Torus Network](image)

Most of the popular toroidal switches fall into the categories $k$-ary $n$-cubes, and $n$-dimensional meshes. A $k$-ary $n$-cube switch has $n$ dimensions, each having $k$ nodes connected by point to point links. For example, a 10-ary 2-cube switch will have a total of 100 switching elements connected by point-to-point links over the 2 dimensions, each having 10 elements per dimension. The main difference between a $k$-ary $n$-cube switch and a $n$-dimensional mesh is that the latter does not have wrap-around links. In $k$-ary $n$-cubes all the switching elements or nodes have an equal number of neighbors whereas in meshes the nodes on the boundary have fewer neighbors than other nodes, because of the absence of wraparound links. Figure 2.12 shows some $k$-ary $n$-cube topologies. A particular subclass of $k$-ary $n$-cubes called binary $n$-cubes or hypercubes is of particular interest to us and will be discussed in detail in later sections.

A toroidal network with unidirectional links known as the Manhattan Street Network (MSN) was proposed by Maxemchuk [43]. MSN initiated much of the current
interest in toroidal networks for communications. The links in adjacent rows and columns in the network alternate in direction. Each node has two input links (from two adjacent neighboring nodes) and two output links (to two different neighboring nodes). Therefore, at any instant each node may hold at most two incoming cells for which one or both output links may correspond to a shortest path to their destinations. A $4 \times 4$ MSN is shown in Figure 2.13. Nodes are indexed as $(i,j)$.

Certain structural characteristics of MSN make it an ideal candidate where deflection can be used to avoid output port contention problem. When there are two cells at the two incoming links of a switching node and both of them prefer the same output link, one cell is randomly chosen for the preferred link and the other cell is deflected to the non-preferred link. The penalty of deflection is an increase in the distance traveled by the other cell. The reward is that the cell need not be dropped or buffered.

2.4.2 Shuffle Network Switches

Another direct interconnection network topology that has received attention in terms of its suitability for deflection routing is the shuffle network [39]. An important characteristic of these networks is that the diameter (i.e., maximum value of the
shortest distance between any two nodes) of the network grows logarithmically with
the number of switching elements or nodes. The diameter of a MSN, by way of
comparison, grows as the square root of the number of nodes. One interesting class
of shuffle networks is known as shuffle-ring networks [39]. A shuffle-ring network
with parameters \( n \) and \( k \) consists of \( k \) columns of switching nodes, with \( 2^n \) nodes in
each stage. The nodes are assumed to be of size 2 \( \times \) 2. So, an \((n,k)\) shuffle-ring
network has a total of \( k \times 2^n \) nodes. The columns are arranged in a ring, with each
column connected to the next by a perfect shuffle interconnection. These networks
do not belong to the category of multistage networks because cells can be injected by,
or destined to, any of the \( k \times 2^n \) nodes in the network. When \( k = n \) in a \((n,k)\)
shuffle-ring network, it is called ShuffleNet [2, 29]. Figure 2.14 shows a ShuffleNet
where \( k \) is equal to 3.

An apparent drawback of using deflection in shuffle-ring networks is that a single
deflection can cause the distance of a cell from its destination to increase significantly
[39]. The penalty due to deflection in a ShuffleNet with \( k \times 2^k \) nodes is constant
and equal to \( k \). If a cell is within \( k \) hops of its destination in a ShuffleNet, the cell
has a preferred direction at every node on its path. But at nodes that are beyond \( k \)
hops from the destination, the cell has no preferred direction.
2.5 Direct Binary Hypercube Switches

As mentioned earlier, the family of direct binary n-cubes or hypercubes [55] form a subset of a larger class that includes k-ary n-cubes. Switches based on hypercube topology are promising for future B-ISDN applications. This thesis is concerned with the design and performance of these switches.

Several considerations were made in selecting the direct binary hypercube switches for our study:

1. In order to get better utilization of buffers and space in router implementations, the important design considerations that should be made are the number of buffers and the location of these buffers. Unlike multistage switches where the sources and destinations are connected to different switching elements connected through some intermediate elements, direct binary switches have their sources and sinks attached to the same switching elements and every node in the switch acts as an intermediate node for others. Therefore, a better utilization and sharing of buffers can be achieved by placing them at one place.

2. Direct binary hypercube switches provide multiple paths for routing a cell. Multiple paths help avoid performance penalties due to congested spots and faults.

3. In basic multistage switches where there is only one path from an input to
an output, the structure needs to be complete. We will see later direct binary hypercubes have the ability to route cells with the only available communication channels. The switching network need not be a complete cube in order to route cells, i.e., hypercube switches provide greater structural flexibility.

4. The diameter of a hypercube switch increases logarithmically with an increase in the number of switching nodes. This makes it attractive compared to other direct switches.

5. Because of its regular topology, deflection routing can be adopted in hypercube switches to avoid output port contention. This leads to a much simpler implementation than traditional store-and-forward cell-switching networks. The simpler implementation suggests that they can be used as high-performance high speed switching fabrics.

A \( d \)-dimensional direct binary hypercube switch has \( N = 2^d \) switching elements or nodes connected by \( d \times 2^{d-1} \) point to point links or edges. The elements in the cube are numbered from 0 to \( N - 1 \) and are represented in binary using \( d \) bits. Two nodes whose binary representations differ in exactly one bit position are connected by an edge. For example, in a 4-dimensional hypercube switch, nodes having binary representations 1001 and 1101 are connected by an edge. In a complete hypercube every node is directly connected to \( d \) other nodes. The graph of such a system is said to have degree \( d \). Figure 2.15 shows direct binary hypercube switches for various dimensions.

The hypercube switch can be built recursively: a \( (d + 1) \)-dimensional cube is created from two \( d \)-dimensional cubes, arranged side-by-side. An extra address bit is prepended to the binary addresses of the \( d \)-dimensional hypercube nodes: those in one half receive a leading 0 bit and those in the other half receive a leading 1 bit. The two nodes - one in each half - having the same least significant \( d \) address bits but differing most significant bit are connected. The degree of each node in the resulting hypercube increases by one.
Figure 2.15: Hypercubes for Various Dimensions: (a) Dimension 0. (b) Dimension 1. (c) Dimension 2. (d) Dimension 3. and (e) Dimension 4.

In general, each switching node in a $d$-cube can be viewed as consisting of a source or a local host. a sink or destination. and $d$ input-output line pairs as shown in Figure 2.16 below. An input-output line pair in the figure corresponds to a bidirectional link in the binary $d$-cube and the switch is connected to $d$ of its neighbors using $d$ such pairs.

![Diagram](image)

Figure 2.16: Single Switching Element of a Hypercube

We assume that the hypercube switches operate synchronously. i.e., they have a system-wide clock that permits all nodes in the cube to operate in uniform cycles. When a cell is injected by its local source or local host, the switching element attempts to route the cell along a shortest path to its destination. The shortest path of a cell is calculated by performing an exclusive-or (XOR) operation between the source and destination address of the cell. The result of this is the routing code of the cell, which is also $d$ bits long, one bit for each dimension of the hypercube. The routing code indicates the dimensions the cell must traverse to arrive at its destination. The tally
of ones in the *routing code* indicates the distance of the cell from its destination. The presence of a 1 in the *routing code*, in any position, say $i$, indicates that if the cell is routed on the $i$-th dimension of the hypercube to the node's $i$-th neighbor, it will move one step closer to its destination. When a cell traverses a communication link in one dimension, the corresponding bit in the *routing code* is inverted. A cell is normally routed in such a way that it is moved closer to its destination. However, it is possible for a cell to undergo *retrograde* motion or deflection and temporarily move away from its destination. When a cell reaches its destination, all the bits in the routing code are zero.

An important property of the direct binary hypercube switch is the multiplicity of paths for routing a cell. Multiple paths avoid congested spots and faults. If $x$ is the distance of a cell from its destination, then there are $x!$ shortest paths available for routing the cell. However, the number of disjoint shortest paths is only $x$.

The *distance* between two nodes of a hypercube is the minimum number of cycles needed for a cell to travel from one node to the other. This is equivalent to the number of differing bit positions between the two node addresses. The distance between any two nodes is an integer between 0 and $d$. There may be more than one node in a hypercube at a given distance from any given node. For example, in a dimension-3 hypercube distance between any two nodes is an integer between 0 and 3. From any node there are 3 nodes at a distance of 1. 3 nodes at a distance of 2, but only one node at a distance of 3. In general, in a $d$-dimension hypercube, the number of nodes at a distance $k$ from any given node is given by $\binom{d}{k}$. Since there is a total of $2^d$ nodes in the hypercube, the probability that there is a node at a distance $k$ from any other node is given by $\binom{d}{k}/2^d$.

### 2.6 Routing Algorithms for Direct Switches

A number of algorithms have been proposed for routing cells in hypercubes [1, 46], as well as in other direct switches like MSN's [43] and shuffle-nets [39]. Variations
in these algorithms are due to basic differences in certain physical or conceptual parameters of the switches. Some of these parameters that influence the routing algorithms are:

1. The topology of the switches.

2. The mode of operation of the switches. i.e., circuit switching or cell switching, synchronous or asynchronous.

3. The conflict resolution technique. i.e., when there are multiple cells at a switch contending for a single outgoing link or for a single delivery channel (in situations where the cells have already reached their destination), then the switch may exercise the following options to resolve the conflict:

(a) *queue* multiple cells at the output. if such a queueing buffer is available at the switch output or

(b) *route* or *deliver* only one cell and drop the remaining ones. thereby causing cell loss or

(c) *grant* the line to one of the contending cells and *deflect* or *misroute* the others to some other outgoing links or

(d) apply a combination of queueing and deflection together. whereby cells that could not be forwarded or delivered are first sent to the queue buffer and if the queue happens to be full the cells are retrograded.

4. The routing strategy. i.e., the strategy by which the queued cells, the incoming cells and the locally sourced cells are assigned outgoing links. Two of the many possible link assignment strategies are: *one-pass* and *two-pass* rules. In a *one-pass* rule, cells to be routed or delivered are considered randomly one at a time. For a cell which has not reached its destination, the switching element examines which outgoing links are on shortest paths to the cell’s destination. Those are the preferred links of the cell. If any of those preferred links is free, i.e., not already assigned to another cell, then the router selects randomly one such free
link and forwards the cell on this link. Otherwise, the cell is misrouted to one of
the free non-preferred links. if we assume that contention resolution technique
uses deflection only. The choice of a free non-preferred link is made at random
again. The cell is assigned an unsuitable outgoing link right away: it does not
wait until all other cells have examined the set of outgoing links. This greedy
assignment of an outgoing link may cause another cell which might have selected
the link to be deflected.

In the two-pass version, a cell that can not be assigned a useful outgoing link
and needs to be misrouted, waits until all other cells are examined and assigned
outgoing links: after all such cells have been granted their preferred links. only
the cells to be deflected are left over. The remaining outgoing links are then
allocated among these deflected cells in any order.

The incoming cells may be considered in various orders for assignment to the
outgoing links. They may be examined: (a) in random order, (b) in some
fixed order (the order of the dimensions of the cube), (c) in order of increasing
distance of the cells from their destinations, or (d) in order of decreasing
distance from their destinations. In case of fixed order routing, since cells that are
examined first are more likely to move forward, cells that come on low num-
bered dimensions of the cube are favored for successful assignment than those
on higher numbered dimensions. Similarly, cells closer to their destinations and
cells further from their destinations are favored in cases (c) and (d) respectively.
The nearest-first policy of cell routing may reduce congestion by quickly deliv-
ering the easily deliverable cells whereas the farthest-first advantage cells that
require the most time to deliver [46].

5. Fault tolerance aspects of the algorithm. Some algorithms suffer a major setback
in the presence of faults in the switching fabric while others survive and still
manage to route cells. of course, with degraded performance.

It is obvious that some of the parameters of routing algorithms may have pro-
nounced effects on the performance of the switches whereas some parameters may
not matter much. For example, a routing algorithm which does not employ either queueing or retrograding suffers from cell loss under medium to large loads. On the other hand, algorithms that employ queueing or retrograding avoid cell loss, but at the expense of increased latency in cell delivery.

We now describe in detail a particular routing algorithm known as the SCS routing algorithm developed by McCrosky [46]. The algorithm uses a combination of queueing and deflection together to allow a limited buffer size, to avoid loss of already accepted cells, and not to have a high latency in cell delivery.

2.6.1 The Saturated Constant Shuffle (SCS) Algorithm

The Saturated Constant Shuffle (SCS) routing algorithm is a synchronous store-and-forward distributed routing algorithm developed for cell-switched direct binary hypercube networks [47]. The algorithm uses a combination of queueing and retrograding in order to resolve contention among multiple forwardable and/or deliverable cells contending for limited output and/or delivery channels. Nodes in SCS switches are equipped with a single physical queue that is shared by all the cells that can not be forwarded or delivered to the sink. Cells stored in the queue are processed in a pseudo-FIFO manner, thereby avoiding the HOL blocking problem. If the number of cells to be delivered exceeds the number of delivery channels, then there is a contention for the delivery channels themselves and non-deliverable cells are sent to the shared queue for temporary storage. While routing cells for various outgoing links, cells in the queue are considered first, followed by the cells that came from the neighboring nodes, and then the cells freshly injected from the node’s local source. The SCS routing algorithm allows both one-pass and two-pass link assignment strategies with all the possible choices (random, dimension-ordered, nearest-first, farthest-first) of considering the cells for routing at a switching element. In this thesis we assume a one-pass link assignment strategy and nearest-first policy.

The following section describes the SCS routing algorithm.
The Algorithm

Before we present the SCS routing algorithm, it is necessary to introduce symbols and data structure that are used in the algorithm.

Consider a single node and its behavior in one time slot. Figure 4.1 below shows one such node. A node in a $d$-dimension hypercube has $d$ forward incoming links and $d$ forward outgoing links. Each of the incoming links has an associated forward-in buffer $(FI_{i}, 0 \leq i \leq d - 1)$ to hold the arriving cells. Similarly, each of the outgoing links has an associated buffer called a forward-out buffer $(FO_{i}, 0 \leq i \leq d - 1)$ to hold cells that are to be moved to the node's neighbor. The switch is a synchronous system. During each cell exchange cycle (CEC), all the switching elements exchange cells with their neighboring elements and with their local sources and sinks.

Figure 2.17: Details of a Single Switching Element of a Hypercube

Injection buffers (INJ) hold the cells generated locally at the switch. Delivery buffers (DEL) absorb the cells destined for the sink. There may be one or more injection and/or delivery buffers at a switch element. Each buffer can hold only one cell. The single physical queue buffer of the switching node is logically partitioned into two separate queue buffers called queue-out buffers $(QO_{j}, 1 \leq j \leq Q)$ and queue-in $(QI_{j}, 1 \leq j \leq Q)$ buffers, where $Q$ is the size of the queue buffer. At the beginning of each CEC, routing decisions are made: first for the cells in the queue-in buffers, next for the cells in the forward-in buffers and last for the cells in the injection buffers. Cells having a routing code of zero are delivered to the node's sink. But if there are more distance zero cells than the number of delivery channels, then some of
them can not be delivered. They are stored in the queue-out buffers of the switch, if
buffer space is available, or else are retrograded. Similarly, for cells having non-zero
routing code, if not all of them can be successfully forwarded, the unforwardable cells
are stored in the queue-out buffers or else are retrograded. At the end of the routing
cycle, cells stored in queue-out buffers are transferred to the queue-in buffers of the
switch for routing in the next cycle. As a result, the number and the distances of the
cells stored in the queue-in buffers in the next cycle become identical to those stored
in the queue-out buffers in the previous routing cycle.

The symbols \( I_N \), \( J_w \), \( DEL_w \), \( source_w \) and the \( sink_w \) represent the \( w \)-th Injection
buffer, the \( w \)-th Delivery buffer, the \( w \)-th source and the \( w \)-th sink respectively. The
following functions and symbols are also used to describe the algorithm.

1. \( R \): represents a hypercube switch having \( 2^d \) switching elements, where \( d \) is the
dimension of the cube.

2. \( R_r \): represents the \( r \)-th switching element of a switch \( R \).

3. \( R_r.FI_j \): represents the \( j \)-th forward-in buffer \( FI_j \) of the \( r \)-th element of a
switch \( R \).

4. \( connected(r,i) \): is a function that returns the number of the \( i \)-th neighbor of a
switching element \( r \). If \( R \) is a hypercube switch, then \( R_{connected(r,i)}.FO_i \)
represents the \( i \)-th forward-out buffer of a switching element which is
connected to element \( r \) on its \( i \)-th dimension.

5. \( home(X) \): is a function that returns true, if the current node is the destination
node of a cell stored in buffer \( X \).

6. \( useful.direction(i, X) \): returns true if the \( i \)-th forward-out buffer \( FO \), offers
forward motion to the cell stored in buffer \( X \).

7. \( W \): total number of source channels connected to each switching element.

8. \( D \): total number of sink channels connected to each switching element.
9. \( Q \): total number of queue buffers at each switching element.

10. \( X.V \): is a boolean variable which is true if the buffer \( X \) contains a valid cell. The basic SCS routing algorithm is now given. The parallel and sequential actions of the algorithm are represented by the terms parallel and sequential.

**Clear all buffers.**

```
parallel for r = 0 to 2^{D-1} do
    sequential loop forever
        parallel
            parallel for i = 0 to d - 1 do \( R_r.FI_i \leftarrow R_{\text{connected}(r,i)}.FO_i \)
            parallel for q = 0 to Q - 1 do \( R_r.QI_q \leftarrow R_r.QO_q \)
            parallel for w = 0 to W - 1 do \( R_r.IN.J_w \leftarrow \text{source}_w \)
            parallel for k = 0 to D - 1 do \( \text{sink}_k \leftarrow DEL_k \)
        sequential
            parallel for i = 0 to d - 1 do \( FO_i \leftarrow \text{nil} \)
            parallel for q = 0 to Q - 1 do \( QO_q \leftarrow \text{nil} \)
            sequential for q = 0 to Q - 1 do assign(r, QI_q)
            sequential for i = 0 to d - 1 do assign(r, FI_i)
            sequential for w = 0 to W - 1 do assign(r, IN.J_w)
```

The function \( \text{assign}(r, X) \) makes routing decisions regarding a cell stored in buffer \( X \):

At the router \( r \), do the following

**If Home(\( X \)) and not DEL then**

\( \text{DEL} \leftarrow X \)

return

**For** \( i = 0 \) to \( d - 1 \) do

**If useful_direction(\( i, X \)) and not FO_i,1' then**

\( FO_i \leftarrow X \)

return

**For** \( q = 0 \) to \( Q - 1 \) do
If not $QO_q.1$ then
$QO_q \leftarrow X$
return
For $j = 0$ to $d - 1$ do
If not $FO_i.1$ then
$FO_i \leftarrow X$
return

The SCS routing algorithm never drops an injected cell that has been accepted into the network. Cells can be rejected at the time of injection and hence lost; but the source of the cell can be notified immediately. Rejection of injected cells arises when a switch has accepted enough cells such that all its input buffers, i.e., $d$ forward-in buffers, $Q$ queue-in buffers, and $W'$ injection buffers of a router are full, and all these cells need to go to the same forward-out buffer. After processing all the queued-in and the forward-in cells, the algorithm considers the fresh cells for routing. If there is no space available in the queue-out buffer or in the forward-out buffer, the fresh cells are rejected.

The algorithm can handle traffic priorities. The algorithm is flexible enough to manage any number of priorities of cells. $C_0$, $C_1$, $C_2$, $\cdots$, $C_{n-1}$, where $n$ is the number of priorities. An extension of the basic SCS routing algorithm that can handle prioritized cells is given in [33]. However, we restrict ourselves in this thesis to single priority.

2.6.2 Out-of-Orderness

Even though direct binary hypercube switches running the SCS routing algorithm are promising for B-ISDN applications, they suffer from one disadvantage. The cells belonging to a particular traffic stream may not be delivered at their destinations in sequence. This phenomenon, known as out-of-orderness of cells, happens because of the presence of multiple paths. queueing buffers, and retrograde motions of the cells in the switch. One of the performance objectives of the SCS routing algorithm is to
try to deliver cells to their destinations in the same sequence as they were accepted from their sources. The number of in-sequence arrivals of cells at their destinations should be maximized. In other words, the degree of out-of-orderness of cells should be minimized. A modeling study of this out-of-orderness phenomenon is one of the goals of this thesis work. Chapter 7 discusses the issue further.

2.7 Summary

In this chapter we have reviewed a number of proposed architectures for high-performance cell switches. Particular interest has been shown in a family of switches, called direct binary $n$-cube switches that belong to the more general class of toroidal switches. Some of the physical and/or conceptual parameters that influence the various routing algorithms running in direct switches have been presented. Finally, one particular algorithm - the SCS routing algorithm that runs on hypercube switches - has been discussed in detail.
Chapter 3

Switch Models

Performance analysis is a crucial step in the design and development of switching systems. A variety of techniques ranging from intuition to experimental evaluation are used to study the various design issues [19]. Intuitive techniques are fast and flexible, but less reliable. Experimental techniques are more accurate but expensive, and inflexible. Between the two extremes, there is another approach called modeling.

A model is an abstraction of a system that eliminates details not essential for understanding the behavior of the system. The two most common techniques for modeling systems are analysis and simulation. An analytical model attempts to build equations that relate system parameters to required performance measures. The solution of these equations may capture both transient and steady-state behavior of systems. Quite often a number of simplifying assumptions are made to make the model mathematically tractable. Sometimes, a single system resource (e.g., a single switching element) is represented in substantial detail to predict the behavior of the entire system (e.g., the entire switch system). In order to produce fast but approximate results, sometimes heuristics are used in the analytical models of switches. Problems with analytical models are the simplifying assumptions that make the solution tractable. The traffic models used in most of the analytic solutions are also unrealistic.

A simulation model is used when an analytical model becomes intractable or detailed modeling is required to understand the system’s behavior. Simulation models
are more general and flexible, but are complex to build and require very long run times to achieve the desired accuracy.

One of the popular techniques for solving an analytical model is numerical analysis, which is mostly restricted to Markovian models. Numerical analysis follows a three step procedure: (1) generation of the states of the underlying Markov process. (2) generation of the transition rate matrix $Q$ that describes the stochastic behaviour of the process in the steady state. and (3) numerical solution of the linear systems of equations $\pi Q = 0$ and $\pi e = 1$, where $\pi$ is the steady-state probability vector of states and $e = (1, 1, \ldots, 1)^T$ [51]. Numerical analysis is restricted to small switches in order to avoid explosion of the state space.

Any systematic performance study needs at least two key components: selection of performance metrics, and selection of a load model. For example, in a circuit-switched network, the quality of service measured as blocking probability is an important performance criterion. On the other hand, in a cell-switched network, delay is used as one of the performance measures. Delay of a cell is defined as the time difference between the generation of the cell at its source and delivery of the cell at its destination. In the case of cell switches, some of the performance figures that have been of interest are steady-state throughput, average cell delay, longest cell delay, link utilization, loss probability, and blocking probability. Steady-state throughput of a switch is defined as the number of cells that are delivered under equilibrium conditions at the destination ports of a switch per unit time. The maximum achievable throughput under ideal workload conditions is called the bandwidth of the switch. The link utilization measures the fraction of the link's capacity used to carry traffic. If it is assumed that a link can carry at most one cell per cycle, then the link utilization is given by the probability with which a cell is transmitted on the link. In the case of switches, where there are multiple cells contending for some limited resources like transmission links, some of the cells cannot be allocated their desired resources and are said to be blocked. Blocking probability measures the probability with which a particular cell is blocked. Depending on the switch architecture and the routing algorithm running in the switch, blocked cells can be temporarily buffered and/or misrouted. If not,
blocked cells are lost. The blocking probability becomes the loss probability when there are no provisions in the switch to buffer them or misroute them. The injection loss probability is the probability with which new cells from the source that are ready for injection, are not accepted and hence lost. Switches that use queueing buffers and/or deflection also choose probability of deflection and average queue occupancy as two other important performance metrics.

In case of direct switches, the load or traffic model refers to the injection traffic model, i.e., the model that describes cells generated by a source. A load model refers to the probability with which a new cell or cells are generated at the local source in each cycle, and the probability with which a new cell will have a particular destination. For example, a simple Bernoulli injection model $\text{Ber}(p)$ indicates that the probability with which a new cell is injected in each cycle is $p$, where $0 \leq p \leq 1$. A uniform traffic assumption indicates that new cells may be generated for any of the possible destination nodes (may include itself) with equal likelihood. If there are $N$ nodes in a switch, then the probability with which any particular node is selected as the destination of a newly generated cell, in case of uniform traffic assumption, is $\frac{1}{N}$ (including itself). In case of non-uniform traffic, the destinations of cells are selected with some arbitrary general probability distributions.

A load model may also allow injection of multiple cells at a single node. For example, in a $d$ dimension hypercube, as many as $d$ cells may be injected during any slot. A Binomial injection model $\text{B}(n, p)$ with parameters $n \geq 1$ and $0 \leq p \leq 1$ allows from 0 to $n$ cells to be injected where $p$ is the probability parameter.

One other key parameter that needs to be specified before a performance study is made is the number of delivery channels in a switching element or node, i.e., how many cells are allowed to be delivered to the sink or the destination per unit time. We may allow only one cell to be delivered per unit time in which case the switch follows a single-accepting scheme. In a multiple-accepting scheme multiple cells are allowed to be delivered to the sink in a single cycle. In most switch models, it is assumed that all deliverable cells are absorbed at its destination. If we relax this assumption, i.e., impose restrictions on the number of delivery channels, it can be
shown that the performance of the switches are significantly affected. Intuitively, in cases where the single-accepting scheme is followed, contention will arise among the multiple deliverable cells and to avoid cell loss, these cells will have to be assigned resources like queueing buffers or transmission channels.

We now present some switch models developed to analyze the different switch architectures discussed in the previous chapter. We start with models of shared-medium switches in Section 3.1 and cover models of shared-memory and space-division switches in Sections 3.2 and 3.3 respectively. Only the basic concepts are discussed in these models. Since, this thesis is concerned with the design and performance of a particular class of switches called direct binary \( n \)-cubes or hypercubes, which belong to the more general category of toroidal switches. Special emphasis is given on some models developed for hypercube switches as well as for other relevant toroidal switches like MSN. In the process, we motivate the development of a new analytical model for buffered binary hypercube switches that employ deflection. The new model is discussed in the next chapter.

### 3.1 Models of Shared-Medium Switch Architectures

Most of the analyses of shared-medium switches study switching modules in isolation. The performance parameters of interest are cell loss probability and delay.

Cells arriving from incoming links of a module are multiplexed into a common medium, such as a bus or a ring, and then demultiplexed into the FIFOs of outgoing links. The models of shared-medium switches depend on whether it is blocking or non-blocking. If the bandwidth of the switch is greater than the sum of the bandwidths of the individual incoming links, then the switch is non-blocking. By employing small input buffers with a capacity to hold only a few cells, cell losses at the incoming ports of a non-blocking switch can be avoided. However, if the switch bandwidth is less than the sum of the incoming link bandwidths, the switch becomes blocking
and relatively large input buffers are needed to meet cell loss QoS requirement. As far as output buffers are concerned, they should be dimensioned to avoid momentary overload situations. Cells are lost if they arrive at a time when there is no space available at the output buffer.

The model of a non-blocking shared-medium switching element is shown in Figure 3.1. The small input buffers are not explicitly modelled because of their insignificant contribution to the performance parameters of interest. As shown in the figure, \( N \) streams of incoming cells are demultiplexed into \( N \) output queues. The traffic on an incoming link consists of multiplexed cell streams generated by different sources in the network. The demultiplexing of traffic from an incoming link to a particular output port is called the thinning process. The arrival of streams at a particular output port is the superposition of individual thinned arrival streams from different incoming links. One way to model the thinned processes from an arrival stream is by assigning probabilistically the destination ports of the cells. For example, if the destination ports of consecutive cells in a multiplexed stream are independent, then the probability that a cell at incoming link \( i \) is destined for output port \( j \) is given by \( p_{ij} = m_j / m \), where \( m \) and \( m_j \) denote total average cell arrival rate at a link and the average cell rate of connections on that link destined for output port \( j \).

![Diagram of Nonblocking Shared-Medium Switch](image)

Figure 3.1: Model of a Nonblocking Shared-Medium Switch

Once the thinned processes are characterized, each output queue can be analyzed numerically independent of other queues with \( N \) arrival streams and a deterministic service time corresponding to the cell transmission time (53 \( \times \) 8 / linkspeed), where 53 bytes is the size of an ATM cell.
In models of blocking shared-media switches, the significance of buffers at the input ports are no longer negligible, because cells may be lost due to lack of space in buffers both at the input ports and at the output ports. A queueing model of a blocking shared-medium switch is shown in the following Figure 3.2. There has been extensive study in the literature for multiqueue systems served by a single server. A round-robin type of service policy can be adopted to serve the cells at input buffers, in which each input buffer is served in a cyclic manner. Three different types of service disciplines may be considered: (1) exhaustive, (2) gated, and (3) limited. In exhaustive service, each time a queue is visited, all customers in the queue including the ones that arrive during the service, are served whereas in the gated service, only the customers that were waiting when the server visits the queue are served. In limited-service discipline, a prespecified number of customers are served.

![Figure 3.2: Model of a Blocking Shared-Medium Switching Element](image)

If the nature of arriving traffic is bursty, then the techniques used to analyze polling systems cannot be directly applied to study the models of blocking shared-buffer switches. An approach to approximately analyze the input buffers, based on the framework given in [37], is described in [51].

### 3.2 Models of Shared-Memory Switch Architectures

In a shared-memory switch, cells arriving at incoming links are stored in the memory. The destination addresses of the cells are decoded and the memory locations at which the cells are stored, are linked to the output queues associated with the output ports. The queueing model of an $N \times N$ shared-memory switch therefore consists of $N$ single-
server queues corresponding to the logical queues associated with the output ports.
The model is quite similar to that used for non-blocking shared-medium switches shown in Figure 3.1, except for the fact that the total number of cells waiting in the different queues is bounded by the memory size $M$. Each logical queue has a finite capacity $B_i$. The individual queue sizes are less than or equal to the memory size $M$, i.e., $B_i \leq M$. When $B_i = M$, we have a fully shared memory.

An output port corresponds to the server at each queue. Time is slotted and the service time is equal to one cycle. The service times at output ports are synchronized, i.e., one cell from each output port may depart from the switch during one cycle. The cell arrival times at the incoming links are also slotted. An independent Bernoulli process can be used to characterize the arrival process.

Approximate models of shared-memory switch architectures are based on the analysis of a single queue with geometric inter-arrival times, deterministic service times, and infinite queue sizes, referred to as the $Geo/D/1$ queue. The switch itself is modeled with $N$ identical $Geo/D/1$ queues. The queue length distribution of the shared-memory is obtained as the $N$-fold convolution of the probability distribution of individual queues [28].

Recently, Yamashita et al. proposed another approximate algorithm [62] to analyze the queueing system of a shared-memory switch, in which the shared buffer is decomposed into individual queues and each queue is analyzed in isolation. Figure 3.3 shows the queueing structure of subsystem $i$ in isolation, which consists of queue $i$ and the subsystem $N - i$. The subsystem $N - i$ models the dynamics of the remaining $N - 1$ queues and is important to capture the effects of upper bound $M$. An arrival stream on incoming link $j$ is decomposed and routed to the corresponding output queues. If $b_{ji}$ is probability that a cell stream on incoming link $j$ after decomposition is routed to the output queue $i$, then $1 - b_{ji}$ is the probability that cell streams are directed to output queues other than $i$. Once we have the decomposition process repeated for each $j$ with $b_{ji} > 0$, we have up to $N$ arrival streams of cells to queue $i$ and a total of $2N$ arrival streams for both queue $i$ and subsystem $N - i$.

Now, as far as the departure process is concerned, one cell departs from the $i$-
th queue each cycle. The subsystem $N - i$ corresponds to all queues other than $i$. The total number of cells in subsystem $N - i$ is known, but how these cells are distributed in individual queues is not known. This may make a difference: because at one extreme all the cells may be in the same queue in which case only one cell can depart in one cycle. The other extreme: all these cells may be uniformly distributed over these $N - 1$ queues, and as many as $N - 1$, depending on total number of cells in subsystem $N - i$ may depart each cycle. So it is necessary to estimate the number of busy queues ($k$) in subsystem $N - i$, when there are $n$ cells in the system and $n_i$ cells in queue $i$. The probability distribution can be obtained iteratively from the solution of $N$ subsystems in isolation with each subsystem consisting of queue $i$ and subsystem $N - i$, $i = 1, \ldots, N$. Yamashita assumed that this distribution depended only on $n - n_i$ (i.e., $p_i(n - n_i, k)$), and not on both $n$ and $n_i$.

3.3 Models of Space-Division Switch Architectures

A space-division switch can be modelled as a collection of a large number of single queueing stations. The output port of each switching element acts as a server. The queueing structure of servers depends on the type of buffers used in the switching elements: input buffers, output buffers, and/or shared-memory buffers. Queueing network models of a $4 \times 4$ Banyan network consisting of $2 \times 2$ switching elements are shown in Figure 3.4. All the models have both input and output buffers.
Figure 3.4: Queueing Network Models of a $4 \times 4$ Banyan Network with I/O Buffers: (a) $4 \times 4$ Banyan Network; (b) Input Buffering; (c) Output Buffering; (d) Input/Output Buffering

The service time at each server is slotted and takes one slot time. Arrivals of cell streams at the input ports are also slotted. Cells arriving at the input ports go to their output ports by passing through a number of switching elements, one at each stage. After completion of service at stage $j$, a cell at switching element $s_j$ attempts to enter the next stage switching element $s_{j-1}$ along its path. If there is no space available in $s_{j-1}$, the cell at $s_j$ is blocked. There are two different ways blocked cells can be handled: (1) drop the blocked cell, thereby eliminating HOL blocking, or (2) force the blocked cell to wait another cycle before it attempts to enter $s_{j-1}$ again. The second type of blocking is referred to as repetitive service.

Queueing models of Banyan switches are, in general, more complex than shared-medium or shared-memory switches. Switches with only a small number of ports have exact numerical solution. Simulation models are complex to build and require very long run times to obtain accurate estimates of performance measures. Initial studies considered only unbuffered switches with identical Bernoulli input sources and uniform destinations. Later models considered switches with single buffers as well as multiple buffers. In the later case, FIFO buffers were considered either at the input ports or at the output ports, but not at both. Based on the assumption of the Bernoulli arrival process, models with complex buffering schemes and non-uniform
traffic distributions have also been reported [38].

One approach to solve the queueing network of the switch is to decompose the network into individual subsystems, and then analyze each subsystem in isolation. We have seen this decomposition method in connection with analysis of shared-memory switches. This needs a definition of the subsystem and also an estimation of the arrival and effective service processes operating on the subsystem. For example, if the number of ports in a switching element is not large, each switching element itself can be considered as a subsystem. It is also possible to further decompose each switching element into its individual queues and then analyze each queue in isolation.

The arrival process at each subsystem is derived from the departure processes of its upstream subsystems. The effective service process is determined by the way blocked cells at any switching element are handled. If the blocked cells are allowed to be lost, then whether there is space available in the downstream queues or not, the cells at the previous stage queues receive service and depart. In the next cycle, the server starts serving another cell. If there is one waiting in the queue. But if repetitive service is assumed, then there is a complex relationship between a server and its upstream queues. For blocked cells, it is necessary to approximate the probability distribution function of the number of cycles it takes for the blocked cell to depart from the server [51].

3.4 Models of Distributed Switches

Performance models of distributed switches having regular patterned topology, like MSN and Hypercubes, are discussed in this section. The switching elements of a distributed high-speed switch should support simple local routing decisions. One such decentralized routing scheme, mentioned earlier, and particularly suitable for distributed switches, is deflection routing. The analysis of switches employing deflection routing is difficult, because a cell can have an unbounded number of distinct routes from its source to its destination. Also, the probability that a cell is deflected depends, in the case of unbuffered deflection routing, on the probability that it conflicts
with other cells that might themselves have been deflected [7].

In order to make models tractable, switches have been analyzed under uniform traffic assumptions. Approximate analysis of deflection routing studying unbuffered MSN networks was carried out by Greenberg and Goodman [25] under uniform traffic assumptions. Performance evaluations of multi-buffered MSN with deflection routing and with uniform traffic load are reported in [16]. A number of models have also been developed for hypercubes [1. 24. 57]. Some of these models study both buffered and unbuffered hypercubes that do not employ deflection routing and some consider deflection routing, but no buffers. The following subsections provide brief discussions of distributed switch models of MSNs and Hypercubes.

3.4.1 Greenberg and Goodman's MSN model

Greenberg and Goodman did a performance analysis [25] of Manhattan Street Networks (MSN), a two-dimensional directed mesh as shown in Figure 3.5

![Figure 3.5: 4 x 4 MSN Network](image)

As shown, the MSN is a regular 2-connected network, having both indegree and outdegree (i.e., the number of input links and output links of any node) equal to 2. There are \( n^2 \) nodes in an \( n \times n \) MSN. Each node operates in a discrete, time slotted fashion and attempts to route cells on shortest paths to their destinations. At any time \( t = 0, 1, 2, \ldots \), a given node may receive up to two cells from its neighbors.
one on each incoming link. Also, a new cell may be injected from the node's local source. If a node receives two continuing cells, i.e., cells that are not to be delivered from two of its neighbors and also, a newly generated cell from its own local source, then the new cell is blocked. It is rejected and cleared. A conflict arises if the shortest paths of the two continuing cells demand the same outgoing link. Greenberg and Goodman proposed deflection routing to resolve conflicts in which one cell is selected for transmission on the desired outgoing link and the other cell is misrouted or deflected on the other outgoing link. The injection traffic model assumes that each node generates a single new cell independently with a fixed probability. The new cells' destinations are chosen independently at random from a uniform distribution of the other \(n^2 - 1\) nodes (excluding its own source).

The two important network characteristics of a MSN employing deflection are: (1) the diameter, and (2) the deflection index of the network. MSN has the disadvantage that it has a relatively large diameter equal to \(n = \sqrt{N}\), where \(n^2 = N\) is the total number of nodes in the MSN. An advantage of MSN is that its deflection index is just 4. The deflection index of a network is defined as the average number of hops that a single deflection adds to a cell's delay. Therefore, the delay of a cell can be defined as the length of a shortest path between its source and destination plus the product of the deflection index and the number of deflections experienced en route [25].

Figure 3.6: MSN Grid

Formulation of an exact Markov chain model to compute performance of a \(n \times n\) MSN network will have at least \(O(n^3)\) states. This is intractable even for modest
\( n \) and it is not possible to compute performance statistics using an exact Markov model. Two approximate performance models that give estimates of the steady-state throughput and the average cell delay for cells admitted to the network were developed. These new models are called the \textit{one node} model and the \textit{one cell} model [25]. We discuss each of these models in the following subsections.

### 3.4.2 One Node Model

Let the row and column indices of a \( n \times n \) Manhattan Street Network be numbered from 0 to \( n - 1 \). Even numbered rows point east and odd numbered rows point west as shown in the Figure 3.6. Similarly, even numbered columns point south and odd numbered columns point north.

During any given cycle \( t \), a node \((i, j)\), \(0 \leq i, j \leq n - 1\), receives up to two cells, one from each incoming link. If a cell is destined for the node \((i, j)\), it is absorbed. If not, a cell destined for another node may have the following routing preferences:

- \textit{don't care} if both the outgoing links of node \((i, j)\) lie on shortest paths to the cell's destination.

- \textit{row}: if only the outgoing row link lies on a shortest path

- \textit{column} if only the outgoing column link lies on a shortest path

If a cell's routing preference is \textit{don't care} then at time \((t + 1)\) the cell is transmitted on either outgoing row or outgoing column link - the choice is decided by a fair coin toss. Otherwise, the cell is assigned its preferred outgoing link. When there are two \textit{don't care} cells at a node \((i, j)\), a fair coin toss pairs the two cells to the two links. If one cell is \textit{don't care} and the other has a preference, then the latter is transmitted first on its preferred link and the \textit{don't care} cell is transmitted on the other link. Conflict arises when two cells have identical preferences. Greenberg and Goodman consider two conflict resolution rules: (1) \textit{random} in which a fair coin toss pairs the two cells to the two links and the losing cell is deflected, and (2) \textit{straight-through} in which the cell that arrives on the row incoming link is transmitted on the outgoing row link and the
cell that arrives on the column incoming link is transmitted on the outgoing column link. They mentioned a third option closest to finish in which the cell closer to its destination is given the preferred link, but did not consider it in their performance analysis. A cell attempts a critical bend when its routing preference needs a change in direction, i.e., if a cell enters a node on the column input and its routing preference is the row output or if a cell enters a row input and its routing preference is the column output of the node. There is a shortest path from any incoming edge of any node \((i, j)\) to any other node \((u, v)\) that contains at most three bends [25].

The analytical model developed to compute network throughput and mean cell delay is based on two major assumptions: (1) relabeling invariance and (2) independence. The idea behind the relabeling invariance assumption is that at all times \(t\) and for all \(i, j, u, v\) \((0 \leq i, j, u, v \leq n - 1)\) the probability that node \((i, j)\) generates a new cell destined for node \((u, v)\) at time \(t\) is the same as the probability that node \((0, 0)\) generates a new cell destined to node \(A_{i,j}(u, v)\) at time \(t\). \(A_{i,j}\) is called the automorphism used for relabeling of the mesh. It is defined as follows:

For all \(i\) and \(j\) \((0 \leq i, j \leq n - 1)\), one can define the automorphism \(A_{i,j}\) as a relabeling of the mesh, putting \((i, j)\) at the origin: \(A_{i,j}(u, v) = (w, x)\) where

\[
w = \begin{cases} 
(u - i) \mod n & \text{if } j \text{ is even} \\
(i - u) \mod n & \text{if } j \text{ is odd} 
\end{cases}
\]

\[
x = \begin{cases} 
(v - j) \mod n & \text{if } i \text{ is even} \\
(j - v) \mod n & \text{if } i \text{ is odd} 
\end{cases}
\]

Thus, row \(i\) (column \(j\)) is renumbered 0, and the other rows (columns) are renumbered from 1 through \(n - 1\). scanning from row \(i\) (column \(j\)) in the direction of node \((i, j)\)'s outgoing row (column) link.

Based on the above relabeling invariance assumption, we can focus on only one base node \((0, 0)\) and refer to a cell destined to a node \((i, j)\) as cell \((i, j)\). The independence assumption is defined as: for all \(u, v, w, x\) \((0 \leq u, v, w, x \leq n - 1)\) and all
time \( t \geq 0 \), the probability that at time \( t \) node \((0,0)\) receives cell \((u,v)\) on its row input and cell \((w,x)\) on its column link is the product of the probabilities of these two events.

Let

\[
    r_t(i,j) = \Pr\{ \text{node (0,0) transmits cell (i,j) on its row output link at time } t \}
\]

and

\[
    c_t(i,j) = \Pr\{ \text{node (0,0) transmits cell (i,j) on its column output link at time } t \}
\]

Greenberg and Goodman developed a set of difference equations relating the probability distributions of cells that a node transmits at time \( t + 1 \) \((r_{t+1}(i,j) \text{ and } c_{t+1}(i,j))\) in terms of the probability distribution of cells that a node transmits at time \( t \). The fixed point solution of the set of equations provides estimates of the steady-state values of these distributions, from which mean throughput and mean delay are computed.

### 3.4.3 One Cell Model

The **one cell** model was used to estimate the steady-state throughput of the network. In this model a single cell performs a random walk in the network. At each node en route to the cell's destination the cell is either deflected with some probability \( p \) or is routed correctly with probability \( 1 - p \). From an analysis of this random walk, one can obtain the link utilization \( u \) and the probability of deflection \( p \) as a function of \( u \) and \( p \). As in the one node model, fixed point solutions of these two parameters \( u_\ast \) and \( p_\ast \) are obtained from which the steady-state throughput of the network is derived.

The **one cell model** is simpler than the **one node model**. Also, the **one node model** cannot analyze the performance of very large networks having as many as \( 10^4 \) nodes, which the **one cell model** can do. Both the models provide estimates of the throughput of a MSN with an accuracy of 1% for the full range of loads.

Associating buffers with a node's two outgoing links may result in improvement
of throughput and delay figures. In that case when a conflict occurs, and the corresponding buffer is not full, a cell can be enqueued in the buffer rather than deflected onto another link. However, the authors claim (without proof) that buffers having sizes larger than the deflection index of MSN (which is 4) may not achieve further improvement, primarily because large buffers may hurt. The argument given is that use of deflection defuses hot spots in the links that may sometimes arise under heavy loads, but use of large buffers may result in less deflection and hence make the algorithm less adaptive to defuse hot-spots [25].

3.4.4 Choudhury and Li’s Model of 2-connected Networks

An approximate analysis of deflection routing in an unbuffered Manhattan Street Network, under uniform traffic model, was carried out by Greenberg and Goodman. It was suggested there that adding just a few buffers would provide significant improvement in the throughput and delay performance of the MSN over that of the unbuffered network. A simulation study was also carried out to study the effect of adding buffers in MSN [16]. But no analytic models were developed to study buffered deflection routing in different regular topology networks. Choudhury and Li develop an analytic model based on the model developed by Greenberg and Hajek [24] to study the performance of buffered deflection routing in any regular network under a uniform traffic model. They choose a toroidal network MSN, and shuffle-based networks like single-stage Shuffle Exchange Network and the multistage ShuffleNet as the candidates for developing analytical models.

All these networks are two-connected, i.e., each node in the network has two incoming and two outgoing links. The networks operate synchronously. At any node, at the beginning of a cycle, there are \( U \) transit cells received from other nodes during the previous cycle and \( V \) new cells generated locally. Since at most one cell arrives per incoming link, \( U \leq 2 \). Of the \((U+V)\) cells present, \((U+V-2)\). where the notation \( x^- \) means \( \max\{x,0\} \). are blocked. Since, the transit cells have priority over the new cells generated at the node, \( \min\{V,(2-U)\} \) of the new cells are accepted. If two cells
want the same output link. contention is resolved by tossing a fair coin. Unlike in unbuffered deflection routing where the cell that loses contention is deflected towards its non-preferred direction, in buffered deflection routing, the losing cell is placed in the buffer associated with its preferred link behind the winning cell, provided buffer space is available. Only if the buffer associated with the preferred link is full is the losing cell deflected. The main difference between unbuffered deflection routing and buffered deflection routing is that the losing cell in the former case does not get the preferred link, but in the later case, it gets the preferred link but is placed in the buffer behind the winning cell (provided buffer space is available). A uniform traffic model is assumed. The number of new cells generated per unit time, by the local source, is assumed to follow a Bernoulli distribution $Ber(p)$, where $p$ is the probability $0 \leq p \leq 1$. If $N$ is the total number of nodes in a network and $q(i)$ is the probability that a new cell has its destination $i$ hops away, $1 \leq i \leq d$ (d is the diameter of the network), then $(N-1) \times q(i)$ will be the average number of nodes placed at a distance $i$ from the source. The network topology decides the distribution $q(i)$, and also the average increase in path length of a cell due to a single deflection.

Choudhury also assumes that because of the regularity of the network and the uniform traffic assumptions a single node model can be used to draw conclusions about the entire network. The structure of a single node to study the model is as shown in Figure 3.7.

![Diagram of a node](image)

**Figure 3.7: The Structure of a Node**

Based on an observation by Maxemchuk [45] that in a two-connected network, the
probability of deflection is the same when there are $K$ buffers shared by two output links or there are $K$ buffers associated with each output link. The state transition diagram of the buffers can be drawn as in Figure 3.8. The total number of cells stored at a node never exceeds $K$. The states are labeled with the number of cells waiting at each output link. The transition arcs are labeled with the number of cells directed to each output link. The symbol $\bar{Y}$ represent possibilities other than $Y$ and the symbol $\bar{X}$ implies all possibilities. For example, $(X, 1)$ can be $(0, 1)$ or $(1, 1)$. Similarly, $(0, \bar{2})$ is either $(0, 0)$ or $(0, 1)$.

![State Transition Diagram for Buffer Utilization with 3 Buffers per Outgoing Link](image)

Figure 3.8: State Transition Diagram for Buffer Utilization with 3 Buffers per Outgoing Link

The model is developed based on three major approximations:

1. **Approximation 1**: Arrivals of cells on different incoming links are independent.

2. **Approximation 2**: The arrival of cells on an incoming link is independent from slot to slot.

3. **Approximation 3**: The choice of an output link made by a cell is independent of the choice of other cells.

An estimate of the link utilization is used to derive the performance statistics such as the probability of deflection, the average throughput per node per slot, and the average cell delay. Let $m_t(i)$ be the probability that a transit cell which will have a hopcount $i$ at the end of a slot is received on a particular link in slot $t$. Therefore, the
probability that a transit cell traverses the link in slot $t$ will be given by $\sum_{i=1}^{d} m_t(i)$. The vector $m_t$ is defined as $m_t = [m_t(0), m_t(1), \ldots, m_t(d)]$, where $d$ is the diameter of the network. Let $a(m_t, v)$ be the probability that a typical new cell is accepted, and $p(i, m_t, v)$ (or $p_0(i, m_t, v)$) be the probability that a typical transit cell (or a typical accepted new cell) loses contention, where $v$ is the Bernoulli injection parameter. We define another vector $\pi_t$ as the vector of output buffer occupancy probabilities $\pi_t = [\pi_t(0, 0), \pi_t(0, 1), \pi_t(0, 2), \ldots, \pi_t(K - 1, K), \pi_t(K, K)]$, where $\pi_t(i, j)$ is the probability that the buffer is in state $(i, j)$ at the beginning of slot $t$. By applying a one-to-one transformation on the state space such that a state $(x, y)$ is mapped onto a state $w$, where $w = (K + 1) \times x + y$, the state vector $\pi_t$ can be reduced to another vector $\nu_t$ where $\nu_t$ is given as $[\nu_t(0), \ldots, \nu_t(K), \nu_t(K + 1), \ldots, \nu_t((K = 1)^2 - 2), \nu_t((K + 1)^2 - 1)]$. At the beginning of slot 0, $\pi_0 = \nu_0 = [1, 0, 0, \ldots, 0]$. Let $Q_t$ be the state transition matrix consisting of elements $q_{ij}^{t}$. $0 \leq i, j \leq K$ where $q_{ij}^{t}$ denotes the transition probability from state $i$ to state $j$ during slot $t$. From $m_t$, the transition matrix $Q_t$ can be computed. Once we have $Q_t$, from the theory of Markov chains, the buffer occupancy probability vector $\pi_{t-1}$ at the beginning of cycle $(t - 1)$ can be computed using the equation $\nu_{t-1} = \nu_t Q_t$ and the identity $\pi_{t-1} \equiv \nu_{t-1}$.

Figure 3.9 shows the iterative evaluation of $\pi_t$ and $m_t$.

![Figure 3.9: Iterative Evaluation of $\pi_t$ and $m_t$](image)

At each iteration, the transition probabilities between the various states in the buffer are computed from the buffer occupancy probability vector $\pi_t$ and the link utilization vector $m_t$. These transition probabilities determine the buffer occupancy probability vector for slot $(t + 1)$, $\pi_{t-1}$. From $\pi_{t-1}$ the link utilization vector, $m_{t-1}$ is derived.
When the iterative process converges, the steady-state vector $\mathbf{m}$ is obtained, which can be used to derive the various performance metrics. The steady-state link utilization $\rho$ is given by $\sum_{i=0}^{d} m(i)$. The probability that a tagged cell is deflected is given by the product of the probability that the tagged cell loses conflict and the probability that the desired output buffer is full. At steady-state, the rate at which cells are offered and accepted at the source equals the rate at which cells are delivered to the sink. Therefore, the steady-state absolute throughput is given by $2 \times m(0) = c \times a(m, v)$. Choudhury and Li study performance of buffered deflection routing in the cases of the MSN, the ShuffleNet, and the Shuffle Exchange and compare their analytic results to those obtained from simulation. The simulation results show good agreement between the model and the simulation except in some networks which have very few equal-length paths between node pairs. They point out that deflection routing requires cell re-sequencing at the receiver and techniques to minimize re-sequencing delays and losses due to re-sequencing buffer overflow need to be investigated.

3.4.5 Abraham and Padmanabhan's Hypercube Model

Abraham and Padmanabhan provide a mathematical model [1] for predicting the performance of direct binary $n$-cubes or hypercubes and then compare their results with previous results for crossbars and indirect binary $n$-cube networks. Both unbuffered and buffered hypercubes are analyzed. In the case of buffered networks, each of the outgoing transmission links of a routing node is associated with queuing buffers, such that multiple cells contending for the same output link in a cycle are saved up in a queue associated with the link. A one-pass routing strategy (discussed in the previous chapter in Section 1.6) is employed. Cells at the incoming links are considered for routing in random order. In the case of unbuffered networks, contention among multiple cells to be routed to the same output link is resolved by selecting one cell at random as a winner from among the group of cells and blocking the rest of the cells. All the blocked cells are rejected and cleared from the system. In case of buffered networks, multiple cells heading for the same output channel in a cycle are saved in
a queue associated with the particular output channel. If the buffer size is assumed to be infinite then no cells are lost.

Abraham points out that under uniform traffic model assumption fairly small buffer sizes yield more than 90% of the performance of infinite buffers. Both single-accepting and multiple-accepting delivery channels are examined. The cell generation rate \( m_g \) - the probability of receiving a cell from the source - varies between 0 and 1. The performance measures of interest are the probability of acceptance of a newly generated cell, and the bandwidth of the network as a whole. The bandwidth of a network is defined as the maximum number of cells that can be absorbed by the network per unit time. In the case of buffered networks, the mean cell delay time, i.e., the average time a cell spends in the system before reaching the destination, is also an important performance metric. The results obtained from both the analytical model and the simulation data show that under a uniform traffic model, the direct cube performs better than the indirect cube. If indirect networks are constructed using switches of size less than \( 8 \times 8 \). Also, the network bandwidth is fully realized if multiple delivery channels with multiple sources per routing node are assumed.

Adding a small number of buffers (1 or 2) at the output side of the switching node dramatically reduces the cell loss probability. If a single delivery channel is assumed, then the node is provided with a buffer at the sink, to store multiple cells that are to be delivered. The required size of this sink buffer increases with increasing cell generation rate. As the cell generation load \( m_g \) approaches 1, the average cell delay also increases, but it does not grow to infinity even for infinite-buffered multiple accepting cube networks. This is because, even at a load of \( m_g = 1 \), the links of the network are not saturated. A link is said to be saturated when the probability that a cell is transmitted on a link approaches 1.0. Abraham and Padmanabhan point out that the maximum utilization of the links in the cube is \( N/2(N-1) \) when \( m_g = 1 \). They were interested to find out how much higher \( m_g \) can be before the cube reaches saturation in case of multiple-delivery channels. Since a switching node is connected to \( d \) output links, one bound can be established quickly: \( m_g \leq d \). A tighter bound is derived from the fact that the node under steady-state conditions can not
accept more than \( d \) deliverable cells arriving at the incoming links of the switching element. Under uniform traffic assumptions, \( m_g \) was shown to saturate at a value of \( 2(N - 1)/N \).

Performance of direct binary hypercubes under any non-uniform arbitrary general distribution of destinations of the cells is also studied. Instead of an equiprobable reference pattern, it can be assumed that a message generated at a node will have hopcount \( i \) with probability \( h_i \). Different reference patterns can be analyzed. One such reference pattern where locality of reference can be utilized is harmonic pattern \( h_i = 1/(\alpha * i) \) with \( \alpha = \sum_{i=1}^{d} 1/i \). This means the larger the distance \( (i) \), the lower the probability \( (h_i) \) of those packets being generated. To give an example, a 10-dimension hypercube that uses this harmonic reference pattern has 34% of all its references to nodes that are at distance one.

3.4.6 Greenberg and Hajek's Hypercube Model

In [24], Greenberg and Hajek carried out an approximate analysis of the transient and steady-state behavior of hypercube networks. They used deflection routing to resolve contention among multiple cells trying to go to the same outgoing link, instead of buffering them in a queue. The traffic model was assumed to be uniform, i.e., the destination of a new cell at any given node can be any of the \( 2^d - 1 \) nodes (excluding the node in which the cell was generated) with equal probability, where \( d \) is the dimension of the hypercube. The number of new cells generated at a node during any cycle followed a binomial distribution \( B(d, v/d) \), where \( 0 \leq v \leq d \) is the number of new cells generated during a slot. Therefore, as many as \( d \) cells may be injected into a node per unit time. The routing strategy adopted was one-pass and the order in which the incoming cells were considered for routing was random. The cells arriving at the incoming links of a node have higher routing priorities than newly generated cells. So, it may happen that some of the new cells are blocked as a result of non-availability of outgoing links. These blocked cells are rejected and cleared. The performance metrics of interest are the probability that a typical cell is blocked (possibly as a function of
time of arrival), the average delay of an accepted cell, and the distribution of how far a cell involved in typical deflection is from its destination at the time of deflection.

Two important assumptions were made in deriving the model:

1. The arrivals of cells on different incoming links of a node during any slot are independent.

2. Suppose there is a single cell at the node at any time slot which is \( i \) hops from its destination where \( 1 \leq i \leq d \). The cell will have \( i \) preferred outgoing links. The identities of these \( i \) links are randomly, uniformly distributed, independent of the links already assigned to other cells.

Using the above approximations, Greenberg and Hajek first modelled one node during one time slot. Before the beginning of a routing cycle any cells that arrived at the node for delivery in the previous slot are absorbed. Therefore, at the beginning of a slot there are \( \mathcal{U} \) continuing cells that were received from other nodes during the previous slot having non-zero routing code, and \( \mathcal{V} \) new cells generated locally and offered at the node. Since, at most one cell arrives per incoming link, \( \mathcal{U} \leq d \). Also, since only \( d \) cells can be transmitted out of \( \mathcal{U} + \mathcal{V} \) cells. \( \max\{0, (\mathcal{U} + \mathcal{V} - d)\} \) cells are blocked and rejected. Because continuing cells have higher priorities than newly generated cells, it will be the new cells that are blocked and lost. Out of \( \mathcal{V} \) new offered cells, \( \min\{\mathcal{V}, (d - \mathcal{U})\} \) are accepted. For any link, if \( m_t \) denotes the probability that a cell is transmitted or received on the link during the slot, then the number of continuing cells \( \mathcal{U} \) is given by a binomial distribution \( B(d, m_t) \). The number of new cells offered at a node \( \mathcal{V} \) is also given by another binomial distribution \( B(d, \nu/d) \), where \( \nu \) is the offered load. \( \mathcal{V} \) is independent of \( \mathcal{U} \). Let \( a(m_t, d, \nu) \) represent the probability that a typical new cell is accepted. The probability that a typical continuing cell that is \( i \) hops away from its destination is deflected is given by \( p(i, m_t, d, \nu) \). Similarly, the probability that a typical accepted new cell with hopcount \( i \) is deflected is given by \( p_0(i, m_t, d, \nu) \). If \( m_t(i) \) is the probability that a cell \( i \) hops away from its destination (at the end of the slot \( t \)) traverses a link, then the vector \( \mathbf{m}_t \) can be written as
\[ m_t = (m_t(0), m_t(1), \ldots, m_t(d)). \] (3.1)

At the beginning of slot 0, the network is empty, so \( m_0 \equiv 0 \).

From \( m_t \) one can compute \( m_{t-1} \), \( t \geq 0 \) using the following equations.

\[
\begin{align*}
    m_t &= \sum_{i=1}^{d} m_t(i) \\
    m_{t-1}(i) &= m_t(i-1)p(i-1, m_t, d, v) \\
    &\quad + m_t(i+1)(1-p(i+1, m_t, d, v)) \\
    &\quad + \frac{a(m_t, d, v)v}{d}q(i-1)p_0(i, m_t, d, v) \\
    &\quad + \frac{a(m_t, d, v)v}{d}q(i+1)(1-p_0(i+1, m_t, d, v));
\end{align*}
\] (3.3)

where \( q(i) \) represents the probability that a cell generated has a hopcount equal to \( i \). Equation 3.3 signifies that the probability that a link will carry a cell during slot \( t + 1 \) whose hopcount will become \( i \) at the end of the slot, will be computed from the following four probability values:

- the probability that a continuing cell with hopcount \( i - 1 \) received at the node during slot \( t \) will be deflected during slot \( t + 1 \).

- the probability that a continuing cell with hopcount \( i + 1 \) received at the node during slot \( t \) will not be deflected during slot \( t + 1 \).

- the probability that a new cell offered at the node at the beginning of slot \( t + 1 \) with distance \( i - 1 \) is accepted and deflected during slot \( t + 1 \). and

- the probability that a new cell offered at the node at the beginning of slot \( t + 1 \) with distance \( i + 1 \) is accepted and not deflected during slot \( t + 1 \).
The above equations, called the update equations, can be used to compute the expected number of cells with any hopcount $i$ that the node transmits during a slot $t+1$. For example, the expected number of cells whose hopcount values will become $i$ at the end of the slot $t+1$ will be given by $d \times m_{t-1}(i)$. The expected number of cells in each of the above four groups is given by $d$ times the corresponding term in Equation 3.3.

As time $t$ tends to infinity, the network approaches equilibrium. The link utilization vector $\overline{m}$ reaches a steady-state value $\overline{m}$ where $\overline{m} = (\overline{m}(0), \ldots, \overline{m}(d))$. The subscript $t$ is dropped from the update equations and the new equations are given by:

\[
\begin{align*}
\overline{m} &= \sum_{i=1}^{d} \overline{m}(i) \\
\overline{m}(i) &= \overline{m}(i-1)p(i-1, \overline{m}, d, \nu) \\
&+ \overline{m}(i+1)(1-p(i+1, \overline{m}, d, \nu)) \\
&+ a(\overline{m}, d, \nu)\nu q(i-1)p_0(i, \overline{m}, d, \nu) \\
&+ \frac{a(\overline{m}, d, \nu)\nu}{d} q(i+1)(1-p_0(i+1, \overline{m}, d, \nu)) \\
0 \leq i \leq d
\end{align*}
\]

(3.4)

(3.5)

At equilibrium the rate at which new cells are offered and accepted at a node (given by $a(\overline{m}, d, \nu) \times \nu$) becomes equal to the rate at which continuing cells are received and absorbed (given by $\overline{m}(0) \times d$). The unknown vector $\overline{m}$ is obtained from a fixed point solution of the time-recurrence equation. Once $\overline{m}$ is obtained, all the performance figures like the probability of deflection $p$ of a continuing cell or the probability of deflection $p_0$ of a new accepted cell, the probability of acceptance of an offered new cell, and the average cell delay $T(\overline{m}, d, \nu)$ are all derived.

Greenberg and Hajek also study the asymptotics of the key performance parameters like $\overline{m}$, $a(\overline{m}, d, \nu)$, and $T(\overline{m}, d, \nu)$ as the hypercube dimension $d$ tends to infinity ($d \to \infty$). The asymptotic analysis indicates that as the network size increases performance becomes optimal. If the number of new arrivals plus the number of continuing
cells exceeds the number of outgoing links \((d)\). some of the new arrivals are dropped. Provision of input queueing to hold these excess arrivals until they can be admitted into the network may avoid the loss but will result in a queueing delay. Greenberg and Hajek claim that their model can be extended to analyze the network with input queues. They note that deflection routing can be extended to use buffers internal to the nodes: buffers may increase throughput under a uniform traffic model, but if the traffic is not uniform then additional buffering may lessen the algorithm's adaptivity to congestion and may be detrimental to the performance.

3.5 Szymanski's Hypercube Model

A fiber optic hypercube can have a very high link bandwidth (in the gigabit/sec range). Each directed link in a fiber optic hypercube is implemented with a fixed wavelength laser and all optical transmissions are wavelength multiplexed into one or more fibers. Traditional cell-switched hypercubes use shortest-path routing with buffering. When the buffers are full, some sort of flow control is required to inform the sending node. This needs a backwards transmission mechanism. In electrical hypercubes, an extra wire can be added to each directed link to achieve that purpose. But in a fiber optic hypercube providing an optical status link will call for another laser/photodetector combination. To solve the problem, Szymanski proposed \([57]\) using deflection routing or hot-potato routing instead of buffering in fiber optic hypercube network. Two different versions of hot-potato routing algorithm are proposed: the basic version and the deluxe version. In the basic version, all cells are assumed to have the same priority whereas in the deluxe version, cells closer to their destinations have higher priority.

The network is operated in a synchronous or discrete-time manner. At the beginning of a time slot \(t\), every router transmits all of its cells and simultaneously receives many cells from its neighbors. Routing decisions are made for the incoming cells as well as for the freshly injected cells during time \(t\), i.e., during the time interval \([t, t + 1)\). Based on those decisions, the cells are transmitted at the beginning of
cycle $t + 1$. Both one-pass and two-pass routing strategies are studied. Each node is assumed to inject at most one cell into the network during any cycle. The traffic is assumed to be uniform. Szymanski assumes that all nodes in the hypercube are statistically identical and independent. Therefore, the state of the entire hypercube can be characterized by the state of a single node. To model a node, further simplifying assumptions are made. All links in the node are also assumed to be statistically identical and independent.

The performance measures computed are average bandwidth, average delay, delay distributions (i.e., the probability distributions that a cell with initial distance $i$ requires $j$ cycles before reaching its destinations), and the probability that an average cell (or a cell with a given initial distance) is delayed more than an arbitrary threshold. In the following we present the analysis given by Szymanski for the basic deflection routing algorithm.

Let $d$ be the number of dimensions of a hypercube which consists of $N = 2^d$ nodes. If $\alpha$ is the probability with which new cells inserted into the network traverse any particular dimension, then the average path distance (including self-destined traffic) is given by $\alpha \times \log N$. For random uniform traffic, $\alpha = 0.5$ and the average path distance is $\log N/2$. The following variables are defined:

\[
\begin{align*}
dst_{i,t} & = \text{Pr}[\text{a cell is distance } i \text{ away from its destination at the beginning of a time slot } t] \\
u_t & = \text{Pr}[\text{a cell arrives on an incoming link during the interval } [t, t + 1)] \\
u_o_t & = \text{Pr}[\text{a cell that arrives on an incoming link during the interval } [t, t + 1) \text{ is not for delivery and must be transmitted out}] \\
rxbw_t & = \text{Average number of cells delivered during cycle } t \\
\text{(i.e., during the interval } [t, t + 1)) \\
txbw_t & = \text{Average number of new cells accepted by a node during cycle } t \\
\text{(i.e., during the interval } [t, t + 1)) \\
\lambda & = \text{Pr}[\text{a new cell is offered to a node during any cycle}]
\text{(i.e., the offered load )}
\end{align*}
\]
The probability that a cell is distance $i$ away at the beginning of cycle 0 is given

$$dst_{i,0} = \binom{d}{i} \times \alpha' \times (1 - \alpha)^{n-i} \quad (3.6)$$

To start with, at the beginning of cycle 0, $u_{i_0} = \lambda/d$. The number of cells delivered during cycle $t$, i.e., the receive bandwidth, the number of new cells accepted during cycle $t$, i.e., the transmit bandwidth, and $u_{o_t}$ are given by the following equations:

$$rxbw_t = u_{i_t} \times dst_{0,t} \times d \quad (3.7)$$
$$uo_t = u_{i_t} \times (1 - dst_{0,t}) \quad (3.8)$$
$$txbw_t = (1 - uo_{i_t}^d) \times \lambda \quad (3.9)$$

Equations 3.7 and 3.8 are self-explanatory. Equation 3.9 signifies that an offered new cell is accepted only when there is at least one free outgoing link. On average, a node will have both continuing cells and new cells given by $d \times u_{o_t} + rxbw_t$. The new cells will cause a redistribution of the distance values ($dst_{i,t}$) of the cells. The new distributions ($dst'_{i,t}$) are given by:

$$dst'_{i,t} = \frac{d \times u_{o_t}}{d \times u_{o_t} + trxw_t} \times dst_{i,t} + \frac{trxw_t}{d \times u_{o_t} + trxw_t} \times dst_{i,0} \quad (3.10)$$

with $0 \leq i \leq d$.

The cells that arrive at the links during a cycle $t$ are stored in the buffers associated with the links. Assume that the buffers are labeled $1, \ldots, d$. Routing decisions are made for all these cells during cycle $t$. Szymanski divides each routing cycle $t$ into a series of conceptual time phases $p$, $1 \leq p \leq d$, during which the incoming buffers are serviced (in any order). The following variables are defined where $p$ is used to denote the phase:
\[ cu_{p,t} = \Pr[\text{an outgoing link will carry a cell in the next clock cycle after considering the cells in the buffers} ~ 1\cdots p \text{ during time } t] \]

\[ pb_{p,t} = \Pr[\text{a cell whose hopcount is } i \text{ can not be successfully forwarded to a suitable outgoing link. after considering the cells in buffers } 1\cdots p - 1, \text{ during time } t] \]

\[ amv_{p,t} = \Pr[\text{a cell will win exclusive access to a suitable outgoing link and hence will be forwarded successfully towards its destination in the next cycle. averaged over all distances. during phase } p \text{ of time } t] \]

\[ mv_{p,t} = \Pr[\text{a cell will be forwarded successfully over a suitable outgoing link. averaged over cells in buffers } 1\cdots p - 1, \text{ during time } t] \]

Given those definitions, the variables can be initialized for the phase 0 of time \( t \) as: \( pb_{0,t} = cu_{0,t} = 0 \). When \( i \geq p \), we can write \( pb_{p-1,t} = pb_{p,t} = 0 \). Since, all cells have the same priorities in the basic routing algorithm, the probability that a cell in buffer \( p + 1 \) with distance \( i \) will not be able to move forward successfully after considering the cells in buffers \( 1\cdots p \) will be given by:

\[
pb_{p-1,t} = \binom{i}{i} \cdot cu_{p,t}^i \cdot (1 - cu_{p,t})^{i-1}
\] (3.11)

Similarly, the probability that a cell in buffer \( p + 1 \) will win exclusive access when averaged over all distances is given by Equation 3.12. After considering all the cells in buffers \( 1\cdots p \), the probability with which an outgoing link will carry a cell is given by Equation 3.13.

\[
amv_{p-1,t} = \frac{1}{n} \sum_{i=1}^{d} (1 - pb_{p,t})
\] (3.12)
\[ cu_{p-1,t} = cu_{p,t} + uo_t \times amv_{p-1,t} \times \frac{1}{d} \] (3.13)

In the one-pass rule, the cells that can not be forwarded successfully and hence deflected are assigned outgoing links right away. This will cause Equation 3.13 to be replaced by Equation 3.14.

\[ cu_{p-1,t} = uo_t \times p/d \] (3.14)

The probability that a cell whose distance is \( i \) at time \( t \) will not be deflected during time \( t \) is computed as:

\[ mv_{i,t} = \frac{1}{d} \times \sum_{p=1}^{d} (1 - pb_{p,i,t}) \]
\[ 0 \leq p < d \] (3.15)

Since the deflection probability of a cell of distance \( i \) changes with each phase \( p \), the average is determined over all phases as shown in Equation 3.15. Finally, the probability that an incoming link will receive a cell in the next cycle is obtained as follows:

\[ u_{i_{t-1}} = uo_t \] (3.16)

The distribution of the distance values of cells for the next cycle is also updated to reflect the fact that some cells move closer to their destinations whereas some move further away.
\[ \begin{align*}
\text{dst}_{i,t+1} & = \text{dst}_{i-1,t}'' \ast \text{mv}_{i-1,t} + \text{dst}_{i-1,t}'' \ast (1.0 - \text{mv}_{i-1,t}) \tag{3.17} \\
\text{dst}_{d,t-1} & = \text{dst}_{d-1,t}'' \ast (1 - \text{mv}_{d-1,t}) \tag{3.18}
\end{align*} \]

where

\[ \text{dst}_{i,t}'' = \frac{\text{dst}_{i,t}'}{\sum_{j=0}^{d} \text{dst}_{j,t}'} \tag{3.19} \]

The time recursive equations are solved by iteration until they converge. In equilibrium, the average number of new cells accepted at the node must be equal to the average number of cells delivered. The steady-state values of the various variables defined above are used to compute the performance metrics. For computing average delay, an iterative approach is taken. Let \( ad_{i,t} \) be the average delay experienced by a cell with initial distance \( i \) to reach its destination. The subscript \( t \) represents the iteration number. If \( \text{mv}_i \) represents the equilibrium probability values for \( \text{mv}_{i,t} \), then starting from an initial condition of \( ad_{i,0} = 0 \), for \( 0 \leq i \leq d \), one can write:

\[ \begin{align*}
\text{ad}_{i,t+1} & = \text{mv}_i \ast (1 + \text{ad}_{i-1,t}) + (1.0 - \text{mv}_i)(1.0 + \text{ad}_{i-1,t}) \tag{3.20} \\
\text{ad}_{d,t+1} & = \text{mv}_d \ast (1.0 + \text{ad}_{d-1,t}) \tag{3.21}
\end{align*} \]

Equations 3.20 and 3.21 are allowed to converge, at which point the equilibrium values of \( ad_i \), \( 0 \leq i \leq d \) are obtained. The average delay of a cell leaving the hypercube is then found as

\[ \text{average delay} = \sum_{i=0}^{d} \text{ad}_i \ast \binom{d}{i} \alpha^i \ast (1 - \alpha)^{d-i}. \tag{3.22} \]
Szymanski points out that a cell with an initial distance \( i \) may require, theoretically, an infinite number of cycles before reaching its destination, but the probability of such an incident happening is very small. He computes the probability that a cell needs more than an arbitrary threshold number of cycles before being delivered.

An analysis of the *deluxe* routing algorithm in which cells closer to their destinations are routed first, is also given. The results of both the *basic* and the *deluxe* routing algorithms are compared with those of the simulation. It is found that the deluxe algorithm performs better than the basic. Under a uniformly distributed random traffic model *hot-potato* routing significantly outperforms shortest-path routing with buffering and flow control. Also, *hot-potato* routing provides a powerful fault-tolerance mechanism by routing around faulty links (or nodes), if there are any.

### 3.6 Motivation of the Thesis

In order to properly provision, control, and design hypercube switches, it is essential that their performance capabilities be completely understood. As discussed earlier, one of the approaches to do such performance studies is to build analytical models of the switch. It is evident from the preceding discussion that efforts have been made to build analytical models for hypercubes. However, either approximate Markov models or some probabilistic analyses have been used to capture the switch performance, mainly because the number of Markov states became excessive and the equations were no longer analytically tractable. Abraham and Padmanabhan provided analytical models for both unbuffered and buffered hypercubes, but their models did not consider deflection as the conflict resolution policy. Greenberg and Hajek also gave an approximate model of hypercube networks which used deflection routing but did not use buffers within the switching nodes. Szymanski advocated using deflection in his fiber optic hypercube to avoid storage buffers. Approximate analytical models for buffered switches that use deflection have been proposed by Choudhury and Li, but in the context of simpler two-connected regular topology switches like MSN, ShuffleNets and Shuffle Exchange. Also, the buffers considered in their models were
output buffers dedicated to each output link. An interesting alternative to output buffering is *shared buffering*, where buffers are shared among all output links. It is well known that shared buffering needs substantially fewer buffers compared to output buffering [28]. Therefore, using deflection routing with few buffers in a node not only makes the design very simple, but also achieves performance comparable to that of output buffered nodes with the same amount of buffers per outlet [11]. There is no single analytic model in the literature that studies deflection routing in hypercube switches that use shared buffering. This motivates the present work. We present an approximate Markov model for studying buffered binary hypercube switches running the SCS routing algorithm. The model is developed in the next chapter. The model results are validated against simulation data in Chapter 6.
Chapter 4

An Approximate Model for a Buffered Deflecting Hypercube Switch

The motivation for building a new model to study the performance of the SCS routing algorithm in buffered deflecting hypercube switches was discussed in the previous chapter. In this chapter, we develop an approximate Markov model of one switching element to capture the behavior of an entire $d$-dimension switch. This approach avoids the generation of state space that increases exponentially with the number of switching elements. The number of possible states for our switch model is finite. A discrete-time model of the switching element which is characterized by time-slotted and synchronous service is built. The approximate model is solved numerically using an iterative approach. The procedure starts with a known initial state and computes slot by slot the distributions of the output states of the element. When the numerical solution converges, a steady-state distribution is realized which is then used to compute the various performance metrics.

The performance metrics of interest are the average number of forward-in cells arriving per cycle at the forward incoming links of a switching element, the average number of cells stored in the queuing buffers, and the average number of cells
retrograded with a variation in offered load. The queue performance is important because queue size is a design parameter that influences the performance of the hypercube switch. Retrograde performance is important since retrograding should be minimal to have low latency figures. Average number of forward-in cells per cycle is also important because that gives an estimate of link utilization.

There are some difficulties in capturing the queuing behavior of the model. First, the cells in the queue buffer are not processed in a conventional FIFO way. At the beginning of every clock cycle (also known as cell exchange cycle or CEC), all the cells in the queue buffer of the switching element are considered simultaneously for routing. Each routing CEC is divided into conceptual time phases $t_p$, $1 \leq t_p \leq Q_n$ where $Q_n$ is the number of cells in the queue buffer. The cells in the queue are then processed according to some routing policy (shortest-first, distant-first, or random) in these time phases. The concept is similar to that used in handling forward-in cells in Szymanski's model [57].

The second problem is in characterizing the cell arrival process to the queue. The cells that arrive at the queue during the routing CEC consist of three different types of cells: the cells that were stored in the queue in the previous CEC and could not be assigned successfully, the cells that arrived at the forward incoming links of the router from its neighbors and the freshly injected cells from the router's local source. While the distribution of the injected cells are known (because of the input traffic assumption), it is quite difficult to find the composite arrival process. These are some of the reasons why we need to make certain approximations and assumptions to build our model, which will be explained later.

In Section 4.1 we define the model of a single node or single switching element. Section 4.2 lists the assumptions made in building the model. Section 4.3 makes an approximate analysis and specifies the steps followed in arriving at numerical solution of the model. Finally, Section 4.4 gives the summary. The results of the analysis are given in Chapter 6.
4.1 Single Node Model

The regular structure of the hypercube switches and the uniform traffic assumptions justify building a model for one switching element instead of a model for the entire switch consisting of $2^d$ elements. Such an approach, known as a one node model, has been used by previous researchers in connection with MSN or regular 2-connected networks [25, 16]. Figure 4.1 shows the model of one switching element of a $d$-dimensional hypercube. Each bi-directional link that connects the node with one of its $d$ neighbors is shown in the figure as a pair of incoming and outgoing links. As a result, the node has $d$ cube edge incoming links and $d$ cube edge outgoing links. Each incoming cube edge link has an associated buffer, called a forward-in buffer to hold the arriving cell. Similarly, each of the outgoing links has an associated buffer, called forward-out buffer, to hold the cell that needs to be routed to the node's neighbor. These buffers are not the same as queuing buffers needed to hold the cells in case of routing conflict.

![Diagram of Single Switching Element of a Hypercube]

Figure 4.1: Single Switching Element of a Hypercube

The injection buffer holds the cell generated locally. Depending on the traffic
model assumptions, there may be more than one injection buffer at a switch. The delivery buffers hold cells destined for the node. We assume, in the case of injection buffers and delivery buffers, that each buffer can hold only one cell.

The switching element has a shared queue buffer to reduce deflection of cells. Unlike the model used in [16] where each of the outgoing links is associated with its own queuing buffers, the buffer space in our model is shared by all the outgoing links. For convenience of analysis, the buffer space is time partitioned into two separate entities called queue-out buffers and queue-in buffers, as shown in Figure 4.1. This partitioning helps visualize the behavior of the switch at the beginning of the routing cycle when all cells are at the input side of the switch and at the end of the routing cycle when all cells are at the output side of the switch. Cells that end up in shared queueing buffers as a result of routing are assumed to be stored in queue-out buffers. At the beginning of the next routing cycle, the cells stored in the queue-out buffer are transferred to the queue-in buffers. Actual implementations, however, avoid the duplication of input and output side buffers.

It was mentioned earlier that the switch elements in the hypercube operate synchronously and the time slot during which they operate is called CEC. As cells arrive at the forward incoming links, they are stored in forward-in buffers. Service or routing of cells is synchronized to take place only at slot boundaries. Cells that arrive in a slot are serviced from the beginning of the next slot. The length of the slot is normalized to unit time. Slots are sequentially numbered with natural numbers, so that the $k$-th slot is located in time $(k - 1, k)(k = 1, 2, \cdots)$. If $k^-$ and $k^+$ denote the two time points immediately before the end of the time slot $k$ and after the time slot $k$, a cell completing service in slot $k$ is considered to be leaving the system in $(k^-, k]$. No restrictions are imposed on arrival times of cells in a slot. Cell arrivals over the entire slot $k$ may be assumed to be taking place at time $k^-$ and as a result may be specified by the number of arrivals in a slot rather than by inter-arrival distribution.

At any node, at the beginning of any CEC, there are $U^-$ queued-in cells, $V$ forward-in cells (received from other nodes during the previous CEC), and $W$ new cells (generated locally) stored in the injection buffers. The number of queued-in cells $U^-$ may
vary between zero and the maximum size of the queue $Q$, i.e., $U' \leq Q$. Since at most one cell arrives per incoming link, $V' \leq d$, where $d$ is the dimension of the hypercube. The number of locally generated cells $W'$ per CEC may vary between 0 and 1 or between 0 and $d$, depending on the injection traffic model assumed.

A cell in any of these input buffers must have a distance value between 0 and $d$. Distance values reduce by one for cells that are successfully forwarded, remain unchanged for cells stored in the queue, and increase by one for cells that suffer retrograde motion.

Routing decisions are made first for the cells in the queue-in buffers, followed by the cells in the forward-in buffers, and finally for the cells in the injection buffer(s). We assume that the cells are routed according to the shortest-first policy, i.e., cells with less distance to travel are given preference over cells with longer distances. If two or more cells have the same distance, then a random selection is made. Cells having a routing code of zero are sent to delivery buffers if available. Cells with non-zero routing codes are forwarded; those which can not be forwarded are stored in the queue-out buffers or are retrograded. At the end of the routing cycle, cells in the queue-out buffers are transferred to the queue-in buffers of the switch for the purpose of routing in the next CEC and cells in the forward-out buffers are transmitted on the outgoing links to the node's neighbors.

In general, if we assume that all queued-in cells $U$, all forward incoming cells $V'$ and all locally generated cells $W'$ have non-zero routing codes, then at any time there will be $U' + V' + W'$ cells at the switch contending for the $d$ outgoing links out of which at least $(U' + V' + W' - d)^+$ cells are blocked. Since locally generated cells have the lowest priority, it is possible that these cells may be rejected and dropped because of non-availability of queuing buffers or forward-out buffers, thereby causing some injection loss. The number of new locally generated cells accepted is $\max\{0, \min\{W', (d - (U' + V'))\}\}$. However, if a new cell is accepted, the SCS routing algorithm guarantees that the cell will never be lost [47].

In order to keep the model analytically tractable, we make the following assumptions:
4.2 Assumptions

1. Local sources at each of the switching elements are assumed to inject traffic following a simple Bernoulli model. Complex traffic models are left for future research. The number of new cells W offered to a node by its own local source during any cycle follows a Bernoulli distribution, $Ber(w)$, where $w$ is the probability parameter $0 \leq w \leq 1$. The number of new cells generated locally is independent from node to node and from CEC to CEC.

2. The destinations of the new cells are assumed to be distributed uniformly over the set of $2^d$ nodes (including itself). In a $d$-dimension hypercube, the number of nodes at a distance $i$ from a particular node is given by $\binom{d}{i}$. Since there are $2^d$ routers in the hypercube, the probability that a new cell has its destination $i$ hops away is given by

$$q(i) = \binom{d}{i} / 2^d.$$  \hspace{1cm} (4.1)

3. If a new cell generated locally at a node is not accepted, it is rejected and cleared.

4. For any node and any time slot, the arrival of a cell on one of the node's $d$ incoming links is independent of arrivals on other incoming links.

5. The choice of an output link among $i$ preferred output links of a cell having hopcount $i$, is done randomly and uniformly independent of the links chosen by other cells.

The performance measures of interest are the average number of forward-in cells arriving per CEC at the forward incoming links of a switching element, the average number of cells per CEC stored in the queuing buffers of the node, the average
number of cells per CEC having retrograde motion, the injection loss probability, and average cell latency, with a variation in injection load. The queue performance is important because that informs us of the size of the queue to be selected for desired switch performance. Retrograde performance is also important since the proportion of retrograded cells should be minimal. Injection loss probability is the probability with which a cell in the injection buffer is not accepted into the network and is the only source of loss in the network when both queuing buffers and retrograding are used. Cell latency is another good performance indicator of switch performance. From the average number of forward-in cells we get an estimate of the link utilization.

4.3 Analysis

The class of a cell is defined as the distance of the cell from its destination. In a $d$-dimension hypercube, a cell can have distance values between 0 and $d$. Therefore, the maximum number of possible classes is $(d + 1)$. All cells in a switching element must belong to one of these classes. In general, if $N$ is the total number of cells in an element during any CEC, and $k$ is the number of classes, then the number of possible ways in which $N$ cells can be grouped into $k$ classes is given by:

$$N_k = \binom{N + k - 1}{k - 1}.$$  \hspace{1cm} (4.2)

Each of these cell-class combinations is called a vector. A set of vectors is called a vector space. Vector spaces can be associated with queue-in buffers ($QI_{vect}$), queue-out buffers ($QO_{vect}$), forward-in buffers ($FI_{vect}$), forward-out buffers ($FO_{vect}$), and injection buffers ($I, V, J_{vect}$). Since each vector is unique, it is possible to index them. A generalized tuple notation for, say, the $j$-th vector or combination will be given by $< n_j : n_{0j}, n_{1j}, n_{2j}, \ldots, n_{dj} >$ where $n_j = \sum_{i=0}^{d} n_{ij}$, is the total number of cells in $j$-th vector. $n_{0j}, n_{1j}, \ldots$ represent the number of cells belonging to class 0, class 1, and so on. If the maximum size of the queue buffer is equal to $Q$, the maximum number of
vectors in $Q_{vect}$ or $QO_{vect}$ are given by:

$$|Q_{vect}| = |QO_{vect}| = \sum_{n=0}^{n=Q} \binom{n + d}{d}.$$  

(4.3)

where $|Q_{vect}|$ and $|QO_{vect}|$ indicate the cardinalities of the queue-in and queue-out vector spaces, respectively. Similarly, since $d$ is the maximum number of forward-in or forward-out buffers, one can enumerate all the possible vectors of forward-in ($FI_{vect}$) or forward-out ($FO_{vect}$) vector space.

By associating probability values with all the vectors of any of these vector spaces, the corresponding distribution is realized. If we assume, for example, that all the vectors of the queue-in vector space are equally likely to occur, then the probability that there are only $x$ cells in the queue-in buffer (of maximum size $Q$), will be given by \( \binom{x + d}{d} / |Q_{vect}| \).

Cells in buffers may undergo class transitions as a result of routing. A class $k$ cell $(0 \leq k \leq d)$ at the beginning of a routing CEC, when successfully forwarded towards its destination becomes a class $k - 1$ cell. The class of a cell remains the same if it gets queued and changes to $k + 1$, $0 \leq k \leq d$, in the event of retrograding.

Operation of a switching element during any routing CEC $t$, $(t - 1, t]$ can be divided, for convenience of analysis, into two separate sub-slots: one at the beginning of the routing CEC when all cells are at the input side of the switching element (i.e., queue-in buffers, forward-in buffers, and injection buffers), and the other at the end of the routing CEC, when all cells are at the output buffers (i.e., forward-out buffers, queue-out buffers, and delivery buffers). We associate two different subscripts 1 and 2 with a routing CEC $t$ (i.e., $t_1$ and $t_2$) to denote these two phases of any CEC $t$. The vectors are also subscripted with $t_1$ or $t_2$ to indicate the time instant. Both symbols $t^-$ and $t_2$ are used synonymously to represent the end of a routing CEC $t$.

The state of a switching element can be defined both at the beginning of a CEC ($t_1$) as well as at the end of a CEC ($t_2$). The state at the beginning of a routing
CEC consists of three components: (1) the number and classes of cells in the queue-in buffers, (2) the number and classes of cells in the forward-in buffers, and (3) the number and classes of cells in the injection buffers. The first component can be described by a queue-in vector, the second one by a forward-in vector, and the last component by an injection vector, all of which belong to the corresponding vector spaces. A typical node state at the beginning of the routing cycle ($t_1$) can be represented as $\langle FI_{m_1}, QI_{n_1}, I\!N\!J_{p_1} \rangle$. where $FI_{m_1}$, $QI_{n_1}$, and $I\!N\!J_{p_1}$ represent the $m$-th forward-in vector, $n$-th queue-in vector and $p$-th injection vectors respectively.

The input state space ($S$) is finite and the number of states in the state space is equal to:

$$|S| = |FI_{\text{rect}}| \times |QI_{\text{rect}}| \times |I\!N\!J_{\text{rect}}|$$

(4.4)

where $|S|$ represents the cardinality of the state space $S$. The input state space can be indexed with a tuple of three indices $<i, j, k>$, where $i$, $j$, and $k$ represent the forward-in vector, the queue-in vector, and the injection vector respectively. By associating probability values with each and every state one can realize the distributions for the entire state space. The probability of a switching element being in an input state $s = <m, n, p>$, $s \in S$ is given by the joint distribution

$$p_{t_1}(s) = Pr[FI_{t_1} = m; QI_{t_1} = n; I\!N\!J_{t_1} = p]$$

(4.5)

where $p_{t_1}(s)$ represents the probability that the state at time $t_1$ is $s_{t_1}$. Since the distributions of the three component vector spaces are independent of each other, as will be seen later. the joint probability will be given by the product of the three individual distributions:
\[ p_{t_i}(s) = Pr[FI_{t_i} = m] \times Pr[QI_{t_i} = n] \times Pr[I.N.J_{t_i} = p] \]  

(4.6)

where \( Pr[FI_{t_i} = m], Pr[QI_{t_i} = n], \) and \( Pr[I.N.J_{t_i} = p] \) indicate the probabilities of having \( m \)-th forward-in vector, \( n \)-th queue-in vector, and \( p \)-th injection vector.

Let

\[ m_t = Pr[ \text{a cell is transmitted on the link during a CEC} t] \]

For a fixed link, let us define \( m_t(i), 0 \leq i \leq d \), to be the probability that a cell \( i \) hops away from its destination (at the end of the slot \( t \)) traverses the link during slot \( t \). Let \( \mathbf{m}_t = (m_t(0), m_t(1), \ldots, m_t(d)) \) represent the link utilization vector. We can write

\[ m_t = \sum_{i=0}^{d} m_t(i) \]  

(4.7)

The distributions of the forward-in vector space \( FI_{vect} \) can be obtained from the link utilization vector \( \mathbf{m}_t \). Since arrival of cells on the different incoming links from the neighboring nodes is assumed independent, the probability of synthesizing \( i \)-th forward-in vector \( FI_i \), which has a total of \( n_i \) cells consisting of \( n_{i0} \) cells of distance 0, \( n_{i1} \) cells of distance 1, \( n_{i2} \) cells of distance 2, \ldots, \( n_{id} \) cells of distance \( d \), where \( n_i = \sum_{j=0}^{d} n_{ij} \), will follow a multinomial distribution:

\[ FI_{i,t_i} = \binom{d}{n_{i0}, n_{i1}, n_{i2}, \ldots, n_{id}} m_t(0)^{n_{i0}} m_t(1)^{n_{i1}} m_t(2)^{n_{i2}} \cdots m_t(d)^{n_{id}} \]  

(4.8)

State of a node \( s'_{t_2} \) at the end of a routing CEC \( (t_2) \) consists of four major components: (1) the number of cells delivered, (2) the number and class of the cells in the forward-out buffers, (3) the number and class of the cells in the queue-out buffer, and (4) the number of cells that are lost at the time of injection. Cells that are in forward-out buffers include both the cells that are successfully assigned their
desired outgoing links and also the cells that suffer deflection. In order to capture retrograde performance of the switch, cells in forward-out buffers are split at the cost of increased state space into two separate subcomponents: (1) the number and class of the cells in the forward-out buffers that have been successfully assigned their desired outgoing links, and (2) the number and class of the cells that have been retrograded. As a result, we use a five tuple representation of an output state at the end of a routing cycle $t_2$: $<\text{Del}_{t_2}, \text{FS}_{t_2}, \text{QO}_{j_{t_2}}, \text{RO}_{k_{t_2}}, \text{L}_{t_2}>$. $\text{Del}_{t_2}$ represents the number of cells delivered and is a random variable ranging between 0 and $(d - 1)$ (includes the possible distance zero cell injected by the local source for self-delivery). $\text{FS}_{t_2}, \text{QO}_{j_{t_2}},$ and $\text{RO}_{k_{t_2}}$ are $i$-th forward-success vector, $j$-th queue-out vector and $k$-th retrograde vectors respectively. The random variable $\text{L}_{t_2}, (0 \leq \text{L}_{t_2} \leq 1)$ indicates the number of cells lost at the time of injection, also known as injection loss. The actual forward-out vector $\text{FO}_{mt_2}$ of the state can be obtained by adding the two subcomponents $\text{FS}_{t_2}$ and $\text{RO}_{k_{t_2}}$. The output state space $S'$ is finite and the cardinality of the state space is given by:

\[
|S'| = (d + 1) \times |\text{FO}_{\text{vect}}| \times |\text{QO}_{\text{vect}}| \times |\text{FO}_{\text{vect}}| \times 2
\]  

(4.9)

where $|\text{FO}_{\text{vect}}|$ and $|\text{QO}_{\text{vect}}|$ represent the cardinalities of the forward-out vector space and the queue-out vector space, respectively. The factor 2 in the above equation accounts for the two different possibilities: cell loss, and no cell loss. The vector space of forward-out buffers is the same as that of forward-in buffers. Similarly, the vector space of queue-out buffers is the same as that of queue-in buffers.

\[
\text{FO}_{\text{vect}} = F\text{I}_{\text{vect}}
\]  

(4.10)

\[
\text{QO}_{\text{vect}} = Q\text{I}_{\text{vect}}
\]  

(4.11)
Strictly speaking, $FS_{vect} \subseteq FO_{vect}$ and $RO_{vect} \subseteq FO_{vect}$. But for ease of analysis, we assume $FS_{vect} = RO_{vect} = FO_{vect}$.

As with the input state space, the distribution of the output state space can be obtained by associating probability values with each and every output state. The probability that a switching element is in state $s'_{t_2} = <x\cdot i\cdot j\cdot k\cdot y>$, $s'_{t_2} \in S'$ at the end of a routing cycle $t_2$, is given by the joint probability:

$$p'_{t_2}(s') = Pr[Del_{t_2} = x; FS_{t_2} = i; QO_{t_2} = j; RO_{t_2} = k; L_{t_2} = y] \quad (4.12)$$

where $p'_{t_2}(s')$ is the probability that the state at time $t_2$ is $s'_{t_2}$. Unlike the case of input state, the output state probability depends on the joint occurrence of all its constituent components and cannot be obtained by the product of the individual distributions.

### 4.3.1 Approximate Algorithm

During a CEC, a switching element makes transitions from an input state to one or more output states. The probability with which various output states are generated depends on the probability of the input state and the routing algorithm running in the switch. Switching, therefore, can be viewed as a mapping or transformation between input and output state(s). An exact Markovian analysis of the element is infeasible. We build an iterative approach in which the probability distribution of the element's states is computed from the distribution in the preceding iteration. Starting from a known initial state of the element, computation is carried out slot by slot until a steady-state is reached. The iterative procedure of computation is captured in the following steps:

1. From $m_t$, compute the distributions of all the possible forward-in vectors $FI_{j_t_1}$, $0 \leq j \leq |FI_{vect}|$ at the beginning of the routing CEC $t_1$. 
2. From the Bernoulli distribution $Ber(w)$ of injection traffic model and the uniform traffic assumption, compute distributions of the injection vector space $INV_{vect}$.

3. Compute the distributions of the queue-in vector space at the beginning of CEC $t_1$. This is the same as the distributions of queue-out vector space at the end of previous routing CEC $(t - 1)_2$, i.e.,

$$Pr[QI_{t_1} = j] = Pr[QO_{(t-1)_2} = j]$$  \hspace{1cm} (4.13)

for all $j$. $0 \leq j \leq |QO_{vect}|$

4. From the three different vector space distributions, compute the input state space $(S)$ distributions.

5. For each input state $s_{t_1}$, $s_{t_1} \in S$, derive the set of output states $\{s'_{t_2}\}$, $s'_{t_2} \in S'$ by applying the tree transformation, discussed later.

6. From the distributions of all the output states $(S' = \{< x, i, j, k, y >\})$ generated at the end of the routing cycle $t_2$, compute cell delivery probability, cell loss probability, and marginal distributions of queue-out and retrograde vector space. The marginal distributions of forward-out vectors is obtained by combining the two subcomponents: forward-success vector and the retrograde vector. The queue-out vector space becomes the queue-in space at the beginning of the next cycle, as mentioned earlier. The delivery probability and the loss probability values are added together and the difference between this sum and the offered injection load is compared against a convergence criteria to decide whether the iterative solution has reached a steady-state or not.

7. From the distributions of forward-out vector space $FO_{vect}$, compute the link utilization vector $m_{(t+1)}$ for cycle $t + 1$. 
Initially, the switch is empty. Therefore, \( m_0 = 0 \). The queue-in buffers as well as the forward-in buffers are also empty. Therefore,

\[
Pr[Q_{I_0} = 0] = Pr[F_{I_0} = 0] = 1.0
\]

(4.14)

where the subscript 0 indicates the start slot of the iteration and the index 0 indicates the zero-th vector which is an empty vector. Only the injection vector space will have some non-zero values at the beginning of CEC 0. Starting from these known initial conditions of the element, the iterative approach keeps computing the distributions of the state space in each CEC. When the relative variation of the injection load with the sum of the cell delivery probability and cell loss probability across consecutive iterations become less than a threshold, we say that steady-state has been reached. From the steady-state probability distributions, one can easily evaluate the various switch performance parameters.

### 4.3.2 Tree Transformation

Tree transformation is perhaps the most important step of the numerical algorithm in which we transform an input state into one or more output states. The number and classes of the cells to be routed are present in the three constituent components of an input state, namely, a queue-in vector, a forward-in vector and an injection vector. Cells are routed one by one starting from the queue-in vector and with minimum distance first. The output state which is initially empty at the start of switching undergoes state transitions as each cell is routed. The transitions can be labeled Markov because the new state or states generated depend on the current state and the class and buffer identity of the cell. When all the cells of the input state are routed, we get a set of output states the sum of whose probability values equals that of the input state.

Let

- \( s_{t_1} \) = state of a switching element at the beginning of a CEC \( (t_1) \) \( s_{t_1} \in S \)
\[ s'_{t_2} = \text{state of a switching element at the end of a CEC } (t_2 \text{). } s'_{t_2} \in S' \]

\[ p_{t_1}(s) = \text{probability that the state at time } t_1 \text{ is } s_{t_1} \]

\[ p'_{t_2}(s') = \text{probability that the state at time } t_2 \text{ is } s'_{t_2} \]

\[ u_{t_2}(s'|s) = \text{probability that the element makes a transition from state } s = s_{t_1} \text{ to } s' = s'_{t_2} \]

At the beginning of a CEC \( t_1 \),

\[
\sum_{s \in S} p_{t_1}(s) = 1 \quad (4.15)
\]

Since it is possible to generate the same output state \( s'_{t_2} \) from a different input state \( s_{t_1} \), we can write:

\[
p'_{t_2}(s') = \sum_{s \in S} p_{t_1}(s) \ast u_{t_2}(s'|s) \quad (4.16)
\]

\[
(4.17)
\]

At the end of the CEC \( t_2 \) when we have all the output states \( \{s'\} \) generated

\[
\sum_{s' \in S'} p'_{t_2}(s') = 1 \quad (4.18)
\]

Tree diagrams often provide a convenient means of ascertaining transformation relations between states. It consists of certain nodes and edges connecting the parent and the child nodes. A node in the tree (except the root node) indicates one of the possible outcomes as a result of routing a cell. Depending on the current state of the switching element (as a result of routing one or more cells), the cell to be routed (i.e., its class. and buffer identity). and the routing algorithm running in the switching element, a node may generate one or more child nodes. Since each level of the tree corresponds to the routing of a cell, the maximum depth of the tree will be equal to the total number of cells to be routed present in all three component vectors of
the input state. The edges connecting the parent and the child nodes in the tree are associated with probability values. The sum of the probabilities of all the child nodes should be equal to that of the parent node itself and the sum of all branch probabilities or edge probabilities should be equal to unity. The root of the tree corresponds to the empty output state of the element when no cells have been routed and as a result, the delivery buffers, the forward-out buffers and the queue-out buffers are all vacant. A transformation tree for an input state in a dimension 3 hypercube is shown in Figure 4.2.

Figure 4.2: Probability Tree Showing the Generation of States

As shown in the figure, there are altogether 7 cells in the input state and the tree grows to a depth of 7. Not all cells can be assigned their desired outgoing links. There may be routing conflicts. The conflict probability will be a function of the current output state of the node (i.e., how many cells before the cell in consideration, also known as the tagged cell, have already been assigned the links) and the hopcount.
value of the cell. We adopt Greenberg and Hajek's approach [24] to compute the conflict probability of a tagged cell.

Let's assume that the tagged cell, which is \( i \) hops away from its destination, is in the queue-in buffer of the element. The cell must compete with other queued-in cells \( \min\{U_0, (d - 1)\} \) where \( U_0 = U - 1 \) and \( U \) is the total number of queued-in cells at a node. Let us assume that all the queued-in cells are cells with non-zero routing code. The number \( R \) of other cells already assigned before the tagged cell is equally likely to be any of the values in \( \{0, 1, \ldots, \min\{U_0, (d - 1)\}\} \). The tagged cell fails to be forwarded if and only if the \( i \) links it prefers are all in the set of \( R \) links already assigned to other cells. Therefore, the probability that the cell suffers a conflict is given by

\[
P_{\text{conflict}} = \left( \frac{j}{d} \right) \left( \frac{j - 1}{d - 1} \right) \cdots \left( \frac{j - i + 1}{d - i + 1} \right)
\]

(4.19)

or equivalently \( P_{\text{conflict}} = (\frac{j}{d},) \), where \( R = j, j_i = j(j - 1) \cdots (j - i - 1) \), and \( d_i = d(d - 1) \cdots (d - i + 1) \). Therefore, the probability that a cell is successfully assigned is given by \( 1.0 - P_{\text{conflict}} \). The conflict probability becomes the queuing probability when buffer space is available in the queue, becomes the retrograde probability when queue is full, but forward buffers are available, and becomes the loss probability when neither queue buffers nor forward buffers are available.

In the case of cell loss, the probability tree does not grow any further on that outcome branch, because the cell is dropped from consideration for any more routing outcomes. The node becomes the leaf node in that case. In the example figure, we do not have any such cell loss situation. Seven output states (B, C, D, E, F, G, and H) have been generated as a result of routing the cells contained in the input state. All these output states are marked as the leaf nodes of the tree, the sum of the probability values of which is equal to that of the input state itself.

The probability tree might generate leaf nodes that are not distinct. For example, states C and E in our example tree are identical. These two states can therefore be
coalesced into a single state by adding the two probability values. Such sharing of the nodes reduces the state space considerably.

4.3.3 Computation of Marginal Distributions

Given the distribution of the output state space, one can compute marginal distributions of all the constituent components. The probability \( \pi_{t_2}(x) \) such that \( 0 \leq x \leq (d + 1) \) cells are delivered at the end of the routing CEC \( t_2 \), is given by:

\[
\pi_{t_2}(x) = \sum_{Del=x}^{D} \sum_{m} \sum_{i} \sum_{n} \sum_{L} p'_{t_2}(<Del_{t_2}, FS_{mt_2}, QO_{it_2}, RO_{nt_2}, L_{t_2}>) \tag{4.20}
\]

The average number of cells delivered (\( \bar{\pi}_{t_2} \)) at the end of the routing CEC is given by:

\[
\bar{\pi}_{t_2} = \sum_{x=0}^{x=d-1} x \pi_{t_2}(x) \tag{4.21}
\]

Similarly, the probability that \( 0 \leq y \leq 1 \) cells are lost. is given by:

\[
Pr[L_{t_2} = y] = \sum_{Del} \sum_{m} \sum_{i} \sum_{n} \sum_{L=y} p'_{t_2}(<Del_{t_2}, FS_{mt_2}, QO_{it_2}, RO_{nt_2}, L_{t_2}>) \tag{4.22}
\]

Therefore, the average number of cells lost \( L_{t_2} \) is given by:

\[
L_{t_2} = \sum_{y=0}^{y=1} y Pr[L_{t_2} = y] \tag{4.23}
\]

The distributions of the queue-out vector space are derived from the following equations:
\[ Pr[QO_{t_2} = q] = \sum_{\text{Del}} \sum_{m} \sum_{i=q}^{n} \sum_{L} p'_{t_2}(<\text{Del}_{t_2}, FS_{mt_2}, QO_{it_2}, RO_{nt_2}, L_{t_2}>) \] (4.24)

The distributions of the forward-out vector are important for estimating the link utilization vector \( m_{t-1} \). Forward-out vectors are obtained by combining the two vectors \( FS_{t_2} \) and \( RO_{t_2} \) of any output state. Given that some output state has \( m \)-th forward-success vector and \( n \)-th retrograde vector, one can obtain \( p \)-th forward-out vector \( (FO_{pt_2}) \) by adding for each class the corresponding number of cells of the two vectors. This gives rise effectively to a 4-tuple representation of an output state space \(<\text{Del}_{t_2}, FO_{pt_2}, QO_{it_2}, L_{t_2}>)\). The distributions of forward-out vectors can now be computed from:

\[ Pr[FO_{t_2} = u] = \sum_{\text{Del}} \sum_{m=u} \sum_{i} \sum_{L} p'_{t_2}(<\text{Del}_{t_2}, FO_{mt_2}, QO_{it_2}, L_{t_2}>) \] (4.25)

### 4.3.4 Computations of Link Utilization Vector

Given the distributions of forward-out vectors \( FO_{vect} \) at the end of a routing CEC \( t_2 \), one can compute link utilization vector \( m_{(t-1)} \) \( (m_{(t-1)}(0), \ldots, m_{(t-1)}(d)) \) by the following equations:

\[ m_{(t-1)}(i) = \sum_{j=0}^{j=|FO_{vect}|} n_{ij} \times Pr[FO_{t_2} = j] \times \frac{1}{d} \] (4.26)

for all \( i, 0 \leq i \leq d \), where \( n_{ij} \) indicates the number of cells of class \( i \) that belongs to \( j \)-th forward-out vector.

The component \( \frac{1}{d} \) indicates the probability with which these cells can be assigned to any of the \( d \) outgoing links. Once we have values for all \( m_{(t+1)}(i), 0 \leq i \leq d \), we can compute \( m_{(t+1)} \) as follows:
\[ m_{(t-1)} = \sum_{i=0}^{d} m_{(t-1)}(i) \] (4.27)

The probability that a link will not carry any cell during CEC \( t - 1 \) is given by \( \bar{m} = 1 - m_{(t-1)} \).

### 4.3.5 Convergence Criterion

The iterative approach uses the distribution of a switching element's output states at the end of each switching cycle to update distributions of \( FI_{\text{vect}} \) and \( QI_{\text{vect}} \) at the beginning of the next cycle. This sequence continues until some limiting probability distribution of the input or output states of the element is realized. This point can be identified by a convergence criterion which determines when the numerical computation stops. The equilibrium point could be defined as:

\[
\begin{align*}
|Pr[FI_{mr}] - Pr[FI_{m(r-1)}]| < \text{convergence factor} & \quad (4.28) \\
|Pr[QI_{nr}] - Pr[QI_{n(r-1)}]| < \text{convergence factor} & \quad (4.29) \\
\text{OR} \\
|Pr[FO_{pr}] - Pr[FO_{p(r-1)}]| < \text{convergence factor} & \quad (4.30) \\
|Pr[QO_{qr}] - Pr[QO_{q(r-1)}]| < \text{convergence factor} & \quad (4.31)
\end{align*}
\]

for all \( m \in FI_{\text{vect}}, n \in QI_{\text{vect}}, p \in FO_{\text{vect}}, m \in QO_{\text{vect}} \) respectively. The subscript \( r \) denotes the iteration number. The model developed here is an ergodic Markov chain which is known to converge. Thus at some point in the numerical process Equations 4.28 - 4.31 must become true and continue to be true. In all of our model evaluations, all of the factors in Equations 4.28 - 4.31 were seen to monotonically converge toward an asymptote. This is consistent with what is expected from the ergodic property of the Markov model. Given this theoretical basis and observed convergence, we adopted a simple heuristic for convergence. When \( x = (\text{injection probability} - \text{delivery probability} - \text{loss probability}) \) is less than or equal to the convergence factor, we consider the model evaluation to be complete, and results
were taken.

In detail, each CEC, the delivery probability and the loss probability are compared with the injection load: if the difference becomes less than an acceptable limit called the *convergence factor*, the model is said to have reached its equilibrium point for practical purposes. A convergence factor of 0.0005 is assumed in our analytical results, which gives us good accuracy for all practical purposes. A more stringent factor will unnecessarily increase the run time of the program.

Our practical convergence criteria was based on values (e.g., injection probability) which have much larger absolute values than some of the values measured (e.g., the loss probability). This was acceptable in practice because the two values - although separated widely in magnitude - followed very similar convergence curves. There is abundant practical evidence that the injection loss probability was a smooth function of the injection probabilities, both over time. Thus it is reasonable to trust the injection loss figures.

### 4.4 Summary

In this chapter, we have presented a new model of a hypercube switch running the SCS routing algorithm. Some approximations have been made to develop the model. A single switching element has been modeled to capture a $d$-dimension switch. The iterative steps followed in arriving at the numerical solution of the model have been specified. Once the solution reaches equilibrium the steady-state distribution of the input or output states are used to compute the performance metrics. The numerical results of the model are presented in Chapter 6 in which the accuracy of the model is established by comparing against simulation.

We present another model in the next chapter to capture out-of-orderness phenomenon based on a single *tagged cell* approach.
Chapter 5

Model for Out-of-Orderness

While direct binary hypercube switches hold promise for future B-ISDN applications, they suffer from one drawback. The cells belonging to a particular traffic stream may be delivered at their destinations out of sequence. This phenomenon is known as *out-of-orderness* of cells and is due to queuing and retrograding of the cells on multiple paths from source to sink.

If $r$ is the distance a cell must travel, then there are $r!$ shortest paths from its source to its destination. In the SCS routing algorithm, cells may be queued and/or retrograded as a result of routing conflicts. That means, different cells in the same connection can experience different latencies through the switch. If the variation in latency exceeds the inter-cell time interval then the cells can arrive at their destinations out-of-order. This is a serious problem in a telecommunication switch and must be eliminated or tightly controlled.

One solution that ensures in-order delivery of cells at their destinations is provision of re-ordering hardware at each sink. This re-ordering hardware is a set of $K$ cell buffers and special-purpose control logic. Each cell is assigned a $\lceil \log_2 K \rceil$ bit sequence number which indicates the cell’s relative position in the buffer. Cells are held in the cell buffers until all previous cells arrive and are delivered to the sink. Thus cells within $K$ steps of their proper order are delivered in sequence at a cost of $K$ buffers and $\log_2 K + 1$ bits in each cell. If the out-of-orderness is more than $K$ steps, then the reordering buffer will fail to re-sequence them and in that case the cell is dropped.
It is important to model this out-of-orderness phenomenon and from that model derive some design insights as to what the re-sequencing buffer size ($K'$) should be or what will be the probability of cells coming $K'$ steps or further out of order. It is shown in this chapter that out-of-orderness can be computed from the delay distributions $dd_{ij}$ of a typical cell. The delay distributions refer to the probability distributions with which a typical cell or tagged cell with hopcount $i$ will be delivered in $j$ cycles, where $j = i, i+1, i+2, \cdots \infty$. The value of $i$ can range from 1 to $d$ in a $d$-dimension hypercube.

In this chapter, we first discuss some of the related out-of-order performance models that were developed by previous researchers [17, 22, 36]. The metrics chosen by them were different but they all related to the issue of out-of-sequence arrival of packets. Next, we propose a model to study the out-of-orderness problems encountered in buffered binary hypercube switches. The proposed model needs an estimate of the delay distributions of a tagged cell. The equilibrium probability values of forward-in vectors and the queue-in vectors already described in Chapter 4 are used in deriving these distributions. The delay distribution figures can be used to make an exact derivation of the out-of-orderness figures which we do not do. As far as switch performance is concerned, the most important information is revealed by the distribution figures themselves. Computing exact out-of-orderness figures will be more of an academic exercise with little extra insight into switch behavior. Also, this will bring extra complexity into the analysis with no extra benefits. The validation of model results is carried out in Chapter 7 by verifying against simulation.

5.1 Previous Models

Several researchers have developed re-sequencing models either in connection with optimal distribution of traffic in high speed networks [22], or in connection with load balancing issues in multistage interconnection networks [35], or in connection with performance studies of deflection routing in regular 2-connected networks like Manhattan Street Networks [17]. Most of them have used multi-path routing techniques
to improve the performance of their systems and as a result have faced the unwanted problems of out-of-sequence arrival of packets at their destinations. These models have been primarily concerned with the cost of re-sequecing these packets which is measured by the re-sequecing delay of a tagged packet. This delay suffered by a packet is in addition to the fixed delays. and the queuing delay. The models developed by some of these researchers are presented in the following paragraphs.

5.1.1 Gogate and Panwar's Model

When there are multiple parallel paths between a source and a destination, packets may arrive out-of-sequence at their destinations because of variable delay along the paths. The traditional models capture this out-of-sequence phenomenon by associating variable delay along the paths with queuing delay at the network elements. Gogate and Panwar [22] argue that, in some cases, queuing delay is an insignificant component of the total end-to-end delay, particularly when compared to the fixed delay components (propagation + fixed processing) of the paths. This happens when the network bandwidth increases considerably. They advocate modifying the traditional models to cater to those observations.

A simple system of two nodes connected by two disjoint paths, as shown in Figure 5.1, is considered. Each of the paths is assumed to have a different fixed delay (which may be due to difference in physical length or may be due to difference in the number of hops). The queuing delay is modeled by an equivalent exponential service rate and the fixed portion of the delay by a fixed delay component. For the sake of analytical tractability, a Poisson arrival process with an arrival rate λ is assumed. The system of two parallel M/M/1 queues operating at service rates μ1 and μ2 is analyzed. The delay lines T1 and T2 are connected in tandem with server-1 and server-2 respectively: T1 is the amount of fixed delay associated with the upper path and T2 with the lower path. Traffic λ splits up probabilistically, with pλ being routed to the first path and (1 - p)λ with the second path.

A typical customer called the tagged customer (TC) finds the system, upon ar-
rival, in state \((m, n)\), where \(m\) is the number of customers in queue-1 (consisting of customers in buffer-1 and server-1), and \(n\) is the number of customers in queue-2. Because of the FCFS service discipline, the packets which come after the TC can not affect the departure time of TC from the re-sequencing buffer. Only the packets in the system when TC arrived can contribute to the re-sequencing delay it suffers. Also, packets which are ahead of the TC in the queue it joins can not affect TC’s re-sequencing delay. Therefore, the re-sequencing delay \(RD\) is conditioned on the state the TC finds the system in, and the queue the TC joins. The average expected value of \(RD\) is then found out by summing over all possible values of \((m, n)\). Thus,

\[
RD = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} R(m, n) P(m, n) 
\]

where

\[
R(m, n) = pR^1(m, n) + (1 - p)R^2(m, n)
\]

\(R^1(m, n)\) and \(R^2(m, n)\) are the RD conditioned on the TC joining either queue-1 or queue-2 and \(P(m,n)\) is the probability that the system is in state \((m, n)\). It is easy to find \(P(m,n)\) from the M/M/1 Markov Chain analysis. The other two components \(R^1(m, n)\) and \(R^2(m, n)\) are discussed in the following subsections.
Evaluation of $R^1(m, n)$

Suppose the tagged customer (TC) finds the system in state $(m, n)$ upon arrival and joins queue-1. The system state becomes $(m + 1, n)$. If $n = 0$, then the TC does not suffer any delay waiting for missing packets in the re-sequencing buffer. Before TC leaves the queue and arrives at the delay line $T_1$, there needs to be $m + 1$ service completions at queue-1. From the point of view of TC, the departure time of the customer in the $n$-th position in queue-2, at the time of TC's arrival, is critical. During the time $m + 1$ service completions are done in queue-1, if there are are $n$ or more service completions at queue-2, then the re-sequencing delay suffered by the tagged packet is nil. This is because packets that arrive after the TC and join the other queue and get serviced before the TC will have to wait for the TC to arrive at the re-sequencing buffer. That won't cause any delay in the delivery of the tagged packet. But, if TC gets serviced before any of the $n$ packets waiting in queue-2, who were there at the time of TC's arrival, then the TC will have to wait in the re-sequencing buffer for the arrival of the missing packets. This will cause re-sequencing delay in the delivery of the tagged packet. If the number of service completions in the second server is $i$, then one can easily compute the probability of $m$ service completions at queue-1. $i$ service completions at queue-2, followed by service completion of the TC. So, there are $n - i$ customers still in queue-2 at TC's service completion time. These $n - i$ customers will have $T$ ($T = T_1 - T_2$) time units to finish their service and therefore not affect TC's departure. On the other hand, if there are $j$ service completions in time $T$, then the TC will have to wait in the re-sequencing buffer for $n - i - j$ service completions at the second server. The re-sequencing delay $R^1(m, n)$ can now be easily computed.

Evaluation of $R^2(m, n)$

If the TC joins queue-2, then depending upon the values of the system state $(m, n)$ and the number of service completions at queue-1, three different cases may arise:

1. System state is $(m, n)$, $m > 0$ and fewer than $m$ service completions at queue-1:
When the TC joins queue-2, the system state becomes \((m, n + 1)\). Before the TC arrives at the re-sequence buffer there have to be \(n + 1\) service completions at queue-2, during which there can be \(i\) service completions at queue-1. If at least one of the \(m\) customers waiting in the queue-1 after the tagged customer gets serviced, it will have a re-sequence delay. If there are only \(i\) service completions at queue-1 \((0 \leq i \leq m - 1\) and \(m > 0\)) then the TC will have to wait in the re-sequence buffer \(\frac{\mu_i}{m-i} + T\) units of time. The expected RD \(R^2(m, n)\) can now be computed summing up over all possible values of \(i\).

2. System state \((m, n)\), \(m > 0\) and exactly \(m\) service completions at queue-1:

If all the \(m\) customers in queue-1 have already been serviced before the TC has finished its service, then depending on the position of the \(m\)-th packet there may or may not be some RD associated with the tagged packet. If the \(m\)-th packet is somewhere in the delay line, then the TC suffers some RD; but if the \(m\)-th packet has already left the system, then there is no delay associated with the tagged packet. Again, following the same procedures mentioned above, one can compute the probability with which the \(m\)-th packet reaches the delay line and \(i\) customers \((0 \leq i \leq n)\) from queue-2 already finish their service and are either in the re-sequence buffer or have left the system. The \((n + 1)\)-st packet is the TC. So, there will still be \(n + 1 - i\) customers in queue-2. The TC will suffer some re-sequence delay if the cumulative service times of all these \(n + 1 - i\) customers is less than \(T\). \(R^2(m, n)\) can then be evaluated from the sum of \(n + 1 - i\) i.i.d. exponential random variables conditioned on the event that the sum is less than \(T\).

3. System state \((0, n)\):

The third possibility of the TC suffering some re-sequence delay occurs if it finds the system to be in state \((0, n)\) when it joins queue-2. The tagged packet waits for a customer which was served in queue-1, but was in the delay line at the instance of TC's arrival. The position of the last customer in the delay line needs to be found to evaluate the expected RD. One can view the position
of the last customer on the delay line as a random point occurring in the idle period of server-1. The idle duration of server-1 is exponentially distributed with parameter \( \lambda p \). If \( x \) is a random variable that represents the portion of the delay line already traversed by the last customer, then \( x \) is exponentially distributed with parameter \( \lambda p \). The re-sequence delay RD will be zero if \( x > T \), i.e., the last customer has already traversed the complete delay line. A random variable \( z \) is defined by \( z = (x + y_1 + y_2 + \cdots + y_{n-1}) \), where \( y_i \)'s are the random variables corresponding to the service times of customers at server-2 and \( x \) corresponds to the portion of the delay line already traversed by the last customer. The RD \( R^2(m,n) \) will be equal to the difference of \( T \) and the expected value of the random variable \( z \), provided the expected value of \( z \) is less than \( T \). multiplied by the probability of \( z \) being less than \( T \).

Once we obtain the values of both \( R^1(m,n) \) and the three different values of \( R^2(m,n) \), \( R^{2a}(m,n) \), \( R^{2b}(m,n) \), and \( R^{2c}(0,n) \) we obtain an expression for average expected re-sequence delay as

\[
RD = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \left[ pR^1(m,n)P(m,n) \right] + (1-p)((R^{2a}(m,n) - R^{2b}(m,n))P(m,n)
R^{2c}(0,n)P(0,n))
\]

(5.3)

5.1.2 Jean-Marie's Model

Alain Jean-Marie [36] studies the re-sequence problem in connection with load balancing issues in multi-stage interconnection networks. Load balancing is achieved by stochastic re-routing of packets among multiple paths, whenever there is congestion. The presence of several paths allows overtaking of packets, and induces an additional cost due to packet re-sequence time, if the ordering of the packets is important. \( R \) is the re-sequence time a packet has to wait before all the packets that have been emitted before it have reached the output. This extra time is the price to pay for achieving load balancing. The natural performance question that arises is whether balancing
the load actually decreases the mean communication time when the re-sequencing
time is included.

\[ \text{Figure 5.2: Re-sequencing Time of a Customer} \]

Consider a tagged packet arriving in the network whose arrival instant is noted at
time \( t = 0 \), as shown in Figure 5.2. The packet suffers a waiting time and service
time equal to \( A \) along its path. There are only two possible paths in the network,
as shown by the two horizontal lines marked \( \text{UP} \) and \( \text{DOWN} \) in the Figure. The
tagged packet has to wait until all the packets that entered before it have traversed
the network. No overtaking is allowed among the packets that choose a particular
path. Therefore, the tagged packet has to wait only for the last packet that entered
before it but took the other path. Packets leave the system as soon as the one that
blocked them arrives. Let \( \tau \) be the time between the arrival of this last packet into
the network and the tagged packet. This packet is called the \textit{challenger} packet. Also,
let \( B \) be the delay experienced by this challenger packet. The exit time taken by this
challenger packet is \( B - \tau \). Therefore, the re-sequencing delay suffered by the tagged
packet is given by:

\[
\begin{align*}
0 & \quad \text{if } A \geq B - \tau \\
B - \tau - A & \quad \text{if } A \leq B - \tau 
\end{align*}
\]

(5.1)

(5.5)

From the distributions of \( A \) and \( B \), one can now compute the distributions of the
re-sequencing delay \( R \).
5.1.3 Choudhury’s Model

Choudhury [17] studies the re-sequence problem of the packets arriving at their destinations in a performance study of buffered deflection routing in regular topology networks like Manhattan Street Networks, ShuffleNet and Shuffle Exchange. Deflection routing schemes cause packets to take multiple paths to the destination, as a result of which packets belonging to a particular traffic stream may arrive out of sequence. In order to put these packets in sequence a re-assembly buffer is used at the destination node. All previous work on deflection routing assumes that the receiver has an infinite re-assembly buffer to put all the received packets back in sequence. Choudhury studies the effect of finite re-assembly buffer on the performance of deflection routing in the form of packet loss probability. A packet is dropped when it arrives too far out of sequence.

The packet loss probability is also defined as the probability of overflow. Dropping of a packet may initiate retransmission from the source and therefore, all subsequent packets from the source are dropped by the receiving node until the first missed packet is received. A simple mathematical model is provided to estimate the probability of overflow of the re-assembly buffer.

Consider a particular source-destination pair \((s, w)\) where the source \(s\) transmits at an average rate of \(R_s\). Suppose the source transmits the \(i\)-th packet in slot \(u_i\), where \(u_i = u_{i-1} + t_i\), where \(t_i\) is the number of slots between successive transmissions. The \(t_i\)'s are assumed to be independent, identically distributed \((i.i.d.)\) random variables with mean \(E[t_i] = E[t] = 1/R_s\) and variance \(Var[t_i] = Var[t]\). Let \(r_i\) be the slot in which the \(i\)-th packet arrives at the receiver, where \(r_i = u_i + P_{sw} + \Delta_i\), where \(P_{sw}\) is the constant component equal to the length of the shortest path between the source \(s\) and the destination \(w\) and the component \(\Delta_i\) is the random component equal to the additional delay experienced by the \(i\)-th packet in traveling from the source to the destination. The distribution of \(\Delta\) depends on the source-destination pair. \(\{\Delta_i\}\) are assumed to be \(i.i.d.\) random variables with mean \(E[\Delta]\) and variance \(Var[\Delta]\). Let the re-assembly buffer be of size \(K_r\).
The re-assembly buffer is assumed to adopt a conservative buffering policy and accept only packets bearing a sequence number that is within $K_r$ of the last in-sequence packet. When a packet arrives too far out of sequence, a buffer overflow occurs and the packet is dropped. Therefore, the $i$-th packet will be dropped if it arrives before any of the packets with sequence number $(i - K_r)$ or less, i.e., if $r_i < r_j$, for all $0 \leq j \leq i - K_r$. The probability that the re-assembly buffer overflows when the $i$-th packet is received is given by:

$$P_{ov}(i) = Pr\{ \bigcup_{j=K_r}^{i} (z_j < 0) \} \quad (5.6)$$

where

$$z_j = r_i - r_{i-j} = \Delta_i - \Delta_{i-j} + \sum_{l=i-j-1}^{i} t_l. \quad (5.7)$$

Since the $\{\Delta_i\}$ are identically distributed independent random variables, the random variable $z_j$ has a mean $E[z_j] = \lfloor j/R_s \rfloor$ and a variance $\text{Var}[z_j] = 2\text{Var}[\Delta] + j\text{Var}[t]$ and is independent of $i$. Therefore, the overflow probability should be a function of the re-assembly buffer size, $K_r$, the random component of delay through the network, $\Delta$ and the source rate, $R_s$.

### 5.2 Proposed Model

This section deals with the model we build to study the out-of-orderness phenomenon in buffered binary hypercube switches. There is no apparent simple and direct method to compute the out-of-orderness parameters in hypercube switches which have queuing buffers and which use deflection routing. The proposed model uses the steady-state distribution of cells in forward-in buffers and the queue-in buffers derived analytically in Chapter 4. Unlike Chapter 4, where we considered a one node model of a $d$-dimension hypercube switch, we present in this section a one cell model.
A single cell, also called a tagged cell, when accepted, performs a walk in the network from its source to its destination. At each node, en route to its destination, the cell may be in the forward-in buffer of the node as a result of successful forwarding or retrograding, or may be in the queue-in buffer of the node. If the tagged cell has an initial hopcount equal to \( i \), then the minimum latency i.e., minimum number of CECs needed by the cell before it gets delivered is \( i \). However, queueing and retrograding may cause extra latency in the delivery of the tagged cell. The probability distribution of a tagged cell with an initial hopcount \( i \) taking \( j \) cycles, \( i \leq j \) before being delivered to its destination is called a delay distribution \( dd_{ij} \).

Consider a particular source-destination pair in a hypercube switch. Because of queueing and retrograding, the successive cells generated at the source could arrive out-of-sequence at the destination. A re-assembly buffer at the destination is able to put all these cells in proper sequence, if the buffer size is assumed infinite. However, a finite re-assembly buffer will drop or mis-order a cell when it arrives too far out of sequence.

Suppose the source element of the pair injects an \( n \)-th cell into the switch in timeslot \( s_n^i \), where \( i \) is the minimum distance between the source and the destination. We assume that \( s_n^i = s_{n-1}^i + 1 \) meaning that the \( (n-1) \)-th cell and \( n \)-th cell are injected in consecutive time slots, which need not be true. A more general relationship is given by [17] \( s_n^i = s_{n-1}^i + t_x \). where \( t_x \) is the number of slots between successive transmissions and is a random variable with mean \( E[t_x] \) and variance \( Var[t_x] \). In what follows, we consider only the worst case in which cells from the tagged conversation are injected on every CEC. This approach will allow statements to be made about worst case behaviour. Let the \( n \)-th cell be delivered at the receiver in timeslot \( r_n^i = s_n^i + i + \delta_n^i \), where \( i \) is the minimum constant component of the delay required, and \( \delta_n^i \) is the additional random delay experienced by the \( n \)-th cell (with hopcount \( i \)) in traveling from the source to the destination. This additional component of delay is caused by queueing and retrograding of the cell. The delay experienced by the \( n \)-th cell is given by \( d_n^i = i + \delta_n^i \). Similarly, the delay experienced by \( (n-1) \)-th cell is given by \( d_{n-1}^{i-1} = i + \delta_{n-1}^i \).
If \( d_i^n - d_i^{n-1} < 0 \) then the two consecutive cells will arrive out-of-order at their destinations. The \( n \)-th cell will have to wait for arrival of the \((n-1)\)-th cell. Since, \( d_i^n - d_i^{n-1} = \delta_i^n - \delta_i^{n-1} \), it is the non-zero negative difference between the two identically distributed random variables that contribute to the out-of-orderness. The degree of out-of-orderness between two cells can be defined as the magnitude of this difference which may range from 1 to \( \infty \).

When the \( n \)-th cell arrives at its destination before some cell having sequence number smaller than \( n \), i.e., \( n - 1, n - 2, n - 3, \ldots \), we have an out-of-order situation. If the re-assembly buffer size at the destination is infinite, then it can re-sequence any out-of-order cell. But, if the buffer size is finite, say, equal to \( K \), then any cell arriving at the buffer that is not within \( K \) of the last in-sequence cell is dropped [17]. i.e., if \( r_n^i < r_j^i \) for all \( 0 \leq j \leq n - K \), then the \( n \)-th cell is dropped. Following Choudhury's arguments, we write:

\[
P_{\text{dropped}}(n) = Pr\{ \bigcup_{j=K}^{n} (z_j < 0) \}
\]  

(5.8)

where

\[
z_j = r_n - r_{n-j} = \delta_i^n - \delta_i^{n-j} + j
\]  

(5.9)

The random variable \( z_j \) depends on the random component of the delay through the switch \( \delta_i^n \). If we know the delay distributions \( dd_{ij} \) of a cell, we can estimate the out-of-orderness parameters for two cells easily. However, an exact derivation of out-of-orderness could be extracted from this thesis work in a straightforward extension. The following section shows how to compute delay distributions \( dd_{ij} \) of a tagged cell.
5.2.1 Delay Distributions

A new cell in the injection buffer of a source element may suffer four possible consequences in the first CEC: (a) it may not be accepted by the network and hence be dropped or lost. (b) it may be accepted and successfully forwarded. (c) it may be accepted but because of contention may be queued, and finally (d) it may be accepted but because of contention and no queue space, may be retrograded. The hopcount of the new cell (i) may be any value between 0 and d. We assume that a new cell with an initial distance 0 is delivered locally and is not injected into the network. The four probable outcomes of a tagged cell in injection buffer are $P_{\text{loss}}(i)$, $P_{\text{fwd}}(i)$, $P_{\text{que}}(i)$, and $P_{\text{retro}}(i)$ respectively, where the superscript $\text{Inj}$ signifies that the considered cell is in injection buffer. Figure 5.3 shows a tree representation of the routing outcomes.

![Cell in Injection Buffer Diagram](image)

Figure 5.3: Routing Outcomes of a Cell in Injection Buffer

A tagged cell in the queue-in buffer or in a forward-in buffer does not suffer any injection loss. Similarly, a cell in the queue-in buffer will never be retrograded because of the highest priorities assigned to the cells in queue-in buffers. Figure 5.4 shows the two possible outcomes ($P_{\text{que}}(i)$ and $P_{\text{fwd}}(i)$, $1 \leq i \leq d$) of a tagged cell in queue-in buffer. If the number of delivery buffers at each node is assumed to be infinite, there is no chance a distance zero cell will be stored in the queue-in buffer and therefore $P_{\text{que}}(0) = P_{\text{fwd}}(0) = 0$.

Since cells in forward-in buffers with distance zero hopcounts are removed from the network, the probabilities $P_{\text{que}}(0) = P_{\text{fwd}}(0) = P_{\text{retro}}(0) = 0$. The superscript $\text{Forward}$ indicates that the tagged cell is in forward-in buffer of a node at the time of routing. Again, the corresponding tree representation is shown in Figure
5.5.

![Figure 5.4: Routing Outcomes of a Cell in Queue-in Buffer](image)

![Figure 5.5: Routing Outcomes of a Cell in Forward-in Buffer](image)

The various probability figures with which a tagged cell moves in the network are conditioned not only on the type of the buffers (i.e., injection, forward-in, or queue-in) the cell is in at the time of routing, but also on the number and classes of other interfering non-tagged cells that are present along with the tagged cell. For example, for a $d$-dimension hypercube with a queue-in buffer size equal to $Q$, the number of cells in the queue-in buffer may range between 0 and $Q$ and the classes of cells may range between 1 and $d$. The set of possible cell-class combinations, also called vectors constitute a vector space for the queue-in buffer. A generic tuple representation for, say, $k$-th queue-in vector is given as $< n_k : n_{0k} n_{1k} n_{2k} \cdots n_{dk} >$ where $n_k = \sum_{i=0}^{d} n_{ik}$, is the total number of cells in the queue-in buffer, and $n_{0k}, n_{1k}, \cdots$ represent the number of cells belonging to class 0, class 1, and so on. The cardinality of the vector space is given by $|Q_{vec}|$. The probability with which a particular queue-in vector ($Q_{I_k}$) occurs is obtained from the steady-state distributions.
of the queue-in vector space derived in Chapter 4. The probability of having a queue-
in cell with hopcount \( i \) \( (q(i)) \), \( 1 \leq i \leq d \) is given by the sum of the probabilities of all the queue-in vectors that contain cell (or cells) with hopcount \( i \). Assume that the \( k \)-th queue-in vector occurs with a probability value \( Pr[QI_k] \). The conditional probability \( Pr[QI_k'] \) that the vector \( QI_k \) will contain a tagged cell of hopcount \( i \) is given by the ratio of \( Pr[QI_k]/q(i) \).

If

\[
P_{\text{que}}(i, k) \quad \overset{\text{def}}{=} \quad Pr[\text{a tagged cell in k-th queue-in vector and with hopcount } i \\
\text{is queued again as a result of routing}]
\]

\[
P_{\text{fow}}(i, k) \quad \overset{\text{def}}{=} \quad Pr[\text{a tagged cell in k-th queue-in vector and with hopcount } i \\
\text{is forwarded successfully as a result of routing}]
\]

then, we can write:

\[
P_{\text{que}}(i) = \sum_{k=0}^{k=QI_{\text{vect}}} P_{\text{que}}(i, k) \times Pr[QI_k'] \quad (5.10)
\]

\[
P_{\text{fow}}(i) = \sum_{k=0}^{k=QI_{\text{vect}}} P_{\text{fow}}(i, k) \times Pr[QI_k'] \quad (5.11)
\]

The values of \( P_{\text{que}}(i, k) \) and \( P_{\text{fow}}(i, k) \) also depend on the position \( y \), \( 1 \leq y \leq n_k \) of the tagged cell in the \( k \)-th queue-in vector in order of routing considerations. If the tagged cell is the first one considered for routing, the probability that it will be successfully forwarded is 100%. However, if the cell happens to be in any other position and the hopcount of the cell is lower than the number of outgoing links already assigned, then there is a chance the cell will have contention and will be queued instead of being forwarded. Figure 5.6 shows how a tagged cell in a queue-in vector \( <3:0300> \) might be routed depending on the position of the tagged cell in terms of routing sequence.

Let

Figure 5.6: An Example of Computing Routing Outcomes of a Tagged Cell in a Queue-in Buffer

\[ P_{\text{que}}(i, k, y) \overset{\text{def}}{=} \text{Pr}[\text{a tagged cell in } k\text{-th queue-in vector at position } y \text{ and with hopcount } i \text{ is queued again as a result of routing}] \]

\[ P_{\text{fow}}(i, k, y) \overset{\text{def}}{=} \text{Pr}[\text{a tagged cell in } k\text{-th queue-in vector at position } y \text{ and with hopcount } i \text{ is forwarded successfully as a result of routing}] \]

then, we can write:

\[ P_{\text{que}}(i, k) = \frac{1}{n_{ik}} \sum_{y=1}^{y=n_k} P_{\text{que}}(i, k, y) \quad (5.12) \]

\[ P_{\text{fow}}(i, k) = \frac{1}{n_{ik}} \sum_{y=1}^{y=n_k} P_{\text{fow}}(i, k, y) \quad (5.13) \]

Substituting the values of \( P_{\text{que}}(i, k) \) and \( P_{\text{fow}}(i, k) \) in Equations 5.10 and 5.11 above, the average probability values \( P_{\text{que}}(i) \) and \( P_{\text{fow}}(i) \) can be determined.

Figure 5.7 gives another example of a queue-in vector \(<3:0210>\) where there are two cells of distance 1 and one cell of distance 2. If the tagged cell is a distance 1 cell then the probability that a distance 1 cell is forwarded is given by \((1.0 + 2/3)/2 = 5/6\) and that it is queued is given by \((0.0 + 1/3)/2 = 1/6\). On the other hand, if the tagged cell is the distance 2 cell then the corresponding probabilities are given by \((4/9 + 1/3) = 7/9\) and \(2/9\).
Figure 5.7: Another Example of Computing Routing Outcomes of a Tagged Cell in a Queue-in buffer

In the case of a tagged cell in a forward-in buffer, the parameters $P_{\text{que}}^{\text{Forward}(i)}$, $P_{\text{fwd}}^{\text{Forward}(i)}$, and $P_{\text{retro}}^{\text{Forward}(i)}$ depend not only on the particular forward-in vector $FI_k$ in which the tagged cell occurs, but also on the set of all queue-in vectors $\{QI_l\}, 0 \leq l \leq |QI_{\text{vec}}|$ that might occur with the forward-in vector. Since, cells in queue-in buffers have routing priority over those in forward-in buffers, the probable routing outcomes of the tagged cell in forward-in buffer are influenced by the queue-in vector in consideration. The probability of having a forward-in cell with hopcount $i (f(i)), 1 \leq i \leq d$ is given by the sum of the probabilities of all the forward-in vectors that contain cell with hopcount $i$. Assume that the $k$-th forward-in vector occurs with a probability value $Pr[FI_k]$. The conditional probability $Pr[FI_k]$ that the vector $FI_k$ will include a tagged cell of hopcount $i$ given by the ratio of $Pr[FI_k]/f(i)$.

Therefore, if

\[ P_{\text{fwd}}^{\text{Forward}(i,k,l)} \overset{\Delta}{=} Pr[\text{a tagged cell in } k\text{-th forward-in vector with hopcount } i \text{ is forwarded successfully in the presence of } l\text{-th queue-in vector}] \]

\[ P_{\text{que}}^{\text{Forward}(i,k,l)} \overset{\Delta}{=} Pr[\text{a tagged cell in } k\text{-th forward-in vector with hopcount } i \text{ is queued as a result of routing in the presence of } l\text{-th queue-in vector}] \]

\[ P_{\text{retro}}^{\text{Forward}(i,k,l)} \overset{\Delta}{=} Pr[\text{a tagged cell in } k\text{-th forward-in vector with hopcount } i \text{ is retrograded in the presence of } l\text{-th queue-in vector}] \]

then one can write:
\[ P_{\text{Forward}}^{\text{fwd}}(i) = \sum_{k=0}^{k=|FI_{\text{vect}}|} \left\{ \sum_{l=0}^{l=|QI_{\text{vect}}|} P_{\text{Forward}}^{\text{fwd}}(i, k, l) \times Pr[QI_i] \times Pr[F_{k,l}] \right\} \quad (5.14) \]

\[ P_{\text{Forward}}^{\text{que}}(i) = \sum_{k=0}^{k=|FI_{\text{vect}}|} \left\{ \sum_{l=0}^{l=|QI_{\text{vect}}|} P_{\text{Forward}}^{\text{que}}(i, k, l) \times Pr[QI_i] \times Pr[F_{k,l}'] \right\} \quad (5.15) \]

\[ P_{\text{Forward}}^{\text{retro}}(i) = \sum_{k=0}^{k=|FI_{\text{vect}}|} \left\{ \sum_{l=0}^{l=|QI_{\text{vect}}|} P_{\text{Forward}}^{\text{retro}}(i, k, l) \times Pr[QI_i] \times Pr[F_{k,l}'] \right\} \quad (5.16) \]

where \( Pr[QI_i] \) represents the steady-state probabilities with which the \( l \)-th queue-in vector occur. Since the two vector spaces are independent, the joint probability of occurrence of \( l \)-th queue-in vector and \( k \)-th forward-in vector is given by the product of the two individual probability values:

\[ Pr[QI_i = m; FI_{t_i} = n] = Pr[QI_i = m] \times Pr[FI_{t_i} = n] \]

The parameters \( P_{\text{Forward}}^{\text{fwd}}(i, k, l) \), \( P_{\text{Forward}}^{\text{que}}(i, k, l) \), and \( P_{\text{Forward}}^{\text{retro}}(i, k, l) \) depend on the position \( y \), \( 1 \leq y \leq n_k \) of the tagged cell in the \( k \)-th forward-in vector in order of routing preferences. Therefore, if \( P_{\text{Forward}}^{\text{fwd}}(i, k, l) \), \( P_{\text{Forward}}^{\text{que}}(i, k, l) \), and \( P_{\text{Forward}}^{\text{retro}}(i, k, l) \) represent the corresponding probability values conditioned on the position \( y \) of the tagged cell, then:

\[ P_{\text{Forward}}^{\text{fwd}}(i, k, l) = \frac{1}{n_{tk}} \times \sum_{y=1}^{y=n_k} P_{\text{Forward}}^{\text{fwd}}(i, k, l, y) \quad (5.17) \]

\[ P_{\text{Forward}}^{\text{que}}(i, k, l) = \frac{1}{n_{tk}} \sum_{y=1}^{y=n_k} P_{\text{Forward}}^{\text{que}}(i, k, l, y) \quad (5.18) \]

\[ P_{\text{Forward}}^{\text{retro}}(i, k, l) = \frac{1}{n_{tk}} \sum_{y=1}^{y=n_k} P_{\text{Forward}}^{\text{retro}}(i, k, l, y) \quad (5.19) \]

Routing outcomes of a tagged cell in injection buffers (\( P_{\text{loss}}^{\text{inj}}(i), P_{\text{fwd}}^{\text{inj}}(i), P_{\text{que}}^{\text{inj}}(i), \) and \( P_{\text{retro}}^{\text{inj}}(i) \)) are dependent on the particular injection vector in which the cell occurs and also on the set of all queue-in vectors \( \{QI_m\}, 0 \leq m \leq |QI_{\text{vect}}| \) and forward-in
vectors \{FI_l\} \leq l \leq |FI_{ext}| that might occur with the particular injection vector. The probability of having an injection cell with hopcount \(i\) (\(I(i)\)), \(1 \leq i \leq d\) is given by the sum of the probabilities of all the injection vectors that contain cell(s) with hopcount \(i\). Again, assuming that the \(k\)-th injection vector occurs with a probability value \(Pr[I_{nj_k}]\), the conditional probability \(Pr[I_{nj_k}']\) that the vector \(I_{nj_k}\) will include a tagged cell of hopcount \(i\) is given by the ratio of \(Pr[I_{nj_k}]/I(i)\). Again, assuming independence of all the three vector spaces, one can write:

\[
P_{\text{fwd}}^{I_{nj}}(i) = \sum_{k=I_{nj \text{vec}}}^{k=I_{nj \text{vec}}} \sum_{l=I_{vec}}^{l=Q_{vec}} \sum_{m=0}^{|Q|_m} P_{\text{fwd}}^{I_{nj}(i, k, l, m)} \times \frac{Pr[Q_I_m]}{Pr[F_I_l]} \times \frac{Pr[I_{nj_k}]}{Pr[I_{nj}'_k]}
\]

(5.20)

\[
P_{\text{que}}^{I_{nj}}(i) = \sum_{k=I_{nj \text{vec}}}^{k=I_{nj \text{vec}}} \sum_{l=I_{vec}}^{l=Q_{vec}} \sum_{m=0}^{|Q|_m} P_{\text{que}}^{I_{nj}(i, k, l, m)} \times \frac{Pr[Q_I_m]}{Pr[F_I_l]} \times \frac{Pr[I_{nj_k}]}{Pr[I_{nj}'_k]}
\]

(5.21)

\[
P_{\text{retro}}^{I_{nj}}(i) = \sum_{k=I_{nj \text{vec}}}^{k=I_{nj \text{vec}}} \sum_{l=I_{vec}}^{l=Q_{vec}} \sum_{m=0}^{|Q|_m} P_{\text{retro}}^{I_{nj}(i, k, l, m)} \times \frac{Pr[Q_I_m]}{Pr[F_I_l]} \times \frac{Pr[I_{nj_k}]}{Pr[I_{nj}'_k]}
\]

(5.22)

\[
P_{\text{loss}}^{I_{nj}}(i) = \sum_{k=I_{nj \text{vec}}}^{k=I_{nj \text{vec}}} \sum_{l=I_{vec}}^{l=Q_{vec}} \sum_{m=0}^{|Q|_m} P_{\text{loss}}^{I_{nj}(i, k, l, m)} \times \frac{Pr[Q_I_m]}{Pr[F_I_l]} \times \frac{Pr[I_{nj_k}]}{Pr[I_{nj}'_k]}
\]

(5.23)

\(\forall i, 1 \leq i \leq d\)

The probability values \(P_{\text{outcome}}^{I_{nj}(i, k, l, m)}\), \(\text{outcome} \in \{\text{fow, que, retro, loss}\}\) can be determined from \(P_{\text{outcome}}^{I_{nj}(i, k, l, m, y)}\), \(1 \leq y \leq n_k\).

The walk of a tagged cell from injection buffer of a generating source element till the delivery buffer of the destination element, where the cell is finally absorbed, can be statistically captured by a discrete time Markov chain \(L\) as shown in Figure 5.8. The latency count of the cell changes every CEC whereas hopcount of the cell may
or may not change depending on the type of buffer into which the cell moves. For example, if the tagged cell is queued as a result of routing, hopcount does not change, but latency increases by one. A cell with an initial distance \( i \) will consume at least \( i \) cycles before being delivered. If the cell consumes \( j \) cycles before being delivered, the extra count \( j - i \) is caused by queueing and retrograding of the tagged cell.

Let

\[
\begin{align*}
d_{i,k,t} &= \Pr[\text{a cell with initial hopcount } i \text{ is distance } k \\
&\quad \text{away from its destination at the beginning of routing cycle } t] \\
P_{t,k,fwd,t} &= \Pr[\text{a cell with initial hopcount } i \text{ and distance } k \\
&\quad \text{away from its destination is forwarded successfully during time } t] \\
P_{t,k,que,t} &= \Pr[\text{a cell with initial hopcount } i \text{ and distance } k \\
&\quad \text{away from its destination is queued during time } t] \\
P_{t,k,retro,t} &= \Pr[\text{a cell with initial hopcount } i \text{ and distance } k \\
&\quad \text{away from its destination is retrograded during time } t]
\end{align*}
\]

then the dynamics of the Markov chain \( L \) can be captured by the following equation:

\[
\begin{align*}
d_{i,k,t-1} &= d_{i,k-1,t} \times P_{t,k-1,fwd,t} + d_{i,k,t} \times P_{t,k,que,t} + d_{i,k-1,t} \times P_{t,k-1,retro,t} \\
d_{i,d,t-1} &= d_{i,d,t} \times P_{t,d,que,t} + d_{i,d-1,t} \times P_{t,d-1,retro,t}
\end{align*}
\]

\( \forall i, 1 \leq i \leq d \)

The parameters \( P_{t,k,fwd,t}, P_{t,k,que,t}, \) and \( P_{t,k,retro,t} \) represent the average forwarding, queuing, and retrograding probabilities not conditioned on the type of buffers.

Theoretically, the value of \( j \) in \( dd_{ij} \) can be as large as infinity, the reason being the cell may be queued and/or retrograded ad infinitum. However, for SCS algorithm running in the switch, it can be shown the probability that a cell requires greater than some finite number CECs is vanishingly small. A reasonable upper bound on the maximum number of clock cycles (i.e., maximum value of \( j \)) is taken as \( 4 \times d \).
Figure 5.8: Markov Chain Showing the Walk of a Tagged Cell

where \( d \) is the dimension of the hypercube. Many simulations have shown that \( 4 \times d \) is an extremely high cell latency.

The equilibrium values of \( dd_{ij} \), where \( i \leq j \leq 4 \times d \), can be determined by using an iterative approach. Let

\[
dd_{i,j,k} = \Pr[\text{a tagged cell with initial distance } i \text{ requires } j \text{ cycles before reaching its destination, at iteration } k]
\]

\[
P_{i,fwd} = \Pr[\text{a tagged cell with initial distance } i \text{ is forwarded}]
\]

\[
P_{i,que} = \Pr[\text{a tagged cell with initial distance } i \text{ is queued}]
\]

\[
P_{i,retro} = \Pr[\text{a tagged cell with initial distance } i \text{ is retrograded}]
\]

Then, for each value of \( i \), and \( j \) varying from 0 to \( 4 \times d \), one can write:

\[
\begin{align*}
    dd'_{i,j,k+1} & = P_{i,fwd} \times dd_{i-1,j-1,k} + P_{i,que} \times dd_{i,j-1,k} + P_{i,retro} \times dd_{i+1,j-1,k} \\
\end{align*}
\] (5.26)
\[ dd'_{d,j,k-1} = P_{d, fwd} \cdot dd_{d-1,j-1,k} + P_{d, que} \cdot dd_{d,j-1,k} \] (5.27)

Therefore, the probability \( dd_{i,j,k-1} \) will be given by

\[ dd_{i,j,k-1} = \frac{dd'_{i,j,k-1}}{\sum_{j=0}^{d} dd'_{i,j,k-1}} \] (5.28)

The iterative computations of \( dd_{i,j,k-1} \) in consecutive slots can be represented by a tree as shown in Figure 5.9, the maximum depth of which is limited to the upper bound on \( j \). The tree is asymmetrical where some of the branches terminate early, meaning that the cell has reached a hopcount value equal to 0 and is absorbed. Some of the branches of the tree may never terminate in which case the upper bound on the depth of the tree will limit the growth of the tree.

![Figure 5.9: Computing the Delay Distributions of a Cell with a Probability Tree Analysis](image)

Once we have the delay distributions \( dd_{ij}, i \leq j \leq 4 \cdot d \), one can compute the out-of-orderness values, as mentioned earlier.

### 5.3 Summary

One of the drawbacks of binary hypercube switches, using queuing buffers and retrograding is out-of-order arrival of cells. We present some of the related models developed by previous researchers following which we present our model which is based on one cell model. The delay distributions of a tagged cell is derived which can be used to compute the exact figures for out-of-orderness parameters. The out-of-order...
model results are presented in Chapter 7 where the accuracy is also validated against simulations.
Chapter 6

Validation of the Single Node Model

In order to study the performance of a buffered deflecting hypercube switch, an approximate Markov model of the switch was built in Chapter 4. A probabilistic model of a single switching element was developed to capture the behavior of a d-dimension hypercube switch. This approach avoids the generation of an exponential state space and scales well with increases in number of switching elements, i.e., with increases in dimension of the switch. Based on the Markovian characteristic of the model, an iterative approach was developed to solve the model numerically. The procedure starts with a known initial state of the element and computes time-slot by time-slot the distributions of the output states of the switching element. The computation stops when the numerical solution converges, at which point a steady-state distribution of the output states of the element is realized. This steady-state distribution is used to compute the various performance metrics.

The performance measures of interest are the average number of forward-in cells arriving per CEC at the forward incoming links of a switching element, the average number of cells stored in the queuing buffers, and the average number of cells retrograded, with a variation in offered load to the switching element. The four important design parameters, apart from the routing algorithm, that impact performance of the
switch are: (1) the dimension of the switch, (2) the size of the queue buffer, (3) the number of delivery channels, and (4) the offered load. Before the model is run, these parameters need to be specified.

This chapter gives numerical results from the model, validates the model by comparing to the results of detailed simulations, and investigates several interesting issues which arise. Section 6.1 presents numerical results obtained from the model equations. The impact of some of the design parameters on switch performance is discussed. The accuracy of the model is validated against simulation results in Section 6.2. The numerical model is capable of providing some interesting insights into the switch behaviour, which Section 6.3 explores. It was pointed out that provision of queuing buffers and deflection routing promised better switch performance in terms of cell loss figures (zero cell loss for transit cells) and cell latency than either approach used alone. Numerical results are used to substantiate the claim in Section 6.4. Section 6.5 points out limitations of the model to capture switch behavior when the restrictions on the number of delivery channels of the switch are removed. An important design issue regarding the number of delivery channels each switching element should have to provide acceptable performance is addressed. Finally, we conclude the chapter with a summary of the results in Section 6.6.

6.1 Experimentation with the Model

The computer time required to solve the numerical model increases with the dimension of the hypercube. In order to keep a reasonable run time with all the experiments, a dimension-3 hypercube is taken as the basis of the performance study, unless otherwise specified. The size of the queue buffer is varied from 1 to \( d \), the dimension of the hypercube. The number of delivery channels is varied from 1 to \( d + 1 \). Since a switching element can receive no more than \( d + 1 \) cells per CEC (\( d \) from its neighbors, and 1 from its own injection), \( d + 1 \) delivery channels are the most that can be used. The number of delivery channels is equal to \( d + 1 \) in the following numerical examples, unless otherwise specified.
The model assumes Bernoulli injection of traffic at each switching element. The probabilistic injection load is varied from 0.5 to 0.9, in steps of 0.1. The destinations of the new cells are uniformly distributed over the set of elements in the switch, including the source of the cell. The hopcounts of the new cells follow a binomial distribution. For example, in a dimension-3 hypercube, there are \(2^3 = 8\) switching elements. The destinations of the new cells generated at any switching element can be any of these 8 switching elements (including itself, i.e., self-delivery) with a probability equal to \(\frac{1}{8}\). But, when considered in terms of hopcounts, the probability values with which a cell with a particular hopcount is generated follows a binomial distribution. This happens because of the hypercube topology in which 8 switching elements are connected: one element each at distances 0 and 3, 3 elements at distances 1 and 2. The probability values with which a distance-0 cell or distance-3 cell is generated is given by \(\frac{1}{8}\) and the probability with which a distance-1 or distance-2 cell is generated is given by \(\frac{3}{8}\).

The only source of loss in buffered deflection hypercubes takes place when a new cell is not accepted into the switching network for delivery, after being considered for routing. This happens when all resources, the forward-out buffers and the queue-out buffers of a switching element are occupied. In that case, the cell is dropped and cleared from the injection buffer, resulting in injection loss.

There is a complex interplay of the influence of the various design parameters on the performance of the switch. Experimentation using full-factorial design would require many experiments. We avoid the expense of a full-factorial design and carry out selective factor experiments. It will be shown that our selective factor experiments are effective.

The model is solved numerically following an iterative approach in which the computation starts in the zeroth iteration with a known initial state of the switching element. A typical input state of a switching element at the beginning of a routing cycle \(t_1\) consists of the tuple \(<F_{I_{mt}}, Q_{I_{nt}}, I_{N_{J_{pt}}}>\), where \(F_{I_{mt}}\), \(Q_{I_{nt}}\), and \(I_{N_{J_{pt}}}\) represent the \(m\)-th forward-in vector, \(n\)-th queue-in vector and \(p\)-th injection vectors. An output state of a switching element at the end of a routing cycle \(t_2\) is given by a 5-tuple representation \(<Delt, FS_{st}, QO_{ot}, RO_{kt}, Lt>\). \(Delt\) represents
the number of cells delivered and is a random variable ranging between 0 and \((d + 1)\) (includes the possible distance zero cell injected by the local source for self-delivery). \(FS_{it2}\), \(QO_{jt2}\), and \(RO_{kt2}\) are \(i\)-th forward-success vector, \(j\)-th queue-out vector and \(k\)-th retrograde vectors respectively. The random variable \(L_{t2}\), \((0 \leq L_{t2} \leq 1)\) indicates the number of cells lost at the time of injection, also known as injection loss. The actual \(m\)-th forward-out vector \(FO_{mt2}\) of the state can be obtained by adding the two subcomponents \(FS_{it2}\) and \(RO_{kt2}\). In the zeroth iteration, the switching element has no cells in its forward-in buffers, and queue-in buffers, but may have cells in its injection buffer. Therefore, \(Pr[FI_{01} = 0]\) and \(Pr[QI_{01} = 0]\) are 1.0, where the index 0 represents the empty vector of the vector spaces \(FI_{vect}\) and \(QI_{vect}\) respectively. But there are non-zero probabilities associated with other vector components (in addition to index 0) of the injection vector space \(INJ_{vect}\). Before the zeroth switching cycle begins, \(Pr[FS_{02} = 0] = Pr[QO_{02} = 0] = Pr[RO_{02} = 0] = Pr[FO_{02} = 0] = 1.0\) and \(Del_{02} = L_{02} = 0\). During the switching cycle, the element makes transitions from an input state to one or more output states. The probability distribution of the element’s output states at the end of the first switching cycle is used to update distributions of \(FI_{vect}\) and \(QI_{vect}\) at the beginning of the next cycle. As a result, at the beginning of the next iteration (i.e., cycle 1) the input vector spaces \(FI_{vect}\) and \(QI_{vect}\) may have some entries with non-zero probability values. The iterative sequence continues until the convergence criterion defined in Chapter 4 is satisfied at which point the computation stops.

It was pointed out in Chapter 4 that a simpler heuristic can be used to determine the numerical convergence of the switch system towards its equilibrium. An intuitive reason behind choosing the heuristic is that every cycle a switching element system has only one entry point for a cell to enter - the injection and two exit points for a cell to depart from the system - the delivery and the loss. At steady-state, the rate at which cells are offered and accepted at the source should equal the rate at which cells are absorbed from the network at the destination within an acceptable limit which we call the convergence factor. As mentioned in the previous chapter, 0.0005 is chosen as the convergence factor.
Figure 6.1: Transient Behavior of a Dimension 3 Hypercube Switch with 4 Delivery Channels at an Injection Load of 0.9

The convergence of a dimension-3 switch system with 4 delivery channels and 3 queue buffers at each of its switching elements is shown in Figure 6.1 for an injection load of 0.9. It takes 9 or 10 iterations before the equilibrium is reached.

Figure 6.2 shows the scale of convergence for a dimension-4 switch system with 5 delivery channels and 4 queue buffers at an injection load of 0.9. As seen, a dimension-4 system also takes 9 or 10 iterations for convergence.

Changes in the size of the queue buffer do not have much influence on the number of convergence CECs as shown in Figure 6.3.

The numerical results for the three performance metrics of a dimension-3 hypercube switch with 4 delivery channels and a queue-size of 1 are presented in Table 6.1. and Figures 6.4. and 6.5. Accuracy of all these results are established in Section 6.2 by comparing against simulation data.

Figure 6.4 shows the average number of cells in queue-in buffers of switching elements at various loads. The queue buffer size is 1. Figure 6.5 shows the average number of cells retrograded per CEC for the same queue buffer size.

An increase in queue buffer sizes should result in less retrograding. Figure 6.6 compares retrograde performance of cells in a 4 delivery channel switch for three different queue buffer sizes (Q = 1, Q = 2, and Q = 3).
Figure 6.2: Transient Behavior of a Dimension 4 Hypercube Switch with 5 Delivery Channels at an Injection Load of 0.9

Figure 6.3: Transient Behavior of a Dimension 3 Hypercube Switch with 4 Delivery Channels as a Function of the Queue Buffer Size at an Injection Load of 0.9
Table 6.1: Model Data for Average Number of Cells in Forward-In Buffers of a Dimension 3 Hypercube Switch with Queueing Buffers (Queue Size = 1), and Retrograding

<table>
<thead>
<tr>
<th>Load</th>
<th>Average No. of Forward-In Cells per CEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.751757</td>
</tr>
<tr>
<td>0.6</td>
<td>0.903338</td>
</tr>
<tr>
<td>0.7</td>
<td>1.656035</td>
</tr>
<tr>
<td>0.8</td>
<td>1.210089</td>
</tr>
<tr>
<td>0.9</td>
<td>1.365952</td>
</tr>
</tbody>
</table>

Figure 6.4: Model Data for Average Number of Cells in Queue-In Buffers of a Dimension 3 Hypercube Switch with Queueing Buffers (Queue Size = 1), and Retrograding

Variation in queue buffer sizes also impacts injection loss figures of a hypercube switch. Figure 6.7 shows the cell loss probabilities of a dimension 3 switch with changes in injection load for three different queue buffer sizes. The number of delivery channels is 4 in all cases. As we can see, increasing the queue buffer sizes reduces the cell loss probabilities.

Experiments with a higher dimension hypercube switch, i.e., dimension-4 switch with 5 delivery channels were also carried out. Four different queue buffer sizes (Queue = 1, Queue = 2, Queue = 3, and Queue = 4) were tried. The model data for the three different performance metrics with queue size equal to 1 are presented in Table 6.2, and Figures 6.8, and 6.9.

Figure 6.8 shows the average number of cells in queue-in buffers of switching
Figure 6.5: Model Data for Average Number of Cells Retrograded in a Dimension 3 Hypercube Switch with Queueing Buffers (Queue Size = 1), and Retrograding

Table 6.2: Model Data for Average Number of Cells in Forward-In Buffers of a Dimension 4 Hypercube Switch with Queueing Buffers (Queue Size = 1), and Retrograding

<table>
<thead>
<tr>
<th>Load</th>
<th>Average No. of Forward-in Cells per CEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>1.002043</td>
</tr>
<tr>
<td>0.6</td>
<td>1.204248</td>
</tr>
<tr>
<td>0.7</td>
<td>1.407895</td>
</tr>
<tr>
<td>0.8</td>
<td>1.613648</td>
</tr>
<tr>
<td>0.9</td>
<td>1.822367</td>
</tr>
</tbody>
</table>

elements at various loads. The queue buffer size is 1. Figure 6.9 shows the average number of cells retrograded per CEC for the same queue buffer size.

When the retrograde performance of a dimension 3 hypercube switch (Figure 6.5) is compared to that of the dimension-4 switch (Figure 6.9), it is noted that for the same queue buffer size, the dimension-4 hypercube has more cells retrograded than dimension-3 switches. This is expected, because there is more traffic in dimension 4 than in dimension 3 and as a result, more contention for the single queue buffer.

As in the case of dimension 3, an increase in queue buffer sizes results in less retrograding. Figure 6.10 shows retrograde performance of a dimension 4 switch for four different queue buffer sizes (Queue = 1, Queue = 2, Queue = 3, and Queue = 4).
Figure 6.6: Retrograde Performance of Cells in a Dimension 3 Hypercube Switch with 4 Delivery Channels with Variation in Injection Load for Three Different Queue Buffer Sizes (Queue Size = 1, Queue Size = 2, and Queue Size = 3)

The cell loss figures of a dimension 4 switch are also affected by changes in queue buffer sizes. Figure 6.11 shows the cell loss probabilities of a dimension 4 switch with changes in injection load for four different queue buffer sizes. The number of delivery channels is 5 in all the cases.

Link utilization \( (m) \) is defined as the average fraction of the time that a link of a switching element is busy transmitting cells. Link utilization can be used to compute the average number of cells that arrive at the forward incoming links of a switching element. It is given by the sum of the probability values \( m(i) \), where \( i \) is the hopcount of the cell varying from 0 to \( d \). If \( m \) is the link utilization value, then the average number of cells that arrive at the forward incoming links of a switching element is given by \( d \times m \). The link utilization vector \( \mathbf{m} = (m(0), m(1), m(2), m(3)) \) for a dimension 3 hypercube switch with 4 delivery channels and a queue buffer of 1 is shown in Table 6.3.

It is evident from Table 6.3 that incoming cells with distance zero constitute the most significant component of the link traffic. For example, at a load value of 0.9, distance zero cells constitute nearly \( (0.262226/0.455318) \times 100 = 57.5\% \) of the total link traffic. This result is expected due to the fact \( 2^d - 1 \) of the cells eventually take a
Figure 6.7: Cell Loss Probability of a Dimension 3 Hypercube Switch with 4 Delivery Channels with Variation in Injection Load for Three Different Queue Buffer Sizes

Table 6.3: Link Utilization Vectors at Various Loads for 4 Delivery Channels

<table>
<thead>
<tr>
<th>Load</th>
<th>m(0)</th>
<th>m(1)</th>
<th>m(2)</th>
<th>m(3)</th>
<th>$\sum_{i=0}^{d} m(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.145815</td>
<td>0.083604</td>
<td>0.021137</td>
<td>0.000030</td>
<td>0.250586</td>
</tr>
<tr>
<td>0.6</td>
<td>0.174942</td>
<td>0.100510</td>
<td>0.025592</td>
<td>0.000069</td>
<td>0.301113</td>
</tr>
<tr>
<td>0.7</td>
<td>0.204085</td>
<td>0.117560</td>
<td>0.030226</td>
<td>0.000141</td>
<td>0.352012</td>
</tr>
<tr>
<td>0.8</td>
<td>0.233189</td>
<td>0.134795</td>
<td>0.035113</td>
<td>0.000266</td>
<td>0.403363</td>
</tr>
<tr>
<td>0.9</td>
<td>0.262226</td>
<td>0.152272</td>
<td>0.040350</td>
<td>0.000470</td>
<td>0.455318</td>
</tr>
</tbody>
</table>
Figure 6.8: Model Data for Average Number of Cells in Queue-In Buffers of a Dimension 4 Hypercube Switch with Queuing Buffers (Queue Size = 1), and Retrograding final hop which reduces their distances to zero.

In summary, the numerical examples discussed in this section show that the probabilistic model converges towards equilibrium rapidly in the case of \( d + 1 \) delivery channels. The numerical model results for the three performance metrics are presented for a dimension 3 hypercube with 4 delivery channels. It is shown that changes in queue buffer sizes impact the retrograde performance as well as the cell loss ratio figures of a dimension 3 switch. Similar observations are made for a higher dimension switch, i.e., a dimension 4 switch. Finally, the link utilization vector for a dimension 3 hypercube switch with 4 delivery channels and a queue buffer of 1 is presented in which it is pointed out that a significant component of link traffic consists of distance zero cells.

The numerical model results provide significant insight into the behaviour of the switch that can be used to develop guidelines for the design and engineering of hypercube switches. This will be discussed in Section 6.3 following validation of the numerical results, which is done in the next section.
6.2 Simulation Results

In order to support the analytical model results reported in the last section, a switch simulator for a $d$-dimension hypercube was written in the C programming language. Unlike the model where only a single switching element is modeled to keep the analysis tractable, the simulator simulates an entire hypercube switch consisting of $2^d$ switching elements. The simulator is parameterized in the sense that by inputting the key parameters (the dimension of the hypercube, the number of delivery channels, the queue buffer size, and the injection load) the simulator can model a wide variety of switching cores. Bernoulli injection of cells in each switching element with uniform random destinations is simulated using random number generators. Given an injection load, the first random number generator decides whether a cell should be generated at a switching element’s local source during any CEC. The second one decides the destination of the new cell generated. These two generators generate independent sequence of random numbers using separate seed values to avoid correlation among the streams. In addition, each switching element has its own pair of random number generators. The program allows tracking of each and every cell from the CEC it is generated until the CEC it is delivered. The entire history of a cell
in the form of how many CECs the cell has been forwarded, how many CECs it has spent in the queue buffer, and how many CECs it has been deflected, can be easily retrieved. Cell loss occurs in buffered deflection switches only when cells generated in injection buffers are not accepted by the switching elements. This contributes to injection loss of cells in a switch. Once accepted, cells can never be lost. The number of dropped cells is recorded to measure the cell loss probability.

The simulator first initializes the simulated objects (the traffic sources, the switching elements, and the links) and then starts to simulate their functions. During initialization, all the static attributes of the simulated objects are initialized according to the connectivity of the switch model. For example, the switch topology static attributes—such as adjacent neighbors, i.e., previous nodes, next nodes, injection buffers, queue buffers, delivery buffers are all initialized. The dynamic attributes of the objects like the performance measure attributes, the network resources attributes (e.g., available buffer capacity, available link bandwidth), and states of traffic sources are all updated during simulation. After initialization, the simulator repeatedly checks the status of the simulated objects and carries out the required actions.

The simulators are event-driven; a simulation timer is used to schedule the events.
The simulation runs until the simulation timer reaches a pre-defined run length. The run lengths of our simulations are chosen using the techniques described by Jain in [34]. The goal is that a simulation should be run until the confidence level for the measured performance metric is obtained within the desired margin. The technique of computing run length based on desired confidence interval is discussed in Appendix A. In our case, simulations are run for 210000 cycles using the batch means method, with 21 batches of 10000 cycles each. The results of the first 10000 cycles are discarded. Each data value lies within a 95% confidence interval of width less than 2.5% of the value.

The following subsection presents the simulation results which are compared with numerical results to validate the accuracy of the model.

### 6.2.1 Model Validation using Simulation Results

As with the model, the four key parameters that need to be specified before the simulator of a buffered deflection switch is run are: (1) the dimension of the switch, (2) the queue buffer size, (3) the number of delivery channels, and (4) the injection load. The dimension of the switch is 3 in the following examples, unless otherwise
specified. The number of delivery channels is \( d+1 \), i.e., 4, unless otherwise mentioned. The injection load is varied from 0.5 to a maximum of 1.0, in steps of 0.1.

The validation begins by checking the assumption made in developing the model that the hopcount distributions of the cells injected at the switching element of the model are binomial. Table 6.4 compares the hopcount distribution of cells generated in the simulation model with the theoretically computed value. For a given hopcount, the ratio of the two probability figures indicate the accuracy of the model assumptions. Ideally, the ratio should be 1.0. As seen in Table 6.4, there is a close agreement between the hopcount distributions of the cells generated in the simulator and that in the model.

Table 6.4: Comparison of the Hopcount Distribution of the Cells Generated in the Simulator with the Theoretically Computed Value at a Load of 0.9

<table>
<thead>
<tr>
<th>Hopcount</th>
<th>Model</th>
<th>Simulation</th>
<th>Simulation/Model Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1125</td>
<td>0.113815</td>
<td>1.0117</td>
</tr>
<tr>
<td>1</td>
<td>0.3375</td>
<td>0.336707</td>
<td>0.9976</td>
</tr>
<tr>
<td>2</td>
<td>0.3375</td>
<td>0.331203</td>
<td>0.9813</td>
</tr>
<tr>
<td>3</td>
<td>0.1125</td>
<td>0.111904</td>
<td>0.9947</td>
</tr>
</tbody>
</table>

The performance metrics chosen to establish the accuracy of the model are the average number of forward-in cells, the average number of queue-in cells, and the average number of cells retrograded per CEC at injection loads varying between 0.5 and a maximum of 1.0. Table 6.5 shows the average number of cells in forward-in buffers of a dimension 3 switch with 4 delivery channels and single queue buffer. The model and simulation data agree closely.

Figure 6.12 shows the average number of cells in queue-in buffers of switching elements at various loads. The queue buffer size is 1. Again, model and simulation data agree. The average number of cells retrograded per CEC is shown in Figure 6.13.

There is a slight deviation between the model and the simulation in case of retrograde performance of cells. The amount of retrograding of cells in buffered deflection switches is very low as seen in the figure. The model predicts a more pessimistic
Table 6.5: Average Number of Cells in Forward-In Buffers of a Dimension 3 Hypercube Switch with Queueing Buffers (Queue Size = 1), and Retrograding

<table>
<thead>
<tr>
<th>Load</th>
<th>Model Data</th>
<th>Simulation Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.751737</td>
<td>0.751789</td>
</tr>
<tr>
<td>0.6</td>
<td>0.903338</td>
<td>0.902579</td>
</tr>
<tr>
<td>0.7</td>
<td>1.056035</td>
<td>1.055124</td>
</tr>
<tr>
<td>0.8</td>
<td>1.210089</td>
<td>1.207555</td>
</tr>
<tr>
<td>0.9</td>
<td>1.365952</td>
<td>1.362816</td>
</tr>
</tbody>
</table>

Figure 6.12: Average Number of Cells in Queue-In Buffers of a Dimension 3 Hypercube Switch with Queueing Buffers (Queue Size = 1), and Retrograding

picture than the simulation data. Ideally, we would want minimum retrograding in a buffered deflection switch from the point of view of switch performance.

For higher dimension switches, the model also proves accurate. Table 6.6 and Figures 6.14 and 6.15 compare the model data and simulation results of a buffered deflection hypercube with dimension 4. The queue buffer size is 1 and the number of delivery channels is $d + 1$, i.e., 5.

Accuracy of the numerical model can be further established by validating model results in switches with slightly different architectures. Both the numerical model and the simulation model are flexible enough to accommodate different architecture scenarios. For example, deflection switches with no queuing buffers and buffered switches with no retrograding features are two important variants. It is shown that
Figure 6.13: Average Number of Cells Retrograded in a Dimension 3 Hypercube Switch with Queueing Buffers (Queue Size = 1) and Retrograding

Table 6.6: Average Number of Forward-In Cells in a Dimension 4 Hypercube Switch with Queueing Buffers (Queue Size = 1) and Retrograding

<table>
<thead>
<tr>
<th>Load</th>
<th>Model Data</th>
<th>Simulation Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>1.002043</td>
<td>1.002355</td>
</tr>
<tr>
<td>0.6</td>
<td>1.204248</td>
<td>1.203520</td>
</tr>
<tr>
<td>0.7</td>
<td>1.407895</td>
<td>1.406745</td>
</tr>
<tr>
<td>0.8</td>
<td>1.613648</td>
<td>1.612468</td>
</tr>
<tr>
<td>0.9</td>
<td>1.822367</td>
<td>1.819539</td>
</tr>
</tbody>
</table>

our numerical model is capable of capturing performance of these switches also. The two performance metrics of deflection switches - average number of forward-in cells and average number of retrograded cells - match closely. Table 6.7 and Figure 6.16 compare the simulation and the model data for the average number of cells arriving at the forward incoming links of a switching element of a deflection switch and the average number of cells retrograded.

It is observed in Figure 6.16 that the model works well in scenarios where significant retrograding takes place.

The model is also validated even in the case of buffered switches with no retrograde features. The two performance metrics of buffered switches - average number of forward-in cells and average number of queue-in cells show good agreement between
Figure 6.14: Average Number of Cells in Queue-In Buffers of a Dimension 4 Hypercube Switch with Queueing Buffers (Queue Size = 1) and Retrograding

Table 6.7: Average Number of Forward-In Cells for a Dimension 3 Hypercube Switch with Retrograding Features Only

<table>
<thead>
<tr>
<th>Load</th>
<th>Model Data</th>
<th>Simulation Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.818911</td>
<td>0.813451</td>
</tr>
<tr>
<td>0.6</td>
<td>1.004669</td>
<td>0.994793</td>
</tr>
<tr>
<td>0.7</td>
<td>1.199080</td>
<td>1.186201</td>
</tr>
<tr>
<td>0.8</td>
<td>1.402034</td>
<td>1.385909</td>
</tr>
<tr>
<td>0.9</td>
<td>1.611807</td>
<td>1.588865</td>
</tr>
</tbody>
</table>

the model and the simulation, as seen in Table 6.8 and Figure 6.17. The queue buffer size is 1 and the number of delivery channels is $d + 1$.

In summary, all the examples presented so far validate the accuracy of the model. The various performance metrics of the switch agree closely for the model and the simulation. For higher dimension switches the model shows good agreement with the simulation data. Performance of the switches with slightly different architectures establish further accuracy of the numerical model. It can capture situations in which significant retrograding takes place. The results agree in cases where some of the key input parameters are changed.

In the next section, some of the interesting insights that can be derived from the model data are presented. These results can help design and engineer a switch more
Figure 6.15: Average Number of Cells Retrograded in a Dimension 4 Hypercube Switch with Queuing Buffers (Queue Size = 1), and Retrograding

Table 6.8: Average Number of Cells in Forward-In Buffers of a Dimension 3 Hypercube Switch with Queuing Buffers (Queue Size = 1) Only and No Retrograding

<table>
<thead>
<tr>
<th>Load</th>
<th>Model Data</th>
<th>Simulation Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.746856</td>
<td>0.750671</td>
</tr>
<tr>
<td>0.6</td>
<td>0.893910</td>
<td>0.898078</td>
</tr>
<tr>
<td>0.7</td>
<td>1.039124</td>
<td>1.046404</td>
</tr>
<tr>
<td>0.8</td>
<td>1.181824</td>
<td>1.195137</td>
</tr>
<tr>
<td>0.9</td>
<td>1.321289</td>
<td>1.342027</td>
</tr>
</tbody>
</table>

effectively.

6.3 Steady-State Distributions of Vector Spaces

The steady-state probability distributions of the various queue-in vectors and/or forward-in vectors obtained as a solution of the numerical model equations provide significant insight into the behaviour of the switch. Table 6.9 shows probability values of the five most significant components of queue-in vector space of a switching element under equilibrium. The values are shown for a load value of 0.9 for a dimension 3 switch with 4 delivery channels and a queue buffer size equal to 3. The table has 35 possible entries which reflect the entire queue-in vector space. Most of the entries are
Figure 6.16: Average Number of Retrograded Cells per CEC for a Dimension 3 Deflection Switch

Table 6.9: Steady-State Probability Distributions of the Queue-In Vectors of a Dimension 3 Hypercube Switch with 4 Delivery Channels and a Queue Buffer (Queue Size = 3)

<table>
<thead>
<tr>
<th>Total</th>
<th>dist=0</th>
<th>dist = 1</th>
<th>dist = 2</th>
<th>dist = 3</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.876437</td>
</tr>
<tr>
<td>&lt;1:</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.098874</td>
</tr>
<tr>
<td>&lt;2:</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0.007284</td>
</tr>
<tr>
<td>&lt;2:</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.001963</td>
</tr>
</tbody>
</table>

either zero or close to zero. A fairly good idea about steady-state queue occupancy of the switch can be obtained from the table. One can easily make out from the probability values of the three queue-in vectors \(<0:00000>\), \(<1:01000>\), and \(<1:00100>\) that there is a 87% chance of finding the queue buffer empty, 10% chance of finding only one distance 1 cell and a mere 1% chance of finding one distance 2 cell. The probability of finding more than one cell in the queue-in buffer is very low: 0.1% chance of finding one distance 1 and one distance 2 cell and a 0.7% chance of finding two distance 1 cells.

Instead of representing each vector with its unique tuple representation, we use an arbitrary one-to-one mapping between a vector and an unique index number, as shown
in Appendix B. Once the unique index number representations for the vectors are assigned, the probability mass functions (pmf) of all the vector components (except for index zero which represents the vector $<0:0000>$, i.e., the empty buffer) can be represented in the form of a graph whose x-axis is labeled with the index numbers and the y-axis with the probability values. Figure 6.18 shows the pmf of all the queue-in vector components (except for index zero which represents the empty buffer) for a dimension 3 switch with 4 delivery channels and 3 queue-in buffers.

The pmf of forward-in vector components of a switching element can also be represented in a similar way. These vector components provide information about the number and hopcounts of the cells that arrive at the $d$ forward incoming links. These vectors contain the cells that have been forwarded as well as the cells that have been deflected in the previous CEC. The probability mass functions of the forward-in vector components of a dimension 3 switch with 3 queue buffers are shown in Figure 6.19. As seen in Figure 6.19 the significant forward-in vectors are: $<1:0100>$ (index 3), $<1:100:0>$ (index 4), $<2:1100>$ (index 13), and $<2:2000>$ (index 14).

The probability distributions of the various state vectors can also be used to answer various specific design questions like the probabilities of finding only one
Figure 6.18: Probability Mass Functions of the Queue-In Vector Space of a Dimension 3 Hypercube Switch with 4 Delivery Channels and Queueing Buffers (Queue Size = 3)

Figure 6.19: Probability Mass Functions of the Forward-In Vector Space of a Dimension 3 Hypercube Switch with 4 Delivery Channels and 3 Queue Buffers
distance 1 cell, only one distance 2 cell, or only one distance 3 cell in a single queue buffer. Similarly, in the case of forward-in buffers the probabilities of finding exactly 3 distance 0 cells arriving or the probabilities of finding 3 distance 1 cells arriving can be found out from the pmf of forward-in vector space components. These are some of the interesting results that can be obtained from the numerical model.

6.4 Why Buffered Deflecting Switches?

In order to minimize loss of transit cells, i.e., cells that have already been accepted into the switch for delivery, hypercube switches with queuing buffers at the switching elements have been considered. These switches are known as buffered switches. Buffering can reduce loss of transit cells, but can not completely eliminate loss, unless buffer sizes become infinitely large. Also, the provision of queuing buffers at switching elements achieves lower cell loss at the expense of increased cell latency. The increased latency is due to the queuing of cells. The other source of loss of cells, i.e., injection loss, may also be present in these switches.

An interesting alternative to achieve zero cell loss (of transit cells) without having to use queuing buffers at the nodes, is to deflect the cells losing contention to non-optimal links. Deflection routing helps avoid queuing buffers and dropping cells in transit. The only form of cell loss in hypercube switches employing deflection routing is injection loss. These switches achieve superior performance (zero loss of transit cells), reliability, and robustness but at the cost of increased cell latency. Whenever a cell is deflected to a non-optimal link, it travels a number of extra hops, called the deflection index which is equal to 2 in case of hypercubes. This causes an extra latency of the deflected cells, called the retrograde latency. The total latency of cells in switches without queue buffers and employing deflection routing is larger than that in switches with queue buffers and no deflection routing.

A small amount of buffering inside the switching elements of a hypercube switch that employs deflection reduces the probability of deflection which in turn reduces the latency figures of the cells. In the event of congestion, only when the buffer becomes
full and cells that lose are deflected. This mixed approach avoids loss of cells, avoids large buffer size, and keeps latency figures of the cells lower than either pure queuing or pure retrograding solution.

A dimension 3 hypercube with 4 delivery channels is considered for the experimentation. Both numerical results and simulation data are used to substantiate the claim of better switch performance for a buffered deflection hypercube. Three different variants of switch architectures are considered: (1) switches with queuing buffers only, (2) switches with deflection routing only, and (3) switches with both queuing and retrograding features. As a base case, we use a simple vanilla version of the hypercube switch with no queuing buffers and no retrograding features.

The first set of experiments consider cell loss performance of the four variants of hypercube switches with variation in injection load. Figure 6.20 shows cell loss probabilities of a dimension-3 hypercube switch with changes in injection load for the four different architecture scenarios. Numerical results are presented for comparison purposes. The queue buffer size is 1 for switches with queueing buffers only and for switches with both queueing and retrograding. As seen in the figure, the highest cell loss occurs for vanilla switches with no queueing buffers and with no retrograding. The cell loss figures in these switches include both injection loss and loss of transit cells. Addition of queueing buffers improves cell loss performance as seen in the case of switches with queueing buffers only. Cell loss in these switches occurs due to loss of transit cells as well as injection loss. Provision of retrograding or deflection at the switching elements with no queueing buffers also improves loss figures. Loss in this case is purely injection loss due to cells in injection buffers. As seen in the figure, deflection switches perform better than buffered switches with queue size equal to 1. The best loss performance is achieved for switches with queueing buffers and deflection routing.

Buffered switches without retrograding show better cell loss performance than deflection switches. In order to get similar performance as that of deflection switches, buffered switches need more buffers to be added. Figure 6.21 compares cell loss performance for buffered switches with two different queue sizes ($Q = 1$ and $Q = 3$)
Figure 6.20: Cell Loss Probability of a Dimension 3 Hypercube Switch with \( d - 1 \) Delivery Buffers under Various Architecture Scenarios

with that of deflection switches. Adding three queue buffers improves CLR figures of buffered switches compared to deflection switches. Also, buffered deflection switches with single queue buffer perform better than buffered switches with three queue buffers.

An increase in queue buffer sizes in case of buffered deflection switches should improve cell loss figures. Figure 6.22 shows the improvement.

The latency, i.e., the number of CECs required for a cell to be delivered to its destination after being accepted at a switching element, depends on the hopcount of the cell when it is generated and also on the routing algorithm running in the switch. In the vanilla switch with 4 delivery channels and with no queueing buffers and retrograde features, a cell with a hopcount of 1 will take 1 CEC before it is delivered. A cell with 2 hopcounts will consume 2 CECs and so on. The average latency count of a cell in a vanilla switch is given by \( \frac{d}{2} \), based on the assumption that cells are generated with binomial hopcount distributions. In the case of switches with queueing buffers and no retrograde features, the average latency of cells goes up because of queue latency. Similarly, in the case of deflection switches that have no queueing buffers average latency increases because of retrograde latency. In buffered deflection switches, cells suffer both queue latency and retrograde latency. It was
Figure 6.21: Comparison of Cell Loss Probabilities of Buffered Switches with Two Different Queue Buffer Sizes (Queue Size = 1 and Queue Size = 3) with Deflection and Buffered Deflection Switches

Figure 6.22: Comparison of Cell Loss Probabilities of a Dimension 3 Hypercube Switch With 4 Delivery Channels and Two Different Queue Buffer Sizes (Queue Size = 1 and Queue Size = 3)
Figure 6.23: Comparison of Average Latency for a Dimension 3 Hypercube Switch with 4 Delivery Channels for Three Different Variants of Switches at Various Injection Loads

discussed earlier that cells in deflection switches have higher latency than those in buffered switches. However, cells in buffered deflection switches have higher latencies than cells in buffered switches, but definitely lower latencies than unbuffered deflection switches. Figure 6.23 shows the average latency of cells at various injection loads for the three variants of hypercube switches. In the cases of buffered switches and buffered deflection switches the queue size is 1. We present simulation data to compare the latency figures. As seen from the figure, the latency of cells in buffered deflection switches fall between those of buffered switches and deflection switches.

In summary, both the numerical model data and the simulation results show that hypercube switches with queuing buffers and retrograde features have superior performance than either buffered switches or deflection switches. The mixed approach achieves a balance between cell loss ratio and cell latency. However, superior performance of buffered deflection hypercube switches is not a major concern for us. In this thesis, we develop a model for this best-performing switch and then validate the accuracy of the model against detailed simulation results.
6.5 Delivery Channels

One of four important design parameters that influence the performance of the switch is the number of delivery channels. In all the examples presented so far, the number of delivery channels was $d+1$, where $d$ is the dimension of the hypercube. This was in line with the assumptions made by previous researchers. The impact of the number of delivery channels on various performance metrics of a hypercube switch employing hybrid routing approach was studied by removing the $d+1$ delivery buffer assumption. For a dimension 3 cube, two different variations in the number of delivery channels were attempted: single delivery channel and two delivery channels.

The first observation made by restricting the number of delivery channels from $d+1$ to 1 was the slow convergence of the numerical model towards its equilibrium. Figure 6.24 shows that the switch system with single delivery channel took nearly 57 iterations before it attained equilibrium. The reason for slow convergence in the case of a single delivery channel is that the delivery probability is reduced in each iteration as a result of the restricted number of delivery channels. This results in numerous distance zero cells arriving on the forward incoming links and contending for switch resources, i.e., the delivery channel and the queue buffers. The probability distributions of the non-empty vector components of the forward-out and queue-out vector space change slowly from iteration to iteration. Since the distributions of the input vector space components in our model are synthesized from those of the output vector space, the slow change in probability values is also reflected on the input side.

Next, we study the three performance metrics - the average number of forward-in cells per CEC, the average number of cells in queue-in buffers, and the average number of cells retrograded per CEC, for a hypercube switch with only one delivery channel. Table 6.10, and Figures 6.25, and 6.26 compare the model and the simulation data for a dimension 3 switch with only one delivery channel. A significant deviation between the simulation and the model data is observed for load values above 0.6.

Table 6.11 shows the link utilization vector for a switch with single delivery channel. When compared to link utilization vector of a switch with 4 delivery channels
Figure 6.24: Transient Behavior of a Dimension 3 Hypercube Switch with Queue Buffers (Queue Size = 3) as a Function of Delivery Channels at an Injection Load of 0.9

Table 6.10: Average Number of Forward-In Cells for a Dimension 3 Hypercube Switch under Single Delivery Assumptions. Queue Size = 1

<table>
<thead>
<tr>
<th>Load</th>
<th>Model Data</th>
<th>Simulation Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.793097</td>
<td>0.800063</td>
</tr>
<tr>
<td>0.6</td>
<td>0.993266</td>
<td>1.106831</td>
</tr>
<tr>
<td>0.7</td>
<td>1.241494</td>
<td>1.454768</td>
</tr>
<tr>
<td>0.8</td>
<td>1.594574</td>
<td>2.109865</td>
</tr>
<tr>
<td>0.9</td>
<td>2.219451</td>
<td>2.734625</td>
</tr>
</tbody>
</table>

(Table 6.3) Table 6.11 shows that link traffic goes up considerably in the case of a single delivery channel. This is due to an increase in retrograding of cells, when the number of delivery channels is reduced. In the case of a single delivery channel, distance zero deliverable cells may be retrograded. For identical queue buffer size (Queue = 1), if the number of delivery channels is reduced from 4 to 1, the amount of retrogradation increases. The difference becomes more prominent at higher injection load, i.e., at loads greater than 0.7. Table 6.12 shows the results.

The probability mass functions (pmf) of all the queue-in vector components (excepting index zero which represents the empty buffer) in the case of a dimension 3 switch with 3 queue-in buffers are given for comparison between 1 delivery channel
Table 6.11: Link Utilization Vectors at Various Loads in a Dimension 3 Hypercube Switch with Single Delivery Channel

<table>
<thead>
<tr>
<th>Load</th>
<th>m(0)</th>
<th>m(1)</th>
<th>m(2)</th>
<th>m(3)</th>
<th>(\sum_{i=0}^{d} m(i))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.151763</td>
<td>0.090442</td>
<td>0.022071</td>
<td>0.000089</td>
<td>0.264365</td>
</tr>
<tr>
<td>0.6</td>
<td>0.187312</td>
<td>0.115278</td>
<td>0.028209</td>
<td>0.000089</td>
<td>0.331088</td>
</tr>
<tr>
<td>0.7</td>
<td>0.228067</td>
<td>0.147738</td>
<td>0.037133</td>
<td>0.000894</td>
<td>0.413832</td>
</tr>
<tr>
<td>0.8</td>
<td>0.278475</td>
<td>0.196343</td>
<td>0.053789</td>
<td>0.002917</td>
<td>0.531524</td>
</tr>
<tr>
<td>0.9</td>
<td>0.345725</td>
<td>0.284235</td>
<td>0.098139</td>
<td>0.011718</td>
<td>0.739817</td>
</tr>
</tbody>
</table>

Table 6.12: Comparison of Retrograde Performance of a Dimension 3 Hypercube Switch at Various Loads for 4 Delivery Channels and Single Delivery Channel with Same Queue Buffer Size (Queue Size = 1)

<table>
<thead>
<tr>
<th>Load</th>
<th>Delivery Channels = 4</th>
<th>Delivery Channel = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.009915</td>
<td>0.021619</td>
</tr>
<tr>
<td>0.6</td>
<td>0.001784</td>
<td>0.046767</td>
</tr>
<tr>
<td>0.7</td>
<td>0.003198</td>
<td>0.096110</td>
</tr>
<tr>
<td>0.8</td>
<td>0.005386</td>
<td>0.198670</td>
</tr>
<tr>
<td>0.9</td>
<td>0.008651</td>
<td>0.444060</td>
</tr>
</tbody>
</table>

and 4 delivery channels in Figure 6.27. The x-axis is labeled with index parameter values and the y-axis is labeled with probability values. It is interesting to note that all the three major queue-in vector components that have significant probability values are: \(<1:10000>\) (an index value of 4), \(<2:20000>\) (index value of 14) and \(<3:30000>\) (index value of 34), contain only distance zero cells.

Figure 6.28 compares the probability mass functions of the forward-in vector components of a dimension 3 switch with 3 queue buffers for 1 and 4 delivery channels.

If the number of delivery channels is reduced from \(d + 1\) to 2, then the number of deliverable cells in excess of two during any CEC. must be stored in queue buffers or retrograded. Performance under these conditions is shown in Table 6.13, Figure 6.29, and Figure 6.30. The model and the simulation match closely for two delivery channels.

Table 6.14 presents model data for buffered deflection switches with 4 delivery channels and 2 delivery channels. The impact of the number of delivery channels on performance metrics of a buffered deflection switch is shown.
Table 6.13: Average Number of Forward-In Cells for a Dimension 3 Hypercube Switch under Two Delivery Buffer Assumptions. Queue Size = 1

<table>
<thead>
<tr>
<th>Load</th>
<th>Model Data</th>
<th>Simulation Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.732263</td>
<td>0.750063</td>
</tr>
<tr>
<td>0.6</td>
<td>0.904619</td>
<td>0.910683</td>
</tr>
<tr>
<td>0.7</td>
<td>1.058476</td>
<td>1.054768</td>
</tr>
<tr>
<td>0.8</td>
<td>1.214868</td>
<td>1.209865</td>
</tr>
<tr>
<td>0.9</td>
<td>1.374965</td>
<td>1.373908</td>
</tr>
</tbody>
</table>

Table 6.14: Table Comparing the Performance Figures of Model with 4 Delivery Channels and 2 Delivery Channels with Queue Size = 1

<table>
<thead>
<tr>
<th>Load</th>
<th>Foccup.</th>
<th>Qoccup.</th>
<th>Retoccup.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4 Channels</td>
<td>2 Channels</td>
<td>4 Channels</td>
</tr>
<tr>
<td>0.5</td>
<td>0.751757</td>
<td>0.752263</td>
<td>0.031884</td>
</tr>
<tr>
<td>0.6</td>
<td>0.903338</td>
<td>0.904619</td>
<td>0.047803</td>
</tr>
<tr>
<td>0.7</td>
<td>1.056035</td>
<td>1.058476</td>
<td>0.067882</td>
</tr>
<tr>
<td>0.8</td>
<td>1.210089</td>
<td>1.214876</td>
<td>0.092633</td>
</tr>
<tr>
<td>0.9</td>
<td>1.365952</td>
<td>1.374965</td>
<td>0.122783</td>
</tr>
</tbody>
</table>

To find out the reason for failure of the model to capture the switch behavior at high load with one delivery channel, an investigative study was carried out using simulation. A trace of the switch was taken for a large number of cycles. It was found from the trace analysis that at high loads congested spots develop in the switch. Since only one cell is allowed to be delivered per CEC, most of the queuing resources get consumed by temporarily blocked deliverable cells. Also, some of the deliverable cells get retrograded to the router's neighbors because of non-availability of queuing buffers. Even though the particular router avoids cell loss by retrograding the distance zero cells, the neighbors send back those cells to the router the next CEC for delivery. To make the situation worse, because of these distance zero cells, other cells also suffer severe retrograde motion. The to-and-fro movement of the distance zero cells along the links make the links as well as the adjoining neighbors congested spots. Once the congested spots develop the assumption of slot-to-slot independence of cell arrivals on the incoming links of a router becomes invalid. A time correlation or dependence between the arrivals of cells on consecutive CECs develops. The model no longer
captures the switch behavior accurately.

Significant extensions were made to the model to capture this correlation behavior. They were not successful. A two node model was attempted in which two switching elements. switching element 1 (SW1) and switching element 2 (SW2) were connected in tandem with the forward-out links of SW1 connected to the forward-in links of SW2 and the forward-out links of SW2 connected back to the forward-in links of SW1. The individual queue-out links of each switch were connected back to their own queue-in inputs. We thought this two node model would successfully capture the dynamics of to-and-fro retrograde movement of distance-zero cells. However, model results were no better than what our current model could predict.

A four node model, an extension of the two node model was also attempted. The four node model consisted of four switching elements. SW1, SW2, SW3, and SW4. One of the forward-out links of each of SW1, SW2, and SW3 was connected to SW4. The forward-out links of SW4 were connected back to the SW1, SW2, and SW3. The model was quite complex to solve, but this also did not produce better results.

Previous researchers did not consider restricted delivery. Consequently, they did not encounter model inaccuracies with high load and limited delivery. The intent of the present research was to keep the model as general as possible. We conclude that we must question the existing approach of building a model based on a single switching element. The single node model failed to capture the correlated behaviour that prevails under some conditions.

However, the one node model is accurate for cases of \( d + 1 \) delivery, two cell delivery, and up to 60% load value of single delivery. It is important to note that the model is highly accurate under all conditions in which the switch performs well - the range of conditions in which the model begins to fail are the conditions under which we would not want to operate the switch.

The following subsection addresses an important design issue regarding the number of delivery channels each switching element should have to provide an acceptable performance.
6.5.1 The Optimum Number of Delivery Channels

Providing a large number of delivery channels incurs cost but does not result in significant performance improvement. Therefore, an important design question is how many delivery channels each switching element should have to obtain acceptable performance. The answer to this question can be found in part by computing throughput bounds of the switch, i.e., the maximum number of cells that can be delivered per unit time. The two different bounds [52] that limit the maximum achievable throughput in a hypercube switch are as follows:

- **Absolute Consumption Bound**
  This is determined by the number of delivery channels assumed in a switching element. Throughput of a switching element can not be more than the number of deliveries allowed per CEC. In the case of a single delivery channel, a switching element can deliver a maximum of one cell per CEC and hence the maximum throughput of a switch is $2^d$ cells per CEC, where $d$ is the dimension of a hypercube. Two delivery channels can realize a throughput of $2 \times 2^d = 2^{d-1}$ cells. In case of $d+1$ delivery channels the maximum throughput is $(d+1) \times 2^d$.

- **Bisection Bound**
  This bound is determined by the number of links that are required to be snapped at the bottleneck point in order to partition the switch into two completely separate pieces. The number of links connecting the two parts determine how many cells can be transmitted per CEC from one piece to the other. If $C$ is the probability that a cell is likely to cross the snapped links and $N_{\text{snap}}$ is the number of snapped links, then the maximum throughput should be $\leq \frac{1}{C} \times N_{\text{snap}}$. For a uniform random traffic as assumed in the model, the proportion of traffic crossing a bisection would be $\frac{1}{2}$, i.e., there is a 50% chance that a cell will cross from one piece of the switch to the other. The number of snapped bidirectional links $N_{\text{snap}}$ in a $d$-dimension switch is $2^{d-1}$. The total number of snapped links is equal to $2 \times 2^{d-1} = 2^d$. Therefore, the bisection bound is given by $2^{d+1}$.
The minimum of the above two bounds determine the bounds for switch throughput [3]. The consumption bound is independent of the traffic pattern. The bisection bound depends on whether the traffic is uniform or nonuniform. We can see from the above two bounds that when the number of delivery channels is equal to 1, the consumption bound determines the bound on switch throughput. Both the consumption bound and the bisection bound become equal when the number of delivery channels is equal to two. For more than two delivery channels the bisection bound determines the throughput. The switch throughput can not be improved any further by providing more than two delivery channels. Therefore, the optimum number of delivery channels that should be provided in a switching element is equal to 2. This agrees with our model results.

In summary, it was shown that designing switches with \( d + 1 \) delivery channels is not necessary. Useful practical switches can be made with two delivery channels.

### 6.6 Summary

This chapter begins with a presentation of the numerical results obtained by iterative solution of the model equations that were developed for a buffered deflecting hypercube switch. The model results for the three performance metrics of a dimension 3 hypercube switch with 4 delivery channels were presented. It was shown that changes in queue buffer size impact the retrograde performance as well as cell loss ratio figures of the switches. Similar observations were made in case of a higher dimension switch, i.e., a dimension 4 switch.

Accuracy of the numerical model was established by validating model results against detailed simulation results, for which a switch simulator was built. The model data and the simulation data are compared for the selected performance metrics. Results indicate good agreement between the model and the simulation data. Some of the key design parameters like the queue buffer size, the dimension of the cube, and the injection load were changed to study the impact on the performance figures. Performance of switches with slightly different architectures were also compared which
established further accuracy of the numerical model.

The numerical model is capable of providing significant insight into switch behaviour. The steady-state probability distributions of the various forward-in and queue-in vector components can provide answers to specific design questions which can be used to develop guidelines for design and engineering of hypercube switches. Some of these interesting results are discussed.

Both numerical model data and simulation results are used to show that hypercube switches with queuing buffers and retrograde features have superior performance than either buffered switches or deflection switches. The mixed approach achieves a balance between cell loss ratio and cell latency.

One of the four important design parameters that impact the performance of the switch was identified as the number of delivery channels. In all the examples presented so far, the number of delivery channels was \( d + 1 \), where \( d \) is the dimension of the hypercube. The assumption of \( d + 1 \) delivery channels was removed and performance study was made on a dimension 3 switch with 1 and 2 delivery channels. The three performance metrics were compared. The model and the simulation match closely in almost all cases except the case of single delivery channel. For a single delivery channel the results are increasingly inaccurate above a load value of 0.6. The reasons for the failure of the model to predict performance figures above 0.6 were found out from a simulation trace and were attributed to the development of congested spots in the switch which break the assumptions of uncorrelated traffic. A design insight on the optimum number of delivery channels is found by computing the bounds on the throughput performance of a switch.

The numerical model we built was based on a single node model. i.e., the model of a single switching element of a \( d \) dimension hypercube switch that has a total of \( 2^d \) elements. The idea was that the model could scale well with the dimension of the switch. For dimension less than or equal to 4, the model scales well with an increase in number of switching elements or nodes in a switch. However, the state space for a single node model increases exponentially with an increase in dimension. As a result, the run time requirements of the model become a limiting factor for switches above
dimension 4.

The out-of-order model developed in Chapter 5 is validated in the next Chapter.
Figure 6.25: Average Number of Cells in Queue Buffer for a Dimension 3 Hypercube Switch under Single Delivery Buffer Assumptions. Queue Size = 1

Figure 6.26: Average Number of Cells Retrograded for a Dimension 3 Hypercube Switch under Single Delivery Assumptions. Queue Size = 1.
Figure 6.27: Probability Mass Functions of the Queue-In Vector Space of a Dimension 3 Switch with 3 Queue Buffers for Two Different Numbers of Delivery Channels

Figure 6.28: Probability Mass Functions of the Forward-In Vector Space of a Dimension 3 Switch with 3 Queue Buffers for Two Different Numbers of Delivery Channels
Figure 6.29: Average Number of Cells in Queue Buffer for a Dimension 3 Hypercube Switch under Two Delivery Buffer Assumptions. Queue Size = 1

Figure 6.30: Average Number of Cells Retrograded for a Dimension 3 Hypercube Switch under Two Delivery Buffer Assumptions. Queue Size = 1
Chapter 7

Validation of Out-of-orderness Model

While binary hypercube switches use queueing buffers and retrograding to avoid cell loss due to congestion, one problem arises: out-of-order arrival of cells at their destinations. Different cells in the same conversation can experience different latencies through the switch. If the variation in latency exceeds the inter-cell time interval, then the cells can arrive at the destinations out-of-order. A numerical model based on the behaviour of a single tagged cell was built in Chapter 5 to study this out-of-order phenomenon. Unlike Chapter 4, where we considered a one node model of a d-dimension hypercube switch, the out-of-order model uses a one cell model. A single tagged cell en route to its destination may be in the queue-in buffer or in the forward-in buffer of a switching element as a result of successful forwarding or retrograding from the neighboring node during the previous CEC. If the tagged cell has an initial hopcount equal to i, then the minimum latency, i.e., minimum number of CECs required by the cell before being delivered at a switching element is also equal to i. However, queueing and retrograding may cause extra latency in the delivery of the tagged cell. The probability distribution of a tagged cell with an initial hopcount equal to i taking j cycles, \( i \leq j \) before being delivered at its destination is called the delay distribution \( dd_{ij} \). As discussed in Chapter 5, the proposed numerical model
estimates the delay distribution of the tagged cell, which then can be used to make an exact estimate of the out-of-orderness figures, which is not done in this thesis. These delay distributions form the critical base data from which individual switch designers can derive actual out-of-order results which in turn can be basis for designing their re-ordering queue depths. We validate the accuracy of the out-of-orderness model by validating the delay distribution \( dd_{ij} \) itself against simulation. The equilibrium probability values of the forward-in vectors and the queue-in vectors are used to estimate \( dd_{ij} \).

Section 7.1 presents the numerical model data, following which the simulation data is presented in Section 7.2. Section 7.2 compare both model and the simulation data to establish the accuracy of the model. The chapter ends with a summary in Section 7.3.

7.1 Model Results

The four input parameters that need to be specified before the numerical model is run are: (1) the dimension of the switch, (2) the size of the queue buffers, (3) the number of delivery channels, and (4) the injection load. As in Chapter 6, we assume a dimension 3 hypercube with \( d + 1 \), i.e., 4 delivery channels, unless otherwise specified. The size of the queue buffer is varied between 1 and \( d \). The injection load can be varied from 0.5 to 1.0. However, in order to guarantee that cells are generated every CEC, we keep the load value fixed at 1.0 in the following numerical results. Assuming cells generated every CEC will be accepted for delivery, a load value of 1.0 will be the worst load for these types of switches. However, it should be noted that for switches with \( d + 1 \) delivery channels, this is not a worst-case load.

The value of \( i \), i.e., the initial hopcount of a newly generated cell accepted for routing may vary between 0 and \( d \). Therefore, in a dimension 3 hypercube, the values of \( i \) are 0, 1, 2, and 3. The value of \( j \) in \( dd_{ij} \), can range, theoretically, between \( i \) and infinity, the reason being the cell may be queued or retrograded \( ad \ infinitum \). A reasonable upper bound on the value of \( j \) was taken as \( 4 \times d \), where \( d \) is the
dimension of the hypercube. This assumption is justified by numerical results that show the probability a cell with an initial hopcount $i$ will take more than $4 \times d$ CEC is vanishingly small.

Figures 7.1, 7.2, and 7.3 present the delay distributions of cells in a dimension-3 hypercube with $d + 1$ delivery channels for an initial hopcount equal to 1, 2 and 3 respectively. The load value is chosen 1.0. Note that, the probability with which a tagged cell with an initial hopcount equal to 1, 2, or 3 takes more than $4 \times d$, i.e., 12 cycles for delivery at its destination is vanishingly small ($10^{-11}$ or less). This justifies the assumption about the upper bound on the value of $j$ in delay distributions $dd_{ij}$.

Figure 7.1: Model Data for Delay Distributions of Cells with Hopcount 1 for a Dimension 3 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size = 1. and Load = 1.0

Figure 7.2: Model Data for Delay Distributions of Cells with Hopcount 2 for a Dimension 3 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size = 1. and Load = 1.0
Figure 7.3: Model Data for Delay Distributions of Cells with Hopcount 3 for a Dimension 3 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0

For a dimension 4 hypercube switch with 5 delivery channels and queue buffer size equal to 1. Figures 7.4, 7.5, 7.6, and 7.7 present the delay distributions of cells. The initial hopcounts range between 1 and 4. The upper bound on the value of j is taken as 16 cycles.

Figure 7.4: Model Data for Delay Distributions of Cells with Hopcount 1 for a Dimension 4 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0

7.2 Simulation Results and Model Validation

Simulation is carried out on a dimension-3 hypercube with d + 1 delivery buffers. Cells are injected every CEC at a load of 1.0. This is done specifically to study the
Figure 7.5: Model Data for Delay Distributions of Cells with Hopcount 2 for a Dimension 4 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0

Figure 7.6: Model Data for Delay Distributions of Cells with Hopcount 3 for a Dimension 4 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0

worst-case out-of-orderness performance of the switch. Statistics are collected for all the cells that were generated and finally delivered at their destinations. Forwarding counts, queueing counts, and retrograding counts of all the cells are measured to find the average latency figures, i.e., average forward latency, average queue latency, and average retrograde latency of the cells. The total latency counts of the cells are compared against model data as a sanity check for the accuracy of the out-of-order model.

To get confidence in the statistical simulation data, the simulations are run for 510000 cycles using the batch means method with 5 batches of 100000 steps in each
Figure 7.7: Model Data for Delay Distributions of Cells with Hopcount 4 for a Dimension 4 Hypercube Switch with $d+1$ Delivery Buffers. Queue Size = 1. and Load = 1.0

batch. The results of the first 10000 cycles are discarded. Each data value lies within a 95% confidence interval of width less than 2.5% of the value. Appendix A shows the technique of computing simulation run length.

The total latency of cells are shown for a dimension 3 hypercube with variation in injection load in Figure 7.8. Both model and simulation are seen to be in close agreement.

Figure 7.8: Variation of Total Latency of Cells in a Dimension 3 Hypercube ($d+1$ Delivery Buffers. Queue Size = 1. and Load = 0.9) with Injection Load

Comparisons of model delay distributions with simulation data are shown in Figures 7.9, 7.10, and 7.11. We observe that the model and the simulation are in close agreement up to a figure of $10^{-6}$, beyond which the inaccuracy is due to small numbers. To capture those areas the simulation needs to be run for many cycles. The
figures also indicate that even under worst-case load situation, the probability of having cells out-of-order by more than one degree is very low. Therefore, the advantages of having queueing and retrograding scheme outweigh the disadvantages of the scheme.

![Graph](image)

Figure 7.9: Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 1 for a Dimension 3 Hypercube Switch with \(d+1\) Delivery Buffers. Queue Size = 1, and Load = 1.0

![Graph](image)

Figure 7.10: Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 2 for a Dimension 3 Hypercube Switch \(d+1\) Delivery Buffers. Queue Size = 1, and Load = 1.0

Figures 7.12, 7.13, 7.14, and 7.15 present the delay distributions of a dimension 4 hypercube with initial hopcount values equal to 1, 2, 3, and 4 respectively. As seen in the figures, even for higher dimension switches the results are accurate.
Figure 7.11: Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 3 for a Dimension 3 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0

Figure 7.12: Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 1 for a Dimension 4 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0

7.3 Summary

Both model and simulation data are presented to study out-of-order arrival of cells in the switch. It is shown that the probability of a tagged cell arriving before a cell transmitted three or more CECs earlier is vanishingly small.
Figure 7.13: Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 2 for a Dimension 4 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0

Figure 7.14: Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 3 for a Dimension 4 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0

Figure 7.15: Comparisons of Model Data with Simulation Data for Delay Distributions of Cells with Hopcount 4 for a Dimension 4 Hypercube Switch with d+1 Delivery Buffers. Queue Size = 1. and Load = 1.0
Chapter 8

Conclusions and Future Work

8.1 Thesis Summary

B-ISDN networks will require new high-performance switches. Under ATM, a single-class high performance cell switching network would carry voice, video and data, in the form of short, fixed-size cells. Among the many different cell switching systems proposed over the past few years, binary \( n \) cubes (or hypercubes) have shown promise for one such class of high-performance cell switches. Distributed switching fabrics, regular topology, fault tolerance, and multiple paths for routing cells make these switches attractive. In addition, these switches can employ deflection routing to achieve zero cell loss performance.

Our goal in this thesis has been a performance study of hypercube switches that have shared queuing buffers and use deflection routing. This study is important to properly design, provision, and control these switches. We assume, in our performance study, a deflection routing algorithm called the SCS (Saturated Constant Shuffle) algorithm that works well with buffered hypercube switches.

We have proposed an analytic model to study buffered deflection hypercube switches. The presence of distributed logic, multi-path routing, deflection routing, and queueing buffers make the modeling task highly challenging. An approximate Markov model of one of the switching elements has been developed to capture the behaviour an

162
entire switch system. Based on the Markovian characteristic of the model, an iterative procedure has been developed to solve the model numerically. The numerical procedure starts with a known initial state and computes time step by time step the distributions of the output states of the element. When the numerical solution converges, a steady-state distribution is realized which is then used to compute the various performance metrics.

While direct binary hypercube switches hold promise for future B-ISDN applications, they suffer from one drawback. The cells belonging to a particular traffic stream may not be delivered at their destinations in sequence. This phenomenon, known as out-of-orderliness of cells happens because of the presence of multiple paths, queueing buffers, and retrograde motions of the cells in the switch. One solution that ensures in-order delivery of cells at their destinations is provision of re-ordering hardware at each sink. But the finite size of the re-ordering buffer may fail to resequence some of the cells that are more out-of-sequence than the buffer can handle. It is important to model this out-of-orderliness phenomenon. Studies show that the probability of cells getting severely out-of-order is very low. A second objective of this thesis has been to capture this out-of-orderliness phenomenon by building an analytic model. We derive the delay distributions of a typical tagged cell, and claim that these distributions can be used to compute out-of-orderliness figures.

Switch simulators were built and the results of the simulation are used to validate the numerical model results. The steady-state probability distributions of the various forward-in and queue-in vector components provide significant insight into the switch's behaviour and can be used to develop guidelines for design and engineering of hypercube switches. The impact of some of the key design parameters on switch performance is discussed. The model data and the simulation data are compared for selected performance metrics. The model and the simulation match in almost all cases except in the case of a single delivery channel. A design insight on the optimum number of delivery channels is found by computing the bounds on the throughput performance of a switch.

Using both numerical model data and simulation results the superior performance
of buffered deflection hypercube switches, compared to buffered switches or deflection switches, is established.

### 8.2 Conclusions

Cell switches based on hypercube fabrics with queue buffers and deflection routing are promising. They have superior performance compared to buffered switches or deflection switches. In order to design these switches, it is important to understand the performance capabilities of these switches. Both analytical models and simulation have been used to evaluate performance of these switches. The presence of queue buffers and deflection routing make the modeling task highly challenging. To the best of author’s knowledge, there are no analytic models available in the literature that study performance of buffered hypercubes employing deflection routing. This thesis provides the first model of these switches. Model results are in agreement with simulation data for the assumed uniform traffic with Bernoulli injection. Significant computation time and memory requirements prevented us from extending the validation of the model for switches with dimension greater than four. This was due to the exponential increase of state space of the single node model, with increase in dimension of the switches. A model based on a single cell instead of single node, or a model based on a combination of the single node and single cell model, is promising for higher dimension switches and is left as future research.

One of the intentions of the current research has been to keep the model as general as possible. Previous researchers did not consider restricted number of delivery channels. Consequently, they did not encounter model inaccuracies with high load and limited number of delivery channels. We attempted to remove this limitation and incorporate a flexible number of delivery channels in our model. However, the model failed to capture switch behaviour for a single delivery channel. Although, it is true that the model is accurate under all conditions in which the switch performs well and the range of conditions in which the model begins to fail are the conditions under which we would not want to operate the switch, it will be worth extending the
model for a single delivery channel: this is left for future research. This thesis has laid a solid groundwork for further research into some of the areas listed above.

8.3 Contributions

The main contributions of this thesis are as follows:

- Development of a new analytical model of a buffered hypercube switch employing deflection routing. The probabilistic model is accurate and reveals significant insight into the switch's behavior that can be used to design and engineer $d$-dimension hypercube switches.

- Development of an analytic model based on a single tagged cell to capture out-of-order arrival of cells at switching elements which is seen as a potential drawback of these hypercube switches.

- Critical evaluation of the various analytical models that have been used or proposed for predicting performance of a variety of switches.

- Development of a novel approach to analytical modeling based on the idea of probability tree analysis.

At the highest level, this thesis develops new modeling techniques to produce the first model which incorporates both queueing and deflection routing in binary hypercubes. The model was carefully validated by simulations.

8.4 Future Work

This thesis concludes with the above main contributions and findings. Based on the results of this thesis, several interesting studies could be usefully conducted to further improve and extend these achievements.

1. One of the four important design parameters that impacted the performance of the hypercube switch is the number of delivery channels. We assumed in most
of our study that the number of delivery channels in a $d$-dimension hypercube is $d + 1$. This was in line with the assumptions made by previous researchers. However, when we restricted the number of delivery channels to only 1 channel, a significant discrepancy between the model and the simulation data was observed. The model was seen to behave accurately up to a load value equal to 0.6, beyond which it progressively inaccurate. An investigative study into the reason of failure of the model to capture switch behavior at high load with one delivery channel revealed the development of congested spots in the switch.

Although we would not operate a hypercube switch with only one delivery channel, it will be worth investigating an extension of the model to capture switch behaviour with only one delivery channel. To capture the to-and-fro movement of distance zero cells along the links, a time correlation or dependence between the arrivals of cells on consecutive CEC needs to be modeled, which may be difficult.

2. In order to keep the modeling tasks tractable, the injection traffic model was assumed to be Bernoulli, which is a simple model. Performance study of hypercube switches dictate more practical traffic models like Binomial models or two-state ON/OFF Markov models. Building analytical models with these complex traffic models would be a useful, although nontrivial extension of the current research. A more ambitious extension would be using a self-similar traffic model.

3. SCS routing algorithms may have subtle variants, as we discussed in Chapter 2. The routing strategy can be different depending on how the queued cells, the incoming cells, and the locally sourced cells are assigned outgoing links. We can have two possible link assignment strategies: one-pass, and two-pass. Similarly, the incoming cells may be considered in various orders for assignment to the outgoing links. They may be examined: (a) in random order, (b) in some fixed order (the order of the dimensions of the cube), (c) in order of increasing distance of the cells from their destinations, or (d) in order of decreasing distance
of cells from their destinations.

In this thesis, we just considered only one variant of SCS routing algorithm. We considered the one-pass, nearest-first routing strategy. In order to have a complete performance knowledge of the hypercube switches, it would be a worthwhile extension to carry out similar modeling work for other variants of the SCS routing algorithm.

4. Finally, the numerical model we built was based on a single node model, i.e., the model of a single switching element of a \( d \) dimension hypercube switch that has a total of \( 2^d \) elements. The idea was that the model could scale well with increasing dimension of the switch. However, the state space, even for a single node increases exponentially with increase in dimension. As a result, the run time requirements of the model become a limiting factor for switches above dimension 4. Although, the limit to dimension 3 or dimension 4 hypercubes is not important and simulations show that performance scales very smoothly for larger dimension switches, a model based on a single tagged cell, as developed for studying out-of-orderness issues, is possibly immune to the explosion of state space. Analytical models based on a single tagged cell approach or a hybrid approach using both single node and single cell will be an useful future work for studying hypercubes with dimension above 4.
References


[48] C. McCrosky and I. Reid. "Fixed Length Message Routing in Hypercube- 
connected Multiprocessors". Technical Report. no. 86-14. Department of Com-


House Inc.. 1995.


[54] N. Shacham and D. Towsley. "Resequencing Delay and Buffer Occupancy in 
Selective Repeat ARQ with Multiple Receivers". IEEE Transactions on Com-


Switch Architecture for Asynchronous Transfer Mode". IEEE International Con-

California.

[58] F. Tobagi. "Fast Packet Switch Architectures for Broadband Integrated Services 


Appendix A

Simulation Run Length

The length of the simulation run is determined by the desired confidence level of the conclusions drawn from a set of simulation data. If the simulation is too short, the size of the data set is small and therefore, the confidence level of the conclusions inferred from the data set is low. On the other hand, if the run length is too long, the associated confidence level is much higher, but takes more computation time and is therefore expensive. The run length is a compromise between a given level of accuracy and lengthy computation time.

Suppose we want to estimate the mean performance of a system parameter whose actual mean (also called the population mean) is $\mu$ and standard deviation is $\sigma$. We take a sample of $n$ observations $\{x_1, x_2, \ldots, x_n\}$ and compute the sample mean $\bar{x}_1$ which is likely to be different from $\mu$. We may consider another sample of $n$ observations and compute the second sample mean as $\bar{x}_2$. If we take $k$ different samples, we have $k$ different estimates of population mean $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_k$. It is not possible to get a perfect estimate of $\mu$ from any finite number of finite size samples [34]. But one can make a probabilistic statement about the range in which the population mean lies. If $c_1$ and $c_2$ are the two bounds, the probability that the estimate of the population mean is in the region $(c_1, c_2)$ will be given by

$$Pr\{c_1 \leq \mu \leq c_2\} = 1 - \alpha \quad (A.1)$$
The interval \((c_1, c_2)\) is known as the confidence interval for the population mean. and \(\alpha\) is called the significance level and 100(1 - \(\alpha\)) is called the confidence level. A 90% confidence interval may use 5 percentile and 95 percentile of the sample means as the two bounds. Given \(k\) samples sorted in ascending order, the two bounds would be the \([1 + 0.05(k - 1)]\)-th and \([1 + 0.95(k - 1)]\)-th elements of the sequence.

It is not necessary to gather too many samples to achieve a certain confidence level [34]. From just one sample, one can determine the confidence interval. The Central limit theorem states that if the observations in a sample \(\{x_1, x_2, \ldots, x_n\}\) are independent and are taken from the same population with mean \(\mu\) and standard deviation \(\sigma\) then the sample mean for a large sample size would follow a normal distribution with mean \(\mu\) and standard deviation \(\sigma/\sqrt{n}\). The standard deviation of the sample mean is also called standard error. Therefore, as \(n\) increases, the standard error decreases. If \(\bar{x}\) is the sample mean, \(s\) is the sample standard deviation, and \(n\) is the sample size, then according to the Central limit theorem, a 100(1 - \(\alpha\))% confidence interval for the population mean will be given by

\[
(\bar{x} - z_{1-\alpha/2}s/\sqrt{n}, \ \bar{x} + z_{1-\alpha/2}s/\sqrt{n})
\]

where \(z_{1-\alpha/2}\) is the \((1 - \alpha/2)\)-quantile of a unit normal variate. The values for these quantiles are listed in [34].

For sample sizes smaller than 30, a 100(1 - \(\alpha\))% confidence interval is given by:

\[
(\bar{x} - t_{[1-\alpha/2;n-1]}s/\sqrt{n}, \ \bar{x} + t_{[1-\alpha/2;n-1]}s/\sqrt{n})
\]

where \(t_{[1-\alpha/2;n-1]}\) is the \((1 - \alpha/2)\)-quantile of a \(t\)-variante with \(n - 1\) degrees of freedom. These quantiles are also listed in [34]. The pdf of a \(t\)-variante is similar to that of a unit normal. The distribution is bell shaped and symmetric about zeroes. For large degrees of freedom, i.e., \((n > 30)\), a \(t\)-distribution approximates a unit normal.
A.1 Determining Sample Size

It is evident that the confidence level of the conclusions drawn from a set of measured data depends upon the size of the sample set. The larger the sample size, the higher is the confidence and the higher is the cost. Thus, it is important to determine the smallest sample size that can provide the desired confidence.

Suppose we want to estimate the mean performance of a system with a confidence level of $100(1 - \alpha)\%$ within an accuracy limit of $\pm r\%$. The number of observations $n$ needed to realize this goal is determined as follows:

The $100(1 - \alpha)\%$ confidence interval of the population mean is given by $\bar{x} \mp z \frac{s}{\sqrt{n}}$. The $r$ percent accuracy implies that the confidence interval is $(\bar{x}(1 - r/100), \bar{x}(1 + r/100))$. Therefore,

$$\bar{x} \mp z \frac{s}{\sqrt{n}} = \bar{x}(1 \mp \frac{r}{100})$$

$$z \frac{s}{\sqrt{n}} = \frac{r}{100}$$

$$n = \left(\frac{100z s}{r \bar{x}}\right)^2$$

where $z$ is the normal variate of the desired confidence level.

The above formula can be used if the observations in a simulation are independent, which unfortunately, is not true in most of the simulations. If the observations are correlated, the variance of the mean, i.e., $Var(\bar{x})$ may be several times larger than $Var(x)/n$. This may result, if ignored, in narrow confidence intervals and therefore premature termination of simulation. A number of methods have been developed by statisticians to compute the variance of the mean of correlated observations. We use batched means method in which $n$ observations are partitioned into $M$ batches, with $n' = n/M$ observations per batch. The mean and the confidence interval are
calculated from those of the $M$ batches. For each batch, the means $\overline{x}_i$ are given by

$$\overline{x}_i = \frac{1}{n'} \sum_{j=1}^{n'} x_{ij}, \ i = 1, 2, \ldots, M.$$  

The overall mean is given by

$$\overline{x} = \frac{1}{M} \sum_{i=1}^{M} \overline{x}_i.$$  

Now, we can compute the variance of batch means using the following formula:

$$Var(\overline{x}) = \frac{1}{M - 1} \sum_{i=1}^{M} (\overline{x}_i - \overline{x})^2$$

Therefore, the confidence interval for the mean is given by $[\overline{x} \pm z_{1-\alpha/2} \sqrt{Var(\overline{x})}]$. The batch size $n'$ must be large so that the batch means have little correlation. One way to ascertain the batch size is to compute the covariance (also known as autocovariance) of the successive batch means:

$$Cov(\overline{x}_i, \overline{x}_{i-1}) = \frac{1}{M - 2} \sum_{i=1}^{M-1} (\overline{x}_i - \overline{x})(\overline{x}_{i-1} - \overline{x})$$

The batch size $n'$ is increased continually until the auto-covariance of the batch means is small compared to their variance.

### A.2 Transient Removal

In most simulations, only the steady-state values of the metrics are of interest. Results for the initial part of the simulation should not be considered in the final computations. This initial part is also called the transient state. The problem of identifying the end of the transient state, i.e., the period $T$ from the empty initial state during which data is rejected, is usually determined by inspection or heuristically. One other popular method is to use truncation which works on the assumption that the variability during the steady-state is less than that during the transient state. We don't
employ any formal transient removal techniques. Instead, we follow an inspection approach where the results of the first 10000 cycles are discarded.
Appendix B

The Vector Spaces

In Chapter 4, we defined the class of a cell as the distance of the cell from its destination. In a d-dimension hypercube, a cell can have distance values between 0 and d. Therefore, the maximum number of possible classes are \( (d + 1) \). All cells in a switching element must belong to one of these classes. In general, if \( N \) is the total number of cells in an element during any CEC, and \( k \) is the number of classes, then the number of possible ways in which \( N \) cells can be grouped into \( k \) classes is given by:

\[
N_k = \binom{N + k - 1}{k - 1}.
\] (B.1)

Each of these cell-class combinations is called a vector. A set of vectors is called a vector space. Vector spaces can be associated with queue-in buffers \( (QI_{vect}) \), queue-out buffers \( (QO_{vect}) \), forward-in buffers \( (FI_{vect}) \), forward-out buffers \( (FO_{vect}) \), and injection buffers \( (INJ_{vect}) \). Since each vector is unique, it is possible to index them.

A generalized tuple notation for, say, the \( j \)-th vector or combination will be given by \( < n_j : n_{0j}, n_{1j}, n_{2j}, \ldots n_{dj} > \) where \( n_j = \sum_{i=0}^{d} n_{ij} \), is the total number of cells in \( j \)-th vector. \( n_{0j}, n_{1j}, \ldots \) represents the number of cells belonging to class 0, class 1, and so on. If the maximum size of the queue buffer is equal to \( Q \), the maximum number
of vectors in $Q_{I_{\text{vect}}}$ or $Q_{O_{\text{vect}}}$ are given by:

$$|Q_{I_{\text{vect}}}| = |Q_{O_{\text{vect}}}| = \sum_{n=0}^{n=q} \binom{n + d}{d}.$$  \hspace{1cm} (B.2)

where $|Q_{I_{\text{vect}}}|$ and $|Q_{O_{\text{vect}}}|$ indicate the cardinalities of the queue-in and queue-out vector spaces, respectively. Similarly, $d$ being the maximum number of forward-in or forward-out buffers, one can enumerate all the possible vectors of forward-in ($F_{I_{\text{vect}}}$) and forward-out ($F_{O_{\text{vect}}}$) vector space.

For a dimension-3 hypercube, with queue buffer sizes equal to 3, the queue-in vector space $Q_{I_{\text{vect}}}$ and queue-out vector space $Q_{O_{\text{vect}}}$ each have 35 possible entries. The mapping between these vectors and the unique index numbers are shown in Table B.1.

The same mapping holds for forward-in as well as forward-out vector space of a dimension 3 hypercube. There are 35 possible entries for these vector spaces as well.
Table B.1: Mapping between the Vector Space Components and the Index Space

<table>
<thead>
<tr>
<th>Index No.</th>
<th>dist=0</th>
<th>dist = 1</th>
<th>dist = 2</th>
<th>dist = 3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>31</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>33</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>34</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>