Performance of Multistage Interconnection Networks under Non-Uniform Reference Model

by

Mohammad Shahed Akhtar

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

In

ELECTRICAL ENGINEERING

January, 1992
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313-761-4700 800/521-0600
Performance of multistage interconnection networks under non-uniform reference model

Akhtar, Mohammad Shahed, M.S.
King Fahd University of Petroleum and Minerals (Saudi Arabia), 1992
PERFORMANCE OF MULTISTAGE INTERCONNECTION NETWORKS UNDER NON-UNIFORM REFERENCE MODEL

BY

MOHAMMAD SHAHED AKHTAR

A Thesis Presented to the
FACULTY OF THE COLLEGE OF GRADUATE STUDIES
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAH, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In

ELECTRICAL ENGINEERING

JANUARY 1992
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHHRAN, SAUDI ARABIA

This thesis, written by

MOHAMMAD SHAHED AKHTAR

under the direction of his thesis committee, and approved by all the members, has been presented to and accepted by the Dean, College of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

Thesis Committee

A. Zaman
Chairman (Dr. Mohammad Atiquzzaman)

Dr. Al-Shehri
Department Chairman

Dr. Abdulelah N. Al-Shehri

Dr. Ala H. Rabeh
Dean College of Graduate Studies

Date: 2014
Dedicated

to my

Parents & Brothers
ACKNOWLEDGEMENT

Praise and gratitude be to Allah the Almighty, with whose gracious help it was possible to accomplish this work. Acknowledgement is due to King Fahd University of Petroleum and Minerals for extending all the facilities and providing financial support.

I would like to express my appreciation and gratefulness to my advisor and committee chairman Dr. Mohammad Atiquzzaman. His guidance, innovative and pragmatic suggestions and continuous encouragement were of vital help in achieving this goal. I am also thankful to my committee members Dr. Mammar Bettayeb, Dr. Syed Zaki Al-Akhdar and Dr. Khalid Hussain Biyari for their critical evaluation and suggestions for the improvement of the work. Thanks are also due to The Chairman, Department of Electrical Engineering for extending computer facilities for CPU-hungry, time-taking simulations.

Special thanks to my brother Mr. M. Wasim Akhtar, whose suggestions and timely help expedited the completion of this work. Last, but not the least, I am thankful to all the faculty, staff, colleagues and friends who made my stay at K.F.U.P.M a memorable and valuable experience.
# TABLE OF CONTENTS

Abstract ........................................................................................................... xxiii

CHAPTER I: INTRODUCTION ...................................................................... 1

Overview of Parallel Processing ................................................................. 2
Interconnection Networks ........................................................................... 4
- Classification of parallel / distributed systems ................................. 6
- Classification of Interconnection networks ...................................... 7
  - Classification based on Network Topology .................................. 7
  - Classification based on operation mode ....................................... 8
  - Classification based on switching methodology ......................... 8
  - Classification based on control strategy .................................... 8
Basic Performance measures .................................................................... 9
Complexity of IN's .................................................................................. 10
Assumptions in Literature ........................................................................ 11
Non-Uniform References ......................................................................... 12
Literature Survey ..................................................................................... 13
- Literature for crossbar ................................................................. 13
  - Study under URM ........................................................................ 14
  - Study under localized/non-uniform references ......................... 15
- Literature for multiple-bus interconnection networks ............... 16
  - Study under the URM ............................................................... 16
  - Study under non-uniform / localized reference model .......... 19
- Literature for multistage interconnection networks( MSINs ) ........ 20
  - Literature on construction, design and operation ................. 20
  - Study under URM ..................................................................... 21
  - Study under non-uniform / local references ......................... 23
The Problem ........................................................................................... 24
Summary .................................................................................................... 25

CHAPTER II: MULTISTAGE INTERCONNECTION NETWORKS ............... 27

Introduction ............................................................................................. 28
Delta Network .......................................................................................... 28
Shuffle interconnection .......................................................................... 32
Use of shuffle interconnection in INs .................................................. 34
Actual routing strategy ........................................................................... 37
Conflict ....................................................................................................... 41
CHAPTER III: PERFORMANCE EVALUATION OF UNBUFFERED NETWORKS

Non-Uniform Memory References ........................................... 53
Hotspots .............................................................................. 54
Model .................................................................................. 55
Properties ............................................................................ 58
Performance Analysis .............................................................. 64
Simulations ............................................................................ 84
Request Generation ................................................................. 86
Switching Operation ............................................................... 89
Parameters evaluated ............................................................. 89
Results and Discussion ......................................................... 90
Comparison of different INs ................................................... 114
Summary .............................................................................. 117

CHAPTER IV: PERFORMANCE OF SINGLE-BUFFERED NETWORKS

Introduction ............................................................................ 118
Non-Uniform Memory References: ....................................... 119
Tree Saturation ..................................................................... 119
Methods to control Tree Saturation ....................................... 120
Analysis of Buffered Networks .............................................. 121
General Assumptions ............................................................. 122
Description of Network operation: ....................................... 125
Markov Process .................................................................... 126
Properties of Buffered Networks .......................................... 127
Model 1 .............................................................................. 129
Extension Of Model 1 to Non-Uniform references ................. 139
Simulation ............................................................................ 149
Results .............................................................................. 152
Drawbacks .......................................................................... 156
Model 2 .............................................................................. 159
Extension Of Model 2 to Non-Uniform Reference Model ....... 174
Delay ................................................................................... 181
Results .............................................................................. 183
Model 3 .............................................................................. 186
CHAPTER V: PERFORMANCE ANALYSIS OF MULTI-BUFFERED NETWORKS .......................... 230
Introduction ............................................................................................................. 230
Simulation methodology ....................................................................................... 230
Results .................................................................................................................. 231

CHAPTER VI: CONCLUSION ................................................................. 245
Summary of the thesis work ................................................................................. 245
Suggestions for future work: ................................................................................. 249

REFERENCES .................................................................................................. 251

APPENDIX A ...................................................................................................... 257
LIST OF FIGURES

1.1 GENERAL STRUCTURE OF A PARALLEL PROCESSING SYSTEM .................. 3

1.2 FUNCTION OF AN INTERCONNECTION NETWORK .................. 5

2.1 DELTA NETWORK .................................................. 30

2.2 OMEGA NETWORK ............................................. 31

2.3 PERFECT SHUFFLE OPERATION .................................. 33

2.4 OPERATION OF A CROSSBAR SWITCH ............................. 36

2.5 DESTINATION TAG ROUTING .................................. 38

2.6 ROUTING CONFLICT ......................................... 42

2.7 MEMORY CONFLICT ........................................... 43

3.1 OMEGA NETWORK UNDER THE PRESENCE OF A HOT MODULE ................. 59

3.2 DATA RATE DEPENDENCY TREE FOR AN OMEGA NETWORK ................. 70

3.3 DEMARCATION OF REQUESTS GENERATED BY A PSEUDO-RANDOM GENERATOR ........... 87

3.4 BANDWIDTH vs HOTSPOT PROBABILITY FOR DIFFERENT NETWORK SIZES, REQUEST RATE = 1.0 ................................. 91

3.5 BANDWIDTH vs HOTSPOT PROBABILITY FOR DIFFERENT NETWORK SIZES, REQUEST RATE = 0.8 .................................. 92

3.6 BANDWIDTH vs HOTSPOT PROBABILITY FOR DIFFERENT NETWORK SIZES, REQUEST RATE = 0.6 .................................. 93

3.7 BANDWIDTH vs HOTSPOT PROBABILITY FOR DIFFERENT NETWORK SIZES, REQUEST RATE = 0.4 .................................. 94

3.8 BANDWIDTH vs HOTSPOT PROBABILITY FOR DIFFERENT NETWORK SIZES, REQUEST RATE

- viii -
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9</td>
<td>BANDWIDTH vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.2</td>
<td>97</td>
</tr>
<tr>
<td>3.10</td>
<td>BANDWIDTH vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.4</td>
<td>98</td>
</tr>
<tr>
<td>3.11</td>
<td>BANDWIDTH vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.6</td>
<td>99</td>
</tr>
<tr>
<td>3.12</td>
<td>BANDWIDTH vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.8</td>
<td>100</td>
</tr>
<tr>
<td>3.13</td>
<td>BANDWIDTH vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 1.0</td>
<td>101</td>
</tr>
<tr>
<td>3.14</td>
<td>BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES, REQUEST RATE = 1.0</td>
<td>103</td>
</tr>
<tr>
<td>3.15</td>
<td>BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES, REQUEST RATE = 0.8</td>
<td>104</td>
</tr>
<tr>
<td>3.16</td>
<td>BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES, REQUEST RATE = 0.6</td>
<td>105</td>
</tr>
<tr>
<td>3.17</td>
<td>BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES, REQUEST RATE = 0.4</td>
<td>106</td>
</tr>
<tr>
<td>3.18</td>
<td>BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES, REQUEST RATE = 0.2</td>
<td>107</td>
</tr>
<tr>
<td>3.19</td>
<td>BLOCKING PROBABILITY vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.1</td>
<td>109</td>
</tr>
<tr>
<td>3.20</td>
<td>BLOCKING PROBABILITY vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.3</td>
<td>110</td>
</tr>
<tr>
<td>3.21</td>
<td>BLOCKING PROBABILITY vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.5</td>
<td>111</td>
</tr>
<tr>
<td>3.22</td>
<td>BLOCKING PROBABILITY vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.7</td>
<td>112</td>
</tr>
<tr>
<td>3.23</td>
<td>BLOCKING PROBABILITY vs REQUEST RATE FOR</td>
<td>ix</td>
</tr>
</tbody>
</table>
DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.9 ........................................... 113

4.1 BUFFERS AT THE INPUTS OF SWITCHES ........................................... 124

4.2 A CLOCK CYCLE IN MODEL 1 ......................................................... 132

4.3 STATE DIAGRAM FOR MODEL 1 ....................................................... 137

4.4 FLOWCHART ILLUSTRATING THE SIMULATION PROCEDURE (contd...) .................. 150

4.4 (.... contd) ................................................................................. 151

4.5 BANDWIDTH OBTAINED FROM MODEL 1 AGAINST REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES .............................................................. 153

4.6 PERCENTAGE ERROR IN BANDWIDTH OBTAINED FROM MODEL 1 AGAINST REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES ......................... 154

4.7 DELAY OBTAINED FROM MODEL 1 COMPARED WITH SIMULATION RESULTS, FOR AN 8 x 8 OMEGA NETWORK UNDER URM ........................................ 155

4.8 EFFECT OF HAVING INDEPENDENT REQUESTS IN THE BUFFERS; N = 8, REQUEST RATE = 1 AND HOTSPOT PROBABILITY = 0.3 ........................................... 158

4.9 STATE DIAGRAM FOR MODEL 2 ....................................................... 161

4.10 RESULTS OBTAINED FROM MODEL 2 FOR AN 8x8 NETWORK FOR DIFFERENT HOTSPOT PROBABILITIES .............................................................. 184

4.11 COMPARISON OF THE RESULTS OBTAINED FROM MODEL 2 WITH SIMULATIONS FOR 8x8 NETWORK WITH A HOTSPOT PROBABILITY OF 0.3 ........................................... 185

4.12 STATE DIAGRAM FOR MODEL 3 ....................................................... 188

4.13 COMPARISON OF RESULTS OBTAINED FROM THE THREE MODELS WITH SIMULATION FOR A 8x8 OMEGA NETWORK UNDER URM. ........................................ 202

4.14 BANDWIDTH vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE
BUFFERED 8 x 8 NETWORK ......................... 210

4.15 DELAY (HOT PACKET) vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 8x8 NETWORK ......................... 212

4.16 BANDWIDTH vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 16 x 16 NETWORK ......................... 213

4.17 BANDWIDTH vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 32 x 32 NETWORK ......................... 214

4.18 BANDWIDTH vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 64 x 64 NETWORK ......................... 215

4.19 DELAY (HOT PACKET) vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 16 x 16 NETWORK ......................... 216

4.20 DELAY (HOT PACKET) vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 32 x 32 NETWORK ......................... 217

4.21 DELAY (HOT PACKET) vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 64 x 64 NETWORK ......................... 218

4.22 BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED NETWORK WITH AN INITIAL LOAD OF 1.0 ................................. 220

4.23 BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED NETWORK WITH AN INITIAL LOAD OF 0.8 ................................. 221

4.24 BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED NETWORK WITH AN INITIAL LOAD OF 0.6 ................................. 222

4.25 BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED NETWORK WITH AN INITIAL LOAD OF 0.4 ................................. 223
4.26 BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED NETWORK WITH AN INITIAL LOAD OF 0.2 .......................................................... 224

4.27 COMPARISON OF RESULTS OF THE THREE MODELS WITH SIMULATIONS FOR N = 8 WITH FULL LOAD AND HOTSPOT PROBABILITY OF 0.3 .......................................................... 225

4.28 RESULTS FOR PRIORITISED RESOLUTION OF CONFLICTS N = 8 WITH FULL LOAD FOR DIFFERENT HOTSPOT PROBABILITIES. .................................................. 227

4.29 COMPARISON OF THE RESULTS OBTAINED WITH THE TWO CONFLICT RESOLUTION SCHEMES FOR N = 8 .......................................................... 228

5.1 BANDWIDTH vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.05 FOR DIFFERENT BUFFER SIZES .......................................................... 232

5.2 BANDWIDTH vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.10 FOR DIFFERENT BUFFER SIZES .......................................................... 233

5.3 BANDWIDTH vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.15 FOR DIFFERENT BUFFER SIZES .......................................................... 234

5.4 BANDWIDTH vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.20 FOR DIFFERENT BUFFER SIZES .......................................................... 235

5.5 BANDWIDTH vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.25 FOR DIFFERENT BUFFER SIZES .......................................................... 236

5.6 DELAY (HOT PACKET) vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.05 FOR DIFFERENT BUFFER SIZES .......................................................... 238

5.7 BANDWIDTH vs REQUEST RATE FOR A
4-BUFFERED 8 x 8 OMEGA NETWORK FOR DIFFERENT HOTSPOT PROBABILITIES. ................. 239

5.8 BANDWIDTH vs REQUEST RATE FOR A
4-BUFFERED 16 x 16 OMEGA NETWORK FOR DIFFERENT HOTSPOT PROBABILITIES. ................. 240

5.9 BANDWIDTH vs REQUEST RATE FOR A
4-BUFFERED 32 x 32 OMEGA NETWORK FOR DIFFERENT HOTSPOT PROBABILITIES. ................. 241

5.10 BANDWIDTH vs REQUEST RATE FOR A
4-BUFFERED 64 x 64 OMEGA NETWORK FOR DIFFERENT HOTSPOT PROBABILITIES. ................. 242

5.11 DELAY FOR DIFFERENT MEMORY MODULES vs REQUEST RATE FOR A 4-BUFFERED 8x8 OMEGA NETWORK; HOTSPOT PROBABILITY = 0.3. ................. 243

5.12 BANDWIDTH vs NETWORK SIZE FOR DIFFERENT BUFFER SIZES. HOTSPOT PROBABILITY = 0.15 AND REQUEST RATE = 1.0. ................. 244
# List of Tables

3.1 PERCENTAGE ERRORS BETWEEN ANALYTICAL AND SIMULATION RESULTS FOR DIFFERENT N . . . 102

3.2 COMPARISON OF DIFFERENT INs FOR DIFFERENT HOTSPOT PROBABILITIES FOR A REQUEST RATE OF 0.5. ........................................... 115

3.3 COMPARISON OF DIFFERENT INs FOR DIFFERENT HOTSPOT PROBABILITIES FOR A REQUEST RATE OF 1.0. ........................................... 116

4.1 PERCENTAGE ERROR BETWEEN RESULTS OF MODEL 3 AND SIMULATION RESULTS FOR N = 8. .............................................................. 211
LIST OF SYMBOLS

\( A \) : Binary number of \( \text{'m'} \) bits

\( \textit{AMBW} \) : Average memory bandwidth

\( a \) : Number of inputs of a switch

\( b \) : Number of outputs of a switch

\( C(A) \) : Cube of \( A \)

\( D \) : Binary number of \( \text{'n'} \) bits

\( d \) : Average delay for any packet in the network

\( d_i \) : \( i^{th} \) bit of destination address

\( d_j \) : Average delay for a packet of type \( j \), \( 0 \leq j \leq n \)

\( E_i \) : Exchange of \( D \)

\( h \) : Binary address of the hot memory module

\( k \) : Stage number

\( M \) : Number of memory modules

\( \textit{MM}_j \) : \( j^{th} \) memory module

\( \textit{MM}_h \) : Hot memory module

\( N \) : Number of processors

\( n \) : Number of stages in the network

\( PE_i \) : \( i^{th} \) processing element

\( P_i \) : set of data rates at the output of stage \( i \)

\( P_{b,j} \) : Blocking probability for a memory module of type \( j \)

\( P_h \) : Probability that a request is for the hot module, given
that a request is generated

\( P_{\text{w}}(k) \) : probability of a request accessing a hot output link of a switch at stage \( k \)

\( P_{\text{n}}(k) \) : probability of a request accessing a non-hot output link of a switch at stage \( k \)

\( P_A \) : Average probability of acceptance

\( P_b \) : Average probability of blocking

\( p_{i,i'} \) : Data rate on any input link of stage \( i \), \( 0 \leq i \leq (n-1) \), \( 0 < i' \leq i \)

\( p_{j,j'} \) : Data rate on any output link of stage \( j \), \( 0 \leq j \leq (n-1) \), \( 0 < j' \leq n \)

For Chapter 4

\( p_s(k,t) \) : Probability that a buffer of a switching element at stage \( k \) being in state \( x \) at the beginning of \( t^{\text{th}} \) cycle \( x \in \{0,1\} \)

\( p_{s,h}(k,t) \) : Probability that a buffer at the input of a hot switch in stage \( k \) is in state \( x \) at the beginning of \( t^{\text{th}} \) cycle \( x \in \{0,1\} \)

\( p_{s,n,h}(k,t) \) : Probability that a buffer at the input of a non-hot switch, which receives input from non-hot switches, is in state \( x \) at the beginning of \( t^{\text{th}} \) cycle \( x \in \{0,1\} \)

\( p_{s,n,n}(k,t) \) : Probability that a buffer at the input of a non-hot switch, which receives input from non-hot switches, is in state \( x \) at the beginning of \( t^{\text{th}} \) cycle \( x \in \{0,1\} \)

For model 2 and model 3

xvi
\( p_x(t,k) \) : Probability that an input buffer in stage \( k \) is in state \( x \) at the beginning of phase 1 of \( t^{th} \) cycle,
\( x \in \{0,n,h\} \) for model 2 \( x \in \{0,n,hu,hl\} \) for model 3

\( \tilde{p}_x(t,k) \) : Probability that an input buffer in stage \( k \) is in state \( x \) at the end of phase 1 of \( t^{th} \) cycle, \( x \in \{0,n,h\} \) for model 2 \( x \in \{0,n,hu,hl\} \) for model 3

\( p^*_x(t,k) \) : Probability that a buffer at stage \( k \) is able to accept a packet during phase 2 of \( t^{th} \) cycle, given that the buffer is in state \( x \) at the start of phase 1,
\( x \in \{0,n,h\} \) for model 2 \( x \in \{0,n,hu,hl\} \) for model 3

\( p_x(t,k) \) : Probability that a buffer at stage \( k \) is able to accept a packet during phase 2 of \( t^{th} \) cycle, \( x \in \{0,n,h\} \) for model 2 \( x \in \{0,n,hu,hl\} \) for model 3

For Chapter 3

\( q \) : Probability that a request is for the hot module given that a request is generated

\( \hat{q} \) : Probability that a request is for the non-hot module given that a request is generated

For Chapter 4

For Model 1

\( q(k,t) \) : Probability that a packet is ready to come to the buffer
of a SE at the beginning of $t^{th}$ stage cycle

$q_{HH}(k,t)$ : Probability that a packet is offered to a hot switch in stage $k$

$q_{Hh}(k,t)$ : Probability that a packet is offered to a switch of type 2 in stage $k$

$q_{hH}(k,t)$ : Probability that a packet is offered to a switch of type 1 in stage $k$

For Model 2 and Model 3

$q(t,k)$ : Probability that a packet is offered to a buffer of a SE at stage $k$ during phase 2 of the $t^{th}$ cycle

For Model 1

$r(k,t)$ : Probability that a packet in a buffer of a SE at stage $k$ is able to move forward during the $t^{th}$ stage cycle, given that there is a packet in that buffer

$r_{Hh}^A(k,t)$ : Probability that given a packet in a buffer at the input of a hot switch in stage $k$, the packet is able to win the contention and move forward to the hot output link

$r_{hH}^A(k,t)$ : Probability that given a packet in a buffer at the input of a hot switch in stage $k$, the packet is able to win the contention and move forward to the non-hot output link

$r_{hH}(k,t)$ : Probability that given a packet in a buffer at the input of a non-hot switch which is receiving input from a non-
hot switch, the packet is able to win the contention and pass on to a specific output link.

\( r_{HN}(k,t) \) : Probability that given a packet in a buffer at the input of a non-hot switch which is receiving input from a hot switch, the packet is able to win the contention and pass on to a specific output link.

For Model 2

\( R_x^i(t,k) \) : Probability that a packet in a buffer of a SE at stage \( k \) is unable to move forward (to either output links) during phase 1 of \( i^{th} \) cycle, given that the buffer is in state \( x \) at the beginning of phase 1 of \( i^{th} \) cycle, \( x \in \{n,b\} \)

\( r_x^i(t,k) \) : Probability that a packet in a buffer of a SE at stage \( k \) is able to move forward during phase 1 of \( i^{th} \) cycle, given that the buffer is in state \( x \) at the beginning of phase 1 of \( i^{th} \) cycle, \( x \in \{n,b\} \)

For model 3

\( r_{nx}^i(t,k) \) : Probability that a packet in an input buffer at stage \( k \) is able to move forward during phase 1 of the \( i^{th} \) cycle to the upper output link, given that the buffer was in state \( x \) at the beginning of phase 1 of the \( i^{th} \) cycle, \( x \in \{0,n,hu,hl\} \)

xix
\( r_d^1(t,k) \) : Probability that a packet in an input buffer at stage \( k \) is able to move forward during phase 1 of the \( t^{th} \) cycle to the lower output link, given that the buffer was in state \( x \) at the beginning of phase 1 of the \( t^{th} \) cycle, \( x \in \{0,n,bu,bl\} \)

\( r_u^1(t,k) \) : Probability that a packet in an input buffer at stage \( k \) is unable to move forward during phase 1 of the \( t^{th} \) cycle to the upper output link, given that the buffer was in state \( x \) at the beginning of phase 1 of the \( t^{th} \) cycle, \( x \in \{0,n,bu,bl\} \)

\( r_d^0(t,k) \) : Probability that a packet in an input buffer at stage \( k \) is unable to move forward during phase 1 of the \( t^{th} \) cycle to the lower output link, given that the buffer was in state \( x \) at the beginning of phase 1 of the \( t^{th} \) cycle, \( x \in \{0,n,bu,bl\} \)

\( S_i \) : Stage \( i \), \( 0 \leq i \leq (n-1) \)

\( S(A) \) : Shuffle of \( A \)

\( T \) : Binary number

\( t \) : Clock cycle number
Greek Symbols

For Chapter 4

For Model 1

\( \rho_{u,t} \) : The bandwidth on a single link i.e., the average number of packets received on an output link in one stage cycle

\( \rho_{u,h}^k(k,t) \) : Probability that a message is received on the hot output link of a hot switch at stage \( k \)

\( \rho_{h,n}^k(k,t) \) : Probability that a message is received on the non-hot output link of a hot switch at stage \( k \)

\( \rho_{n,h}(k,t) \) : Probability that a message is received on an output link of a non-hot switch which receives input from a non-hot switch

\( \rho_{n,n}(k,t) \) : Probability that a message is received on an output link of a non-hot switch which receives input from a hot switch

For model 2

\( \rho(t,k) \) : Probability that a packet is received on a specific input link of a SE at stage \( k \), during phase 2 of the \( t^{th} \) cycle

For model 3

\( \rho_s(t,k) \) : Probability that a packet is received on the upper input
link of a SE at stage $k$ during phase 2 of the $t^{th}$ cycle

$\rho_k(t,k)$ : Probability that a packet is received on the lower input link of a SE at stage $k$ during phase 2 of the $t^{th}$ cycle

NOTE: In model 2 and model 3, the different probabilities stated above are attached a prefix (example h for a hot link) to identify the different types of probabilities that exist for a single probability for non-uniform references. Sufficient information is given in the text as and when it is required.
THESIS ABSTRACT

NAME OF STUDENT: MOHAMMAD SHAHED AKHTAR
TITLE OF STUDY: PERFORMANCE OF MULTISTAGE INTERCONNECTION NETWORKS UNDER NON-UNIFORM REFERENCE MODEL
MAJOR FIELD: ELECTRICAL ENGINEERING
DATE OF DEGREE: JANUARY, 1992

Shared memory multiprocessors are becoming increasingly valuable in the field of parallel processing. The performance of the interconnection network used for communication has a direct bearing on the efficiency of the system. Sharing of data among hundreds of processors may create bottlenecks in the interconnection network. The bottlenecks that arise due to concurrent accesses to a shared memory are called hotspots. Hotspots result in a congestion of traffic on particular paths in the network which leads to a substantial degradation in the performance of the network. This thesis deals with the performance evaluation of the unbuffered and buffered multistage interconnection networks under the presence of a single hotspot. A new analytical model is proposed for analyzing the network under the effect of this non-uniformity. Extensive simulations are performed to demonstrate the accuracy of the proposed models. The performance evaluation of both unbuffered and buffered omega networks is done.

MASTER OF SCIENCE
KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
Dhahran, Saudi Arabia
خليفة الرسالة

عنوان الرسالة: أداء شبكات الربط متحدة للمرحل تحت ظروف غير منتظمة.

اسم الطالب: محمد شهيد أحمد
التخصص: الهندسة كهربائية
تاريخ الدرجة: 1433 هـ - يناير 1434 هـ

إن متعدد العوامل ذات الذاكرة المشتركة قد ازداد أهميتها خصوصاً في مجال التاغلايا المتوازية. إن أداء شبكات الربط تؤثر بشكل مباشر على كفاءة النظام المستخدم. ومن المنطاق أن تحقق العوامل على مئات العوامل ربط واحد فيما يؤدي إلى اختلافات وإشارات في شبكات الربط وتمى هذه الإشارات أو الاختلافات نقاط ساخنة (HOT SPOTS). وتظهر هذه النقاط الساخنة نتيجة لزيادة بعض قنوات الإتصال في الشبكة والتي تؤدي بدورها إلى انخفاض كبير في مستوى الأداء.

هذا البحث هو دراسة لتقييم أداء شبكات الربط متحدة للمرحل.
وقد استخدم منها نوعان هما: الاحتياطي الذاكر وغير الاحتياطي الذاكر وكلاهما تم دراسته مع وجود نقطة ساخنة واحدة. بالإضافة إلى ذلك فقد اقترح نموذج جديد لتحليل الشبكة في ظروف غير منتظمة. وتم محاكاة هذا النموذج ودراسته لعدها مدى دقته. وفي نفس الوقت تم تقييم أداء كلا نوعي شبكات أوميجا (OMEGA) أو أمانة الذكر.

خريطة الماجستير في العلوم
جامعة الملك فهد للبترول والمعادن
الظهران - المملكة العربية السعودية
رجب 1412 هـ - يناير 1992 م
xxiv
CHAPTER I
INTRODUCTION

Parallel processing is the hub around which the wheel of modern computing process revolves. An interconnection network (IN) is utilized by a parallel processing system to facilitate concurrent operation. Actually an IN is the heart of a parallel architecture such as a multiprocessor system.

This chapter of introduction is organized as follows: First a brief introduction to parallel processing is given. In the next subsection, the need for INs is outlined along with the classification of INs. Then the basic terminologies that are to be used for performance evaluation are defined. A description is given about the assumptions that are usually made in the literature while analyzing INs. In this the uniform reference model (URM) is also defined. A brief introduction is given to the non-uniform reference model, which is followed by a detailed account of literature on interconnection networks. Then the motivation behind the work is summarized. At the end the proposed problem is paraphrased.
1.1 Overview of Parallel Processing

Parallel processing computer systems provide a cost-effective means to achieve a high system performance through concurrent activities. Formally speaking, parallel processing can be defined as an efficient form of information processing which emphasizes the exploitation of concurrent events in the computing processes [1].

Parallel processing can be applied at the system hardware level or at the algorithmic and programming level. The rapid advent and ascent of VLSI technology has created a new boundary in implementing parallel algorithms directly in hardware. Research in this area received impetus because of the observation of significant exploitable parallelism inherent in many scientific and engineering applications.

Some of the fields in which parallel processing has been applied, and in which it has become an indispensable tool are: image processing, intelligent robotics, expert computer systems, remote sensing applications, computational aerodynamics, etc.

The general structure of a parallel processing system and its parts are illustrated in Figure 1.1.
Figure 1.1: General Structure of a Parallel Processing System
1.2 Interconnection Networks

State-of-the-art computer systems involve thousands of processors / memories communicating with one another. For the advantages of parallel / multi processing to be not negated by inefficient communication, such systems require an efficient interconnection network for communication. An IN, which thus forms a critical segment of such systems, can be formally defined as a complex connection of switches and links that permits data communication between processors and memories or processors [2]. A general diagram illustrating the function of INs is shown in Figure 1.2. As a result of this significance of theirs, INs have been the object of research for quite a long time. Extensive investigations have taken place to evaluate their cost and performance. Much emphasis has been laid on cost effective networks for multiprocessor systems.

To understand the classification and performance of INs, it is necessary to understand the different parallel processing environments in which they work. The following section gives the classification of general parallel and distributed systems.
Figure 1.2: FUNCTION OF AN INTERCONNECTION NETWORK
1.2.1 Classification of parallel / distributed systems

Based on the level at which interactions between the processors occur, parallel / distributed systems can be broadly classified as multiprocessors and multicomputers. A brief description of their operations is listed below [3]:

Multiprocessors: A multiprocessor permits all processors to directly share the main memory. All processors have access to a common set of memory modules. A processor itself may or may not have a local memory. A multiprocessor can be further classified into tightly coupled multiprocessors and loosely coupled multiprocessors. In tightly coupled systems, the main memory is situated at the center, though each processor might have some local memory / cache. In a loosely coupled system, the main memory is partitioned and attached to the processors, although the processors share the same memory address space.

Multicomputers: In this type of systems, a processor has its own memory space; thus a processor cannot directly access another processor's local memory. Sharing between processors occurs at a higher level through a complete file etc. The interaction between processors relies on message passing between the source and destination processors. These type of systems have a high communication overhead.
1.2.2 Classification of Interconnection networks

[4,5]

1.2.2.1 Classification based on Network Topology

Based on network topology, INs can be classified as static INs and dynamic INs.

Static INs: In this topology, each switching element is connected to a processor. One dimensional topologies include linear arrays. Two dimensional topologies include ring, star, mesh etc. The chordal ring, 3-cube etc., constitute the three dimensional static INs. The $w$-wide hypercube of $d$ dimensions is also a static IN.

Dynamic INs: These are generally of four types — single-stage, multistage, crossbar and multiple-bus. Single-stage INs are composed of a stage of switching elements (SEs) cascaded to a link connection pattern. Single-stage shuffle/exchange network is a network of such type based on a perfect-shuffle operation. Multistage INs consists of more than one stage of SEs and are generally capable of connecting an arbitrary input terminal to an arbitrary output terminal. Examples of this type of network include baseline network, banyan network, delta network etc. In crossbar type of INs, each input port can be connected to any output port. Number of buses is reduced in a multiple-bus system where the number of buses is $\text{min}(N,M)$, for a system containing $N$ processors and $M$ memories.
1.2.2.2 Classification based on operation mode

Based on the operation mode, the INs can be classified as either synchronous or asynchronous INs. Synchronous INs are those in which there is a global clock that broadcasts clock signals to all devices in a system thus synchronizing all the operations to itself. Asynchronous INs, on the other hand operate by means of handshaking, without any global clock. As a result of this, asynchronous INs have easy expandability and modularity.

1.2.2.3 Classification based on switching methodology

Based on the type of switching involved, INs can be either circuit-switched or packet-switched. In circuit switching, a direct physical path is established between the source and the destination. An initial time delay is always there to establish the path. Generally this scheme is preferable when entire bulk of data is to be transmitted. In packet switching, data is put in a packet, which also contains the destination address and routed through the IN without actually establishing a direct physical path. This type of transmission is most efficient for short data messages. A hybrid or integrated type of switching employs both the types mentioned above.

1.2.2.4 Classification based on control strategy

This concerns the way control signals direct the data flow in a network. Two types of control are possible -- centralized and decentralized. As the name suggests, in a centrally controlled IN, all the control signals come from a single
source. Examples of this type of multiprocessors INs include crossbar. In decentralized control, a small controller is associated with each component of the system. Examples of these type of multiprocessor INs include the multistage interconnection networks (MSINs). Multiple-bus INs employ either of the two schemes.

1.3 Basic Performance measures

The parameters in which the performance of any IN can be critically evaluated are outlined below.

Memory Bandwidth: One of the most common and significant measure for performance evaluation of any multiprocessor interconnection network is its memory bandwidth. It is defined as the mean number of active memory modules in a cycle of IN. Active memory modules, during a cycle, are those memories which are successfully accessed by a set of processors during this cycle. Among the INs mentioned so far, the crossbar has the capability to provide a bandwidth of $N$ for a $N \times N$ system.

A variation of this parameter is the throughput of the system. This is defined as the number of packets delivered by the network in unit time. Thus the bandwidth and the throughput are related by

$$\text{bandwidth} = \text{throughput} \times \text{network cycle time}. \quad (1.1)$$
Probability of Acceptance: This is defined as the probability of a processor's request being accepted during a cycle. It can be expressed as the ratio of the expected bandwidth of the system to the number of requests generated per cycle. The blocking probability of a system can be obtained by subtracting the probability of acceptance from one.

Processor Utilization: This is defined as the expected value of the percentage of time a processor is active. Overall processing power is the sum of all the processor utilizations of all the processors.

Network Delay: This is specified as the mean delay experienced by a packet in getting routed to the concerned memory module. It can also be defined as the reciprocal of the probability of having a packet routed to the network output.

1.4 Complexity of IN's

The amount of hardware needed to construct a crossbar network varies as $O(N^2)$ for a $N \times N$ system where as for a MSIN it is $O(N \log_2 N)$ since we have $\left(\frac{N}{2}\right)$ switches in each stage and there are a total of $\log_2 N$ stages. This decrease in hardware requirements is one of the fundamental causes for the popularity of the MSIN. For large $N$ the cost factor thus plays an important role in the selection of the network needed. The other type of IN i.e., the multiple-bus offers hardware complexity that is close to that of a MSIN. For a $N \times M \times B$ system, the complexity of connections is $O(B(N + M))$. Depending on the type of applica-
tion and environment governing the system a MSIN or a multiple-bus IN may be chosen in preference to a crossbar.

1.5 Assumptions in Literature

A brief pointwise description of the assumptions that are usually made while analyzing any IN are given below:

1. The interconnection network connects $N$ processors to $M$ memory modules. Every processor is assumed to be identical to every other processor.

2. The request rate is assumed to be one i.e., at the start of each cycle each processor submits a new request. This is an idealistic assumption though in reality, not every processor generates a request in every cycle. This assumption is usually made to make the analysis simpler.

3. The operation is assumed to be synchronous i.e., the requests are generated only at the beginning of clock cycle.

4. Requests generated by a processor have an equal and independent probability of being directed to any memory module. This model is generally referred to as the uniform reference model, (URM). This is again an idealistic assumption. In practice, the requests generated are not uniformly distributed. The uniform reference model gives the upper limit of the performance of the IN.

5. Fair routing logic is assumed i.e., in case of a conflict at any switch, the switch randomly selects only one packet, the other being rejected / discarded in an unbuffered network. In buffered networks, the requests are queued up according to any general policy.
6. Temporal independence of requests is assumed. This implies that the request generated by a processor in a cycle is independent of whether the requests in the previous cycle were rejected or accepted. This assumption leads to optimistic results as in reality there is a correlation between the requests of two consecutive cycles.

7. Spatial independence of requests is also assumed. This implies that the requests generated by a processor are independent of those generated by another processor in a cycle.

1.6 Non-Uniform References

As mentioned before, the uniform memory reference model is highly optimistic and away from reality. In practice, the traffic in the network is concentrated in some parts and not-so-concentrated in the others. This leads to traffic congestion in some parts of the network. This type of modelling which takes into account the non-uniform references is called as non-uniform reference modelling. This thesis deals with the performance of MSIN under non-uniform reference model. Of the several types of non-uniformities, the 'hotspot' phenomena, favorite memory, localized reference etc have been analyzed by different researchers.

The phenomena of 'hotspots' was first observed by Pfister and Norton [6]. Variables used for locking, global and barrier synchronization, pointers to shared queues etc., are indivisible primitives which should be stored in a single
shared memory. These primitives will be accessed by all processors, giving rise to an increased request rate for the corresponding memory module. Such memory modules are called hot memory modules and the concept is called as hot spot contention. The potential degradation in performance as reported in [6] is very high. A detailed description of the causes and effects of hotspots is given in chapter 3.

The concept of favorite memory was outlined and analyzed by Bhuyan [7]. He presented the fact that each processor can have a different favorite memory which it accesses more frequently. This again leads to performance degradation. Localized references were analyzed by Sethi & Deo [8].

1.7 Literature Survey

Various interconnection topologies such as crossbar, multistage interconnection network (MSIN), multiple bus interconnection network, ring, mesh, hypercube etc., have been studied in detail.

1.7.1 Literature for crossbar

Crossbar network, which is a fully connected network, is ideal as regards the performance (A fully connected network is one in which every processor can access every memory module). Hence this is taken as the standard to evaluate the performance of the other networks.
1.7.1.1 Study under URM

Len et al [9], analyzed the performance of an $N \times M$ crossbar. They assumed a synchronous operation for both buffered and unbuffered networks. Valero et al [10], were among the very first to highlight the significance of potential bandwidth in crossbars by evaluating the performance of multiple bus interconnection network and explaining the degradation in their performance as compared to that of crossbar. A simple and forthright approach based on probabilistic techniques to determine the memory bandwidth was presented by Bhuyan et al., in [2]. Bhandarkar [11] analyzed the interference in synchronous circuit-switched crossbar systems using discrete markov chains.

Significant results

*Lang, Valero & Alegre* [10] This paper presented the results for a $N \times N$ crossbar with synchronous operation. Simulation results were presented without arriving at the analytical expression. Cycle time of all the memory modules was assumed to be constant.

*Yen, Patel & Davidson* [9] Synchronous $N$ -processor systems with $M$ shared memories are considered here. Blocked requests were resubmitted in the next cycle. Using the steady-state flow approach, the following equation was derived:

$$BW = M \left[ (1 - (1 - \frac{w}{M})^N) \left( 1 - (1 - \frac{1 - (1 - \frac{1 - L}{M})^N}{M}) \right) \right]$$  \hspace{1cm} (1.2)
Here $BW$ represents the bandwidth of a $N \times M$ system with an initial request rate of $\psi$ and $f$ is a performance degradation factor.

1.7.1.2 Study under localized/non-uniform references

Bhuyan [7] analyzed the performance in the presence of a 'favorite memory' — a memory module which is accessed more frequently by a processor. This scenario is more realistic than the URM. Using the work of Bhandarkar as the foothold, Sethi and Deo analyzed the interference for a general $N \times M$ system under the conditions of localized memory accesses [8]. They also presented simulation results for comparison.

Significant results

Bhuyan [7] In this paper, a synchronous crossbar is analyzed. Each processor generates requests at random; requests not accepted are rejected. In this model a processor communicates more frequently with a particular memory module with a probability $m$. The expressions derived for bandwidth for an initial request rate $p_0$ are

$$BW = N \left[ 1 - (1 - p_0)M \left( 1 - p_0 \frac{1 - M}{N - 1} \right)^{N-1} \right], \quad N = M \quad (1.3)$$

$$BW = N \left[ 1 - (1 - p_0)M \left( 1 - p_0 \frac{1 - M}{N - 1} \right)^{N-1} \left( 1 - \frac{p_0}{N} \right)^{M-N} \right], \quad M \geq N \quad (1.4)$$
\[ BW = N - M(1 - p_0^M)(1 - p_0^\frac{1-M}{N-1})^{M-1} - (N - M)(1 - p_0^\frac{1-M}{N-1})^M, \]

\[ M \leq N \]  \hspace{1cm} (1.5)

The equations 1.4 and 1.5 reduce to equation 1.3 for \( N = M \).

### 1.7.2 Literature for multiple-bus interconnection networks

Multiple-bus architectures were proposed as an alternative to the crossbar networks. These utilize fewer buses. Their performance has been analyzed in great detail over the past.

#### 1.7.2.1 Study under the URM

Multiple-bus architectures were first analyzed in detail by Lang et al., in [5]. Using simulations, they evaluated the memory bandwidth. The concept of bandwidth availability and reliability in these architectures was studied by Bhuyan and Das [12]. This study also encompassed the effect of failures on the performance of the network. Mudge et al. [13], presented a survey on multiple-bus architectures emphasizing their importance by finding out the memory bandwidth. The design of hardware arbiters for resolving conflicts was also elicited.

In [6], Mudge et al., analyzed the multiple-bus interconnection network by developing a discrete stochastic model for the bandwidth. Expressions were developed for parameters such as bandwidth, processor utilization, probability
of acceptance of a request, expected waiting time etc. The mathematical expressions developed were validated against simulation results. A descriptive account on the performance of multiple-bus INs was presented by Yang & Bhuyan for different timing and switching methodologies [3]. Liu and Jou [15] developed analytical expressions for effective memory bandwidth and processor blocking probability. Expressions were developed for memory bandwidth in a multiprocessor system by using a Markov model in [16].

**Significant results**

**Liu & Jou** [15] This paper analyzed the multiple bus configuration when there is no queuing. The initial rate of requests is $PA$ and the system has $N$ processors, $M$ memories and $B$ buses. The effective memory bandwidth for such a system is given by

$$embw = M \left[ 1 - (1 - \frac{PA}{M})^N \right] - \sum_{k=0}^{\min(N,M)} \binom{N}{k} PA^k (1 - PA)^{N-k} \left( \frac{k! S(n,k)}{M^k} \right).$$

$$B \leq \min(N,M) \quad (1.6)$$

Using this expression the blocking probability was also found. The results were compared with those of simulations.

**Yang & Bhuyan** [3] For a synchronous circuit-switched system with centralized and decentralized control, analytical expressions were given. For a system having $N$ processors, $M$ memories and $B$ buses, the bandwidth is given by
\[ bw = M \left[ 1 - \left( \frac{1}{M} \right)^N \right] - \sum_{y=0}^{M} \binom{N}{y} p^y (1-p)^{N-y} \sum_{x=y+1}^{\infty} \frac{(x-B)x!S(y,x)(M)}{M^x}, \quad (1.7) \]

where

\[ t_y = \min(N,M) \]

\[ S(y,x) = \text{sterling number of second kind and} \]

\[ p = \text{initial request rate.} \]

A similar expression was also developed for decentrally controlled systems. In this case the bandwidth is given by

\[ bw = \sum_{y=0}^{M} \binom{N}{y} p^y (1-p)^{N-y} M \left[ 1 - \left( 1 - \frac{1}{M} \right)^L \right], \quad (1.8) \]

where

\[ L = \min(y,B) \]

\[ y = \text{number of requests generated per cycle.} \]

Results were validated by comparing them with simulation results.
1.7.2.2 Study under non-uniform / localized reference model

Das & Bhuyan [12] studied the performance under the presence of non-uniform reference pattern such as 'favorite memory'. Generalized Time Petri Nets (GTPNs) have been used to determine the exact performance estimates for memory and bus interference for URM and non-uniform reference patterns in [17]. Sethi et al [8] modelled the interference in a multiple bus system due to localized accesses.

Significant results

Sethi & Deo [8]: This paper deals with high order interleaving of memories in multiple bus systems and the interference arising therein due to localized references. Locality of references arising due to consecutive requests is analyzed using a Markov chain, for a \( P \times M \) system (\( P \) is the number of processors and \( M \) is the number of memories). The average number of busy memory modules was found equal to \( \frac{mp}{m + p - 1} \), independent of the probability with which the next request is made to the same module. Simulation results were presented for comparison. The results from analytical and simulations are comparable.
1.7.3 Literature for multistage interconnection networks (MSINs)

Multistage interconnection networks are the most cost-effective of all the architectures. Much of the study in this field has been concerned with the evaluation of different MSINs and ways and means to improve their performance. Different types of MSINs have been studied in detail. Examples of such networks are the delta network, the shuffle / exchange network, the banyan network, the cube network etc.

1.7.3.1 Literature on construction, design and operation

The "perfect shuffle" interconnection pattern was introduced by Stone [18]. Various algorithms such as the FFT, matrix transposition, sorting etc were presented to highlight the importance of the shuffle operation for parallel processing. Lawrie [19] introduced and discussed the design and operation of a particular class of shuffle / exchange networks called the omega network. The construction and operation of delta networks was explained by Patel [20]. Bhuyan etal., introduced the idea of "generalized interconnection networks" and outlined their permutation capabilities [21]. Kumar & Reddy introduced the "augmented shuffle / exchange networks" wherein switches within a stage can be connected to one another [22]. This prevents disconnection of the network in the event of failure of the switch input or output links. Instead of using crossbars, multiplexers are used as switches. Operation of interconnection networks using shuffles has been clearly explained in [23].
Siegel [24] performed a comparison of various architectures for SIMD computers. Studies on permutations possible in a shuffle / exchange network have been many. Huang [25], Huang & Tripathi [26] described in detail the various permutation sets and routing techniques. Universality of a single stage shuffle / exchange network with respect to permutations done was described in [27]. Verma & Raghavendra [28] expostulated the rearrangability of shuffle / exchange networks by partitioning of switching elements.

1.7.3.2 Study under URM

A general class of networks called the banyan networks, which encompass MSINs such as delta, shuffle / exchange, cube etc., have been investigated as regards to performance by Kruskal & Snir [29]. They analyzed both buffered and unbuffered networks. The performance of unbuffered delta networks was studied by Patel [20] who also presented simulation results. He compared the performance of delta networks with that of a crossbar. His results show that the MSIN is a viable cost-effective alternative to the crossbar. Analytical and simulation results for packet-switched delta networks were also presented by Dias and Jump in [30,31,32]. In [31], the various states of a crossbar switch under URM with single buffers at inputs were described. These were used to arrive at a transition matrix from which the memory bandwidth was evaluated. Simulation results were given for buffer sizes of more than one as the number of states for buffer capacity of more than one grow abnormally large. Analysis of asynchronous packet-switched buffered MSINs was presented in [32]. This research reported deadlock detection and recovery for such shuffle/exchange networks and also studied their performance.
Jenq [33] presented a recursive way based on Markov process to model a buffered switch with a capacity of one per input link. Later his results were extended by Yoon et al., to multibuffers and switches of arbitrary sizes [34]. Simulations were also performed for comparison. The analysis was extended for PM2I networks. Yoon et al., performed performance analysis and comparison of various MSINs such as delta network, PM2I network for both buffered and unbuffered case in [35]. Chen et al., presented the results for performance of delta networks for various queue sizes in [23]. The interference analysis of a shuffle / exchange network was done by Thanawastien using the concept of equivalence classes and state transition matrices to get parameters such as average memory bandwidth, system traffic, blocking probability etc [36].

Burke et al. [37], modelled the operation of a MSIN by a single stage interconnection network. They utilized the queue length as a criteria and furnished expressions for the same which were used to signify the effect of shorter path length and hence lesser delay. As a means of improving the performance of unbuffered shuffle / exchange networks, Kumar & Jump proposed the idea of combining multiple delta subnetworks to form a delta network and/or replacing each link by multiple links [38].

Bhuyan et al. [21], analyzed the performance of generalized interconnection networks while the performance of augmented shuffle networks was studied by Kumar et al., in [22]. A comparison of the performance of MSINs with the other architectures, such as the crossbar and the multiple-bus, was performed by Bhu-
yan etal., in [2]. Bisbee and Nelson developed the expressions for bandwidth under faulty conditions [39].

1.7.3.3 Study under non-uniform / local references

The URM is unnecessarily unrealistic. In real environments the references are non-uniform. As an initial attempt towards finding out the effect of non-uniform references, Bhuyan studied the performance of delta networks in the presence of a favorite memory and compared it with that of a crossbar [7].

Recently there has been a concentrated study in this field. Pfister & Norton were the first to introduce the concept of "Hotspots" and "Tree saturation" in MSINs [6]. Hotspots are a consequence of concentrated traffic to a shared memory module or a synchronization variable. Pfister & Norton [6] presented simulation results to highlight the degradation in the performance of MSINs due to hotspots which were confirmed by Lee [40]. Message combining was proposed as a way to counter tree saturation in [6]. G.Lee etal. [41], studied the effectiveness of various degrees of combining while Yew & Tzeng [42] proposed the method of software combining, with analytical and simulation results to support their proposal. Lang and Kurisaki also performed extensive simulations in their study of non-uniform traffic spots (NUTS) in MSINs [43].

While analyzing the performance of direct binary n-cube network for multiprocessors under URM, Seth etal., performed simulations to analyze the effect of hotspots [44]. G.Lee presented a performance bound for multistage combining
networks by developing expressions for network delay for combining and non-
combining networks [45]. Scott & Sohi [46] extended the concept of feedback to 
multiprocessors in order to control tree saturation. They performed simulations 
on omega network to imply the improvement in performance by control of tree 
saturation. Harrison [47] used Markov chains to derive the probability density 
functions for transmission times on hot and non-hot paths of a network. Analysis 
of delta networks when each processor issues a sequence of requests to suc-
cessive memory modules i.e., the consecutive request traffic model, was done in 
[48] using simulations.

1.8 The Problem

Parallel/Concurrent processing is the order of the day. Keeping in view the 
expansion of the technological frontiers, there is a continual need for time-
effective, cost-effective interconnection networks which are a critical part of any 
multi-processing system. The existing networks, which meet the above require-
ments, need to be evaluated in an environment closer to reality. The aim of this 
thesis work is to evaluate the performance of multistage interconnection network 
under non-uniform memory references. The assumption made in the analyses so 
far is that of uniform referencing which is hypothetical. This work besides pro-
viding a purview of the work done, aims to examine the degradation in the per-
formance of a MSIN under the presence of a single hotspot.

An omega network will be used to perform the analysis. The 'irregular' 
behaviour of the network under hotspots will be postulated. Extensive probabi-
listic techniques will be used to derive the expressions for basic performance measures such as memory bandwidth, blocking probability, probability of acceptance, and the time delay. The above measures will be evaluated for varying amount of load on the network and for various extent of hotspot probabilities. Analysis will be carried out for unbuffered and buffered omega networks. Exact analytical expressions will be attempted for a buffer capacity of one by modelling the states of a switching element using Markovian principles. Simulations will be performed for unbuffered and buffered networks having a capacity equal to or greater than one.

The motive of this work is to provide a comprehensive material to be used in the analysis of MSIN's. Hence, an interpretation of the results obtained will be presented.

1.9 Summary

This chapter gave an introduction to the concepts of parallel processing. A review of the different types of INs was given along with their classification (section 1.2). The basic performance measures to be used were presented in section 1.3. An introduction was given to the concept of non-uniform references. A detailed literature survey of the different types INs was given in section 1.6 (some of the significant results were also mentioned). The problem to be tackled in the thesis was also defined in section 1.8.

The next chapter starts with an introduction to the MSINs following which a description of the delta networks is given along with the principle of operation
of the same. Details of the shuffle interconnection pattern and the different routing and conflict resolution strategies are also given. A concise report is presented on the significant results reported for the unbuffered and buffered MSINs (under URM and non-URM), at the end of the chapter.
CHAPTER II
MULTISTAGE INTERCONNECTION NETWORKS

In the previous chapter, an introduction was given to the operation of a general parallel processing system and INs. A detailed classification was also presented for the same. Results reported in the literature for crossbar and multiple-bus INs were presented along with a brief literature review of the MSINs. In this chapter the MSINs are analyzed in detail.

The multistage interconnection networks (MSINs) were proposed as a cost-effective alternative to the crossbar. MSIN has established itself as a separate but vital class of IN. These MSINs allow a rich subset of one-to-one and simultaneous mappings of processors to memory modules.

This chapter starts with a brief introduction to the different types of MSINs. Then the design and operation of Delta networks is explained. In the same section the shuffle interconnection pattern is also described along with the different routing strategies. Conflicts and conflict resolution techniques are also given in section 2.2. Then a summary of the significant results given in the literature so far is given in section 2.3.
2.1 Introduction

Generally speaking, MSINs can be classified into three categories - blocking networks, rearrangeable networks and non-blocking networks [1].

In blocking networks simultaneous connections of more than one terminal pair may result in conflicts in the use of communication links. Examples of this type of networks are the banyan network, the delta network, the omega network etc. This thesis deals with the performance evaluation of these types of networks.

A network is called a rearrangeable network if it can perform all possible connections between inputs and outputs by rearranging its existing connections so that a communication path for a new input-output pair can always be established. Benes network is an example of this type of network.

A network which can handle all possible permutations without blocking is called as non-blocking network. A crossbar network without memory conflicts is a non-blocking network. Clos network is also an example of this type.

In the following section a general description of the delta network, which is of the blocking type, is given.

2.2 Delta Network

Delta networks were proposed as an inexpensive alternative to the crossbar by Patel [20]. Delta network, as defined by Patel, forms a subclass of a very broad class of networks called the banyan networks. A network is said to be banyan type if it has the capability to provide a unique path from every input to every output.
A delta network is an $a^n \times b^n$ switching network which connects, by some interconnection pattern, $a^n$ input ports (processors) to $b^n$ output ports (memories), where $a$ and $b$ are some arbitrary numbers not necessarily equal. A $a^n \times b^n$ delta network has $n$ stages of switching elements. An $8 \times 8$ delta network is shown in Figure 2.1. Each stage $i$ consists of $a^{n-i} b^{i-1}$ crossbar modules of size $(a \times b)$ as switching elements [20]. Square delta networks are those networks which are composed of $b \times b$ switches i.e., the switches have an equal number of input and output ports.

The omega network forms a subclass of square delta networks wherein $b = 2$. Omega networks can be defined as a network which connects $N$ processors to $N$ memory modules. The total number of stages in a $N \times N$ omega network is $n = \log_2 N$. Each stage contains $(\frac{N}{2})$ $2 \times 2$ crossbar switches connected as shown in Figure 2.2. The operation of the omega network is explained in the subsection describing the operation of a delta network.

In the following subsections (2.2.1, 2.2.2, 2.2.3 and 2.2.4) operation of the individual switching elements and the network as a whole is explained with respect to the interconnection pattern and the routing strategy.
Figure 2.1: DELTA NETWORK
Figure 2.2: OMEGA NETWORK
2.2.1 Shuffle interconnection

The shuffle interconnection pattern is one of the most widely used interconnection patterns for multiprocessor INs. The significance of the shuffle interconnection pattern for application in digital computers stems from the fact that, since it utilizes the binary digits of the destination address for operation, it is not difficult to implement. A perfect shuffle of a vector was defined by Stone [18] as viewing that vector as a card deck and shuffling them so that after a shuffle, the elements from the two halves of the vector alternate. This operation is illustrated in Figure 2.3.

Stone [18] formulated that the indices on the left in Figure 2.3, are mapped onto the indices on the right by a mathematical relation. This relation is given by

\[ P = \begin{cases} 
2i, & 0 \leq i \leq \frac{N}{2} - 1 \\
2i + 1 - N, & \frac{N}{2} \leq i \leq N - 1 
\end{cases} \quad (2.1) \]

This is the operation of a shuffle from its definition. For example: the index number 2 on the left side which is \((010)_2\), when multiplied by 2 as per the definition, \(( since 2 < (\frac{N}{2} - 1) which is 3 here), gives index 4 into which it is mapped as is evident from Figure 2.3.

Formally speaking, the shuffle operation can be defined in mathematical terms as follows:

Let a number \( A \) of \( m \) bits be represented in binary as
Figure 2.3: PERFECT SHUFFLE OPERATION
\[ A = (a_{m-1}a_{m-2}a_{m-3}\cdots a_1a_0)_2 \]

\[ A = a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \cdots + a_12 + a_0 \]  \hspace{1cm} (2.2)

If we multiply this by 2, we get

\[ A = a_{m-1}2^m + a_{m-2}2^{m-1} + \cdots + a_12^2 + a_02 \]  \hspace{1cm} (2.3)

If \( a_{m-1} \) equals zero, then \( A' = 2A \) and if \( a_{m-1} \) equals 1 then \( A' = 2A + 1 - 2^m \) which is just the definition of the shuffle operation as mentioned before.

Shuffle of \( A \) denoted by \( S(A) \) is defined as the number obtained by cyclically rotating to the left, the bits in the binary representation of \( A \). Since the shuffle is a permutation of finite number of objects, the cyclic shift rule implies that after \( m \) shuffles of \( A \), we return to the number \( A \) itself. An example of 8-bit shuffle is given below:

\( A = 214_{10} \) in decimal and in binary it is \((11010110)_2\). After one shuffle this number is \( S(A) = (10101101)_2 \). After \( m = 10 \) shuffles, we get \( A \) again.

2.2.2 Use of shuffle interconnection in INs

Stone [18] has demonstrated how the shuffle operation can be used to effect various algorithms such as the FFT, matrix multiplication etc. Apart from this, the shuffle interconnection pattern finds extensive application in INs. Many MSINs utilize this by shuffling the data paths (links) between the various stages.
An omega network utilizes the perfect shuffle operation which is also utilized by all delta networks. For a network containing \(N = 2^n\) processors connected to an equal number of memory modules via \(n\) stages, a \(n\)-shuffle on the binary indices of each processor is sufficient to establish a connection to any module based on some routing strategy. The routing strategy to be employed depends on the type of switches used. In general any \(a \times b\) crossbar switch should be able to connect any one of its \(a\) input links to any one of the \(b\) output links. Omega and all delta-2 networks utilize \(2 \times 2\) crossbar switches. The operation of a \(2 \times 2\) crossbar switch is shown in Figure 2.4.

Figure 2.4 shows the straight connection where the upper input link is connected to the upper output link and the lower input link is connected to the lower output link. It also shows the exchange operation where the connections are interchanged.

Using these connections all processor-memory interconnections can be formulated.
Figure 2.4: OPERATION OF A CROSSBAR SWITCH
2.2.3 Actual routing strategy

Since there is no predefined path which any request from any processor should follow, there should be some routing strategy which a request undertakes so as to reach its correct destination. The different types of routing strategies in vogue are: destination tag routing and exclusive-OR routing. These strategies apply to both circuit switching and packet switching. In circuit switching, the path once granted is held till the request is completed. While the processor is waiting service on a request, it is held in a wait state. In packet switching the packet itself contains some sort of tag based on which routing of the packet takes place.

**Destination tag control** In general we express the destination $D$ in base-$b$ system for a $a^n \times b^n$ delta network as

$$D = (d_{n-1}d_{n-2}\ldots \ldots d_d)_b \tag{2.4}$$

At each stage $i$ the routing of the packet to an output link is controlled by the bit $d_i$. Since in a base-$b$ system $d_i$ can have $b$ possible values, each input link is thus able to access all $b$ output links based on $d_i$. A detailed proof that this strategy results in all possible connections is given in Patel [20].

For a $2 \times 2$ crossbar switch the operation is as follows. If the destination bit is 0 then the data is routed to the upper output link. If the bit $d_i$ is 1 then the data is routed to the lower link. If the destination bits of packets at both the inputs are the same, then we have a conflict. The above mentioned operation of a $2 \times 2$ switch is shown in Figure 2.5.
$d_i = 1$

$\bar{d}_i = 0$

*Figure 2.5: DESTINATION TAG ROUTING*
In terms of the binary operation the Exchange function is defined as follows:

at stage $i$

$$D = d_{r1}d_{r2} \cdots \cdots d_r \cdots \cdots d_0 \quad (2.5)$$

$$E_i(D) = d_{r1}d_{r2} \cdots \cdots \overline{d_r} \cdots \cdots d_0$$

With the aforementioned definition of shuffle & exchange operations, an example is now given to illustrate the operation:

Say, processor number 2 in Figure 2.2 on page 31 issues a request for memory module number 6.

$$D = 6_2 = (110)_2 = d_2d_1d_0 \quad (2.6)$$

At the first stage $d_2 = 1$. Therefore the request is routed through to the lower link i.e., link no: 3. After shuffle link (3) is connected to S(011) i.e., 110. At port 6 the bit $d_1$ is examined. $d_1$ is 1 and the packet is routed to the lower link i.e., link 7. Now S(7) is 111 itself. Hence, at the third stage the packet is at the input link of SW15. Now $d_0$ is 0. Therefore the packet is routed to the upper link i.e., link 6 which is connected to the memory module 6. Thus memory 6 is accessed.

**Exclusive-OR method** Another form of routing strategy followed in MSINs is the exclusive-OR routing. An example of a MSIN utilizing this is the cube IN.
This method utilises the source address and the destination address to route the data. The cube routing function is defined as follows:

For any number $A$, represented as

$$A = a_{x1}a_{x2}\ldots\ldots a_i\ldots\ldots a_0$$

(2.7)

the cube of $A_i$, denoted as $C(A_i)$, is defined as

$$C(A_i) = a_{x1}a_{x2}\ldots\ldots \overline{a_i}\ldots\ldots a_0$$

(2.8)

Once a request is generated, the binary address of the destination is ex-ORed with the source generating the request to give an arbitrary routing tag denoted by $T$

$$T = S + D$$

(2.9)

where

$$S = s_{x1}s_{x2}\ldots\ldots s_i\ldots\ldots s_0$$

is the source address and

$$D = d_{x1}d_{x2}\ldots\ldots d_i\ldots\ldots d_0$$

is the destination address. At each stage the tag bit is checked. If $T_i = 1$, then at stage $i$, $C(T_i)$ is implemented. In the next section, the concept of conflict is described along with enumeration of the strategies used to resolve the conflicts.
2.2.4 Conflict

A conflict is said to occur when two requests at the two inputs of a switch require the same output link to go to a subsequent stage. Because of conflicts, the performance of a network is degraded as some of the packets are lost if they are not buffered; if they are buffered, there is a time delay before they reach their destinations. Conflicts that occur at intermediate switches are referred to as routing conflicts while conflicts occurring at a switch directly connected to a memory module are referred to as memory conflicts. An example of a routing conflict in an omega network is shown in Figure 2.6.

Here processor '0' generates a request for memory module number 4. Processor '4' generates for module number 7, giving rise to a routing conflict at the first stage. Since both inputs to switch '0' require the lower output link as \( d_2(4) = 1 \) and \( d_2(7) = 1 \) too.

If processor '0' generates a request for module 4 and processor '7' generates a request for module 4 also then it is a memory conflict. There is no routing conflict in this case as these two contend for the same link only at the final stage (SW10). This type of conflict, called the memory conflict, is shown in Figure 2.7.
Figure 2.6: ROUTING CONFLICT
Figure 2.7: MEMORY CONFLICT
2.2.4.1 Resolution of Conflicts:

Several strategies have been proposed to resolve the conflicts. The most common in use is the random resolution or the fair routing logic i.e.; given conflicting requests for an output link, probability that any one of them is selected is \( \left( \frac{1}{a} \right) \). (This type of logic maintains the uniformity in the distribution of requests). Another strategy is the prioritized strategy in which different links of a switch have different priorities i.e., in case of a conflict a particular link has higher priority of being selected than other links. In unbuffered networks, a request that loses the contention in case of a conflict is discarded. Buffered networks hold these requests in buffers (either at the input or the output switches).

2.3 Results reported

This subsection is organized as follows: the significant results and analysis done so far with regards to unbuffered & buffered MSIN's for a URM is given in the first part of this subsection. In the second part the analysis reported for unbuffered and buffered MSIN's for non-uniform reference model is discussed.
2.3.1 URM

2.3.1.1 Unbuffered

Patel [20]:

Unbuffered delta networks, with switches of arbitrary sizes were analyzed. Destination tag routing was assumed. Temporal independence was assumed. All processors were assumed to have submitted requests at the same time (synchronous). Because circuit switching was assumed, once a line is busy, the processor generating a request for the same was held in a wait state. Given the rate of memory requests $m$, at the beginning of a network cycle, the average memory bandwidth was computed. The expression for a $N \times M$ crossbar network with request rate $= m$ is given by

$$BW = N - N(1 - \frac{m}{N})^M$$  \hspace{1cm} (2.10)

Probability of acceptance $P_A$ was found from

$$P_A = \left(\frac{BW}{mM}\right)$$  \hspace{1cm} (2.11)

The expression for delta network $(a \times b)$ is given by

$$BW = b^a m_n$$  \hspace{1cm} (2.12)

where $n = \text{number of stages in the network}$
\[ m_0 = m \]
\[ m_x = 1 - \left( 1 - \frac{m_x - 1}{b} \right)^x \]

The probability of acceptance was found to be

\[ P_A = \frac{b^x m_x}{a^x m} \quad (2.13) \]

*Kruskal & Snir [29]*

Analysis is done for packet-switched banyan networks. The operation is assumed to be synchronous. Blocked packets are discarded. Asymptotic behavior of the probability of a packet on any particular input of any stage of a network is done. This asymptotic value \( p_m \) for stage \( m \) containing \( k \times k \) switches is found to be

\[ p_m = \frac{2k}{(k-1)m} \left[ 1 - \frac{(k+1)\ln(m)}{3(k-1)m} + O\left(\frac{1}{m}\right) \right] \quad (2.14) \]

The bandwidth is thus given by

\[ BW = \frac{4N}{\log_2 N} \left[ 1 - \frac{\ln(\log_2 N)}{\log_2 N} + O\left(\frac{1}{\log_2 N}\right) \right] \quad (2.15) \]

Approximate expressions were developed for \( p_m \).
2.3.1.2 Buffered

All the analysis done for the unbuffered networks suffers from a serious setback -- loss of a packet/request within a network. Queues (at either input/output/both) provide an efficient means to avoid this. The analysis for buffered systems is summarized in terms of significant results below:

*Dias & Jump* [30, 31]

Results based on analytical and simulation methods are presented for a packet switched delta network. The packets contain both the destination address and the actual data to be transmitted. Buffers at network output links are emptied instantaneously. All input packets are assumed to follow URM. The operation is modelled using Petrinets. The time interval is split in to two parts. \( t_{\text{select}} \) -- to select an output link and \( t_{\text{pass}} \) -- to pass the packet to the selected link. The performance measures extracted are throughput and turn-around-time. Initial states of a \( 2 \times 2 \) switch with a single buffer between stages are outlined. These states are based on a discrete time markov chain. State transmission probabilities are developed from which a transition matrix is formed. The matrices are multiplied iteratively to get the steady-state solution. One of the important conclusions was that as the number of buffers increases, throughput converges to a constant value. The rate of increase in the throughput with the number of buffers is largest for a change from 1 to 2 buffers. This rate of increase falls rapidly as buffer size increases.

*Kruskal & Snir* [29]
Study of banyan networks in packet switching environment is performed. Infinite buffers were assumed at each input port of a $k \times k$ switch. The queuing and dequeuing policy was FIFO. A formula for average number of queuing cycles for an initial probability of $p$ was derived as

$$b_\infty = \frac{(1 - \frac{1}{k})p}{2(1 - p)}$$  \hspace{1cm} (2.16)

The average transit time through the $\log_k N$ of a square banyan network was found to be

$$T = \log_k N \left( t_s + t_c \frac{(1 - \frac{1}{k})p}{2(1 - p)} \right)$$  \hspace{1cm} (2.17)

where

$t_s$ = transit time of a packet from one switch to another

$t_c$ = cycle time of a switch i.e.; the interval between successive arrivals is the same.

Simulation results are presented so as to validate these results.

\textit{Jenq} [33]

Using Markov chain approximations extensive analytical equations are presented for the performance analysis of a packet switch with input buffers of capacity one. Performance measures such as normalized throughput, normalized
delay and blocking probability are computed. The state of a buffer is modeled as either being empty or full. Two cases are analyzed; the independence and interdependence of the states of the two buffers of a switching element (SE). It is concluded that the independence assumption, which is motivated by the fact that both the packets arrive from disjoint sets of inputs, does not lead to significant differences from the model which is based on interdependence. Recursive equations are presented for the derivation of the throughput. The analysis is also extended to the case where an input buffer controller controls a network input buffer of finite and infinite length.

*Yoon et al* [34]

This work is an extension of the work done by Jenq to a general $a \times a$ switch based delta network. Simulation results are presented. The analysis is also applied to buffers with capacity more than one at inputs of all stages, using Markov chains. FIFO policy was used. Fair routing logic was assumed. Instead of network cycles a stage cycle is used to model the operation of a switch. As the independence of packets in different buffers is assumed, the analytical expressions are slightly optimistic. The results are extended to different types of networks such as PM2I network to indicate the fundamental generality of the model.

*Theimer & Huber* [49]

In this paper the performance of banyan network is studied by deriving a refined analytical model for a delta-2 network which is a subclass of the banyan network. The paper is motivated by Jenq’s model. Synchronous network is ana-
analyzed with no packets assumed to be lost once they enter the network. The analysis is based on Markov chains. It is outlined that the inaccuracy of Jenq & YLL models is due to the fact that a packet blocked in a cycle requests the same destination again in the next cycle, thus giving a correlation between the two cycles. A buffer capacity of one is assumed. An intermediate state is introduced to take into account the state of the buffer when it contains a blocked packet. State probabilities and state transition probabilities are computed recursively to give a steady state throughput. Results are highly accurate as compared to simulations.

Hsiao & Chen [50]

A modification to Jenq’s approach is done so as to model a single-buffered MSIN’s. As the Theimer model is based on exhaustive equations it is difficult to generalize for \( a \times a \) switches. This paper presents the approach for a \( a \times a \) switch. An intermediate state is again introduced to account for the blocked packet. Binary probability tree is used to derive transition probabilities. Results are compared to simulation results and are found to be accurate. The paper demonstrates the inaccuracy of YLL model at high loads.

Burke et-al [37]

This paper models the behavior of a MSIN by single stage IN. For a \( k \)-stage network, \( k \) recirculations through the SSIN give the same throughput. By reducing the time delay and the associated hardware, this SSIN is proclaimed as a better alternative. Performance is analyzed with respect to average path lengths, with and without buffers. A synchronous system is assumed with no
requests being accepted until the present request is satisfied. Partitioning the requests in terms of their path lengths is done to form a distance matrix. Simulation results are also presented.

2.3.2 Non-uniform reference model

Pfister & Norton [6] As reported earlier, Pfister & Norton were the first to discover the phenomena of hotspots. One of the major conclusions of their work was that, independent of the switching strategy adopted, there is a degradation in the performance of the network. In circuit switching networks, the time required to complete a circuit prior to initiating the transfer of data will be high in presence of hotspots. In packet switching networks, the concept of tree saturation occurs. In this work, the idea of combining is proposed as a means of prevention of the degradation. Simulation results are the plane for discussion.

Lee, Kruskal & Kuck [41] This paper studies the effectiveness of combining in presence of hotspots for a banyan network. A FIFO buffer was included at the output of each switch. Synchronous operation was assumed for packet-switched networks. The types of combining schemes studied include unrestricted combining, restricted combining and no combining. Network delay was used to classify them. In usual cases only $a$ messages are combined at an $a \times a$ switch. An increase from $a$ to $(a+1)$ improves the performance to almost equal to that of unrestricted combining. Simulation results were used for arriving at this conclusion.
Scott & Sohi [46] This paper proposes the use of feedback scheme to control tree saturation. As soon as the buffers in succeeding stage are blocked, this information is passed via a feedback link so as to stop further traffic from entering the network. Regular omega networks were used for simulation of different threshold length for the queues. The results show an improvement in bandwidth.

2.4 Summary

In this chapter, a description of the operation of the MSINs was given along with topological description of the delta networks. The shuffle interconnection pattern was explained and routing and conflict resolution strategies outlined. A survey of the results reported so far for unbuffered and buffered MSINs, under URM and non-URM, was also given.

The next chapter starts with an insight into the causes and effect of non-uniform references. Then a model is developed for the analysis of unbuffered omega networks with the actual analysis following it. The simulations performed are also explained. The results are presented as a separate section at the end of the chapter.
CHAPTER III

PERFORMANCE EVALUATION OF UNBUFFERED NETWORKS

In the previous chapter the operation of the MSINs was described in detail. This chapter gives the actual analysis of the unbuffered omega network. In section 3.1, the causes and effects of non-uniform memory references are explained. The model to be used for analysing an unbuffered omega network is developed in section 3.2 which also contains the properties of the network under a hotspot. In this section, the theorems developed are discussed. The derivations follow in section 3.3. Simulation methodology is explained in section 3.4 following which an interpretation of the results is presented. Then a comparison of the results obtained for different INs is presented.

3.1 Non-Uniform Memory References

In a realistic multi-processing environment, uniformity in accessing memory modules by a set of processors is found wanting. This analysis bases itself on taking into account the peculiarities resulting from non-uniform memory references. The types of non-uniformities identified and studied in literature are:
1. favorite memory, wherein a single memory module is accessed more frequently by a particular processor
2. localized references, wherein references in consecutive cycles tend to follow interleaving patterns
3. hotspots, when a particular memory is accessed more frequently by all processors

This thesis deals with the study and performance evaluation of a MSIN under a hotspot. This section presents a brief description of the phenomena of hotspots — its causes, effects etc.

3.1.1 Hotspots

As mentioned earlier, hotspots can formally be defined as, “memory locations which are accessed more frequently than other locations due to them containing indivisible primitives such as variables used for locking, global synchronizations etc.” A system can have more than one hotspot. The degradation in performance due to a single hotspot is drastic, and offsets all the advantages of large systems. This degradation in performance in unbuffered networks is due to increased traffic on a set of links. This has the effect of reducing the probability of other requests going through. Several types of possible hotspots are:

Module Hotspot: Here it means those non-uniform requests that merge into one memory module but not necessarily on the same location.
Distributed Hotspot: This type of hotspot occurs in distributed type of architectures where the memory is the local memory of each processor. Task scheduling and process execution can result in distributed hotspots.

The effect of hotspots is global in nature i.e., the total traffic is affected by it. Of the many effects of hotspots is the reduction in the number of memory modules accessed in a cycle or the memory bandwidth. The effect of hotspots is independent of the network type/topology, switching mode (packet or circuit). In circuit switching networks, the effect of hotspots is prior to the data transfer. Due to increased accesses to a particular module which results in increased blocking, the time required to complete the circuit is increased.

As outlined in the preceding section the effect of hotspots on the system performance is too significant to be overlooked. In an unbuffered network also there is an appreciable reduction in bandwidth. An accurate analysis of such networks under this non-uniformity is presented in this chapter.

3.2 Model

From the name, unbuffered network implies that in case of conflict at any switching element, the request that loses the conflict is discarded and lost in the network. This is true for SE's of all the stages of the network. An unbuffered Omega network is shown in Figure 2.2 on page 31. The assumptions and conditions under which the analysis is made are as follows:

1. An Omega network is taken as a representative MSIN for analysis. This does not lead to a loss of generality as the Omega network forms a sub-
class of Delta networks which are a sub-class of Banyan networks. The network is assumed to contain \( N = 2^n \) processors and \( N = 2^n \) memories where \( n \) is an integer. The \( i^{th} \) processor and the \( j^{th} \) memory module are denoted by \( PE_i \) and \( MM_j \) respectively, \( 0 \leq i, j \leq N-1 \). A stage \( i \) is denoted as \( S_i \) where \( 0 \leq i \leq N-1 \). The stages are numbered from the input side to the output side.

2. Each stage contains \( \left( \frac{N}{2} \right) \) SEs of size 2 X 2. This assumption is also a standard one. The analysis done is easily extendable to switches of arbitrary size.

3. Synchronous operation is assumed

4. Packet switching is assumed for routing of messages. (A packet contains both data and destination address.)

5. Fair routing arbitration logic is assumed. In case of conflict at any switch, the switch randomly selects one input and the other packet is discarded.

6. Request generated by a processor in a cycle is independent of whether requests at previous cycle were accepted or rejected i.e., temporal independence is assumed.

7. Request generated by a processor is assumed to be independent of requests generated by any other processor. This is defined as spatial independence.

8. At the beginning of each cycle a processor generates a random request with a probability \( p_0 \). In other words \( p_0 \) is the average number of
requests generated per cycle by each processor. The total number of requests entering a $N \times N$ omega network are therefore $N p_0$.

9. A single hot spot is assumed i.e., the memory request pattern is non-uniform. The hot spot, by its definition, is a hot memory module for all processors. This memory module denoted by $MM_a$ is accessed more frequently. If a processor $PE_i$ generates a requests, the probability that it will request $MM_a$ is denoted by $q$ whereas the probability that it will request any other memory module $MM_j, j \neq h$, is $\hat{q} = \frac{1-q}{N-1}$. The above discussion implies that $q > \hat{q}$. When $q = \hat{q}$, we have the uniform reference model or URM described in chapter 1.

10. At any switch the packets that arrive at input links are not necessarily uniformly distributed over the output links of the switch. This is the fundamental cause for propagation of blocking as the congestion on the link that is accessed more frequently increases.

11. Destination tag routing is assumed. At any stage the destination bit $d_i$ determines whether a packet is routed to the upper link ($d_i = 0$) or lower link ($d_i = 1$).

Temporal dependence between requests has been found to yield approximately the same results as without it, [20]. Hence this assumption does not render the analysis inaccurate. The assumption that each processor generates a request with equal probability, is made to reduce the complexity of analysis as
can be seen from the following description. With the above mentioned model the analysis is done. In the following section, some properties and behavior of the network in the presence of a hotspot is presented along with a detailed exposition of lemmas and theorems. Based on these properties, the exact method for the performance evaluation is presented section 3.3.

3.2.1 Properties

In the case of URM, the probability that a request is present on any output link of a switch is the same for all the switches. This is because of equal chance of packet being routed to either of the output links. In the case of non-uniform references, as this assumption no longer holds, the data rates at all the links between any two stages, are no longer equal. In Figure 3.1, the memory module \( MM_k \) is chosen as a hot module for all processors \( PE_{\alpha, i} = 0 - 7 \). As is evident by the thick lines indicating the path backwards from the hot module, these lines will have a increased data rate as compared to the other links in the corresponding stages. This follows from the fact \( MM_k \) is accessed more frequently than any other module. In order that any analysis is done these data rates need to be derived. For the derivation of these data rates some definitions and properties of the network are used. These are elucidated below:

**Hot request** This is a request for the hot memory module \( MM_k \). In Figure 3.1, all requests for \( MM_k \) are hot requests.

**Hot links** The requests from the processors reach their destinations via links. The links leading to the hot memory module are called hot links.
Figure 3.1: OMEGA NETWORK UNDER THE PRESENCE OF A HOT MODULE
These links, as explained before, have increased data rates. The hot links are shown by thick lines in Figure 3.1 on page 59.

**Hot switch** If any switch has at least one hot link connected to it either at input or output side, it is called a hot switch. Hot switches are shown in Figure 3.1 on page 59 by thick boxes. If $MM_4$ is a hot memory module, then as shown in the Figure 3.1 on page 59, switch numbers 0, 1, 2, 3, 5, 7, and 10 are hot switches.

Based on the topology of the network and the operation described before, the following theorems are presented:

**Theorem 1** At any stage the hot links at the output are either the upper output or the lower output of that stage.

Proof: If the address of the hot memory module $MM_A$ is $h$ where $h = h_0h_1...h_i...h_n$, then all the switches at stage $i$ will route the hot requests to either the upper output link or the lower output link for $h_i = 0$ or $h_i = 1$ respectively. Hence at any stage the hot links at the output are either the upper output or the lower output.

**Theorem 2** Both output links of a switch are never hot -- one of them is hot and the other is non-hot or both of them are non-hot.

Proof: As a single hot spot has been assumed, and at stage $i$ a selection of one output is made based on the bit $h_i$ of the hot module address, only one output can carry the hot request forward even if both the inputs contain hot requests.
This is evident from Figure 3.1 on page 59. Also evident is the fact that a switch can have both outputs as non-hot, because by definition hot traffic is only carried by hot switches and non-hot switches do not carry hot traffic. Hence, both outputs of a non-hot switch are non-hot.

**Theorem 3** Both the inputs to a switch at stage $S_{1,1}$ are either upper outputs or lower outputs of different switches at stage $S_r$.

**Proof**: This theorem follows from the definition of the shuffle interconnection pattern that is utilized by the Omega network to connect different stages. This fact can also be verified from Figure 2.2 on page 31.

**Theorem 4** The hot switches and hot links form a bipartite hot tree rooted at the hot memory module. The processors form the leaves of this tree while the hot switches and hot links form the vertices and the edges respectively. There are $2^n$ paths of this tree — the paths correspond to the paths traversed by the hot requests from the $2^n$ processors.

**Proof**: From the definition and the construction of the Omega network it is obvious that there is a unique path from each processor to every memory module. Stated in a slightly different manner this implies that, to every memory, there are unique paths from all processors. As a result of this there is a tree formed in the network. This is shown in the Figure 3.1 on page 59. As is evident from the figure, the links between the stages form the edges of the tree. Since there are $2^n$ processors in the network there are $2^n$ different trees to each memory module including the hot memory module $MM_k$. 
**Theorem 5** The data rates at the outputs of a hot switch are different.

Proof: The construction of the Omega network is such that the two output links of any switch carry requests to disjoint set of memories because if they were not for disjoint sets both requests can reach the destination through different paths. Since the Omega network has a unique path from each input port to each output port this is not true. Hence the output links carry requests for two disjoint sets of memories. From theorem 2, at a hot switch one and only one output link is hot. The other output link carries data for non-hot modules. The data rate on the hot link will be high as it carries all requests for the memory set containing the hot module.

**Theorem 6** The data rates at the outputs of a non-hot switch are the same.

Proof: The two output links of a non-hot switch carry traffic for two disjoint sets of memories which have the same probabilities of being requested, because of the assumption that all non-hot memory modules are equally likely to be requested. As an example, the upper and lower output links of a switch 0 in Figure 3.1 on page 59 carry traffic for the memory sets \( \{MM_6, MM_1, MM_2, MM_3\} \) and \( \{MM_4, MM_5, MM_6, MM_7\} \) respectively. If \( MM_4 \) is the hot module, all the traffic directed to the lower link will have a higher data rate than the upper link at switch 0.

**Theorem 7** Both inputs to a switch will have the same data rate
Proof: Since both the inputs to any switch carry traffic which originates from equal number of identical processors and directed towards the same set of memory modules, the data sets at both the inputs are same.

**Lemma 1** Both inputs to a switch are either hot links or non-hot links.

Proof: The concept of hot tree is the basis for this statement. All hot switches in the network are part of a hot tree. Hence both the inputs will be hot as the hot tree originates from the hot module and diverges to different processors. All switches which are not a part of a hot tree are non-hot switches and will thus have both inputs as non-hot.

**Lemma 2** The number of hot switches reduce by half at succeeding stages and the number of hot links also reduce by half at the outputs of the succeeding stage. The number of hot links at the output of $S_i$ is \( \frac{2^n}{2^{i+1}} \) and the number of hot switches at $S_i$ is $2^{n-1-i}$.

Proof: From theorem 4 it follows that the hot switches and the hot links form a bipartite tree and since any switch can have only one hot output link, the number of switches and hot links reduce by half as the root of the tree is approached.

**Lemma 3** If there is a packet at any input to a switch, the probability that it will be routed to output $y_i$ is \( \frac{r_i}{\sum r_i} \) where $r_i$ is the sum of the probabilities with which a processor requests the set of memories reachable from the output $y_i$. 

Proof: From the theorems mentioned above, since the two output links carry requests to two disjoint sets of memories, the ratio of the sum of the probabilities with which a processor requests sets of memories reachable from \( y_i \) to the sum of probabilities with which a request at that switch can be routed to memories of any set gives us the probability of a packet being routed to \( y_r \).

3.3 Performance Analysis

The basic performance measures that are to be evaluated have been outlined in chapter 1. These items give a wholesome interpretation of the performance of the network. The average memory bandwidth, defined as the expected number of memory modules active in any memory cycle, is given by the expression

\[
AMBW = \sum_{k=1}^{N} k q(k)
\]  

(3.1)

where \( q(k) \) is the probability of \( k \) memory modules being active. Speaking in terms of data rates, the average memory bandwidth can also be defined as

\[
AMBW = \sum_{i=0}^{N-1} p(i)
\]  

(3.2)

where \( p(i) \) is the data rate at the output link of the \( i^{th} \) memory module. In order to calculate the bandwidth, it is necessary to evaluate these rates. As mentioned earlier, these data rates will be different for different links and hence the data rate on each type of link has to be separately calculated. For the calculation of
these data rates, we utilize the properties of the network under the presence of a hot spot described in section 3. As known parameters, we have the non-uniformity pattern i.e., a single hot spot. Also the initial load to the network (or the rate at which the processors request memory modules is known). Based on these parameters a recursive method is presented for calculating AMBW.

Since each switch is assumed to be a 2 X 2 crossbar, the data rates at the outputs of the switch can be found if the input data rate is known along with the probability of routing to the outputs. The recursive method described below takes into account these routing probabilities.

In Figure 3.1 on page 59 requests are assumed to be fed to stage $S_0$ at a rate denoted by $p_{1,0}$. In other words, there are a total of $Np_{1,0}$ requests generated in a cycle. It is assumed that this request rate is constant over different cycles. It has been mentioned before that the requests at the inputs of a switch will be routed by destination tag control. Therefore at stage $S_0$ requests for output memory module $MM_i$ will be routed to upper or lower output link of a SE depending on whether $i \leq \frac{N}{2} - 1$ or $i \geq \frac{N}{2}$ respectively. This is evident from the topology of the Omega networks. If $MM_h$ is a hot module then, the upper output links of stage $S_0$ will be hot links and lower ones will be non-hot if $h \leq \frac{N}{2} - 1$ (Theorem 6). Similarly if $h \geq \frac{N}{2}$, then the lower output links of stage $S_0$ will be hot links and the upper ones will be non-hot. At the input stage $S_0$, where all the input links are connected to 2$^n$ processors, the input links to all switches at $S_0$ are hot.
By lemma 2, $\frac{N}{2}$ of the output links at stage $S_0$ will be hot while $\frac{N}{2}$ will be non-hot. These $\frac{N}{2}$ hot links at the input of $S_1$ will result in $\frac{N}{4}$ links which carry hot data. This will result in a total of $\frac{N}{4}$ links at the input of stage $S_2$ to be hot. Let the data rates at the output of stage $S_0$ be defined as $p_{a,0}$ and $p_{a,1}$ for the hot and the non-hot output links respectively.

As both the inputs to a switch are either hot or non-hot, in the SE's at stage $S_1$ there are $\frac{N}{4}$ non-hot input links with data rates $p_{a,1}$ and there are $\frac{N}{4}$ hot input links with data rate $p_{a,0}$. At any stage $S_n$, the data rate at the output of a hot switch is denoted by:

$p_{a,0}$: data rate on hot output link

$p_{a,1}$: data rate on non-hot output link

According to theorem 2, at stage $S_1$ there will be $\frac{N}{4}$ hot output links and $\frac{N}{4}$ non-hot output links having data rates $p_{1,0}$ and $p_{1,1}$ respectively. Each of the $\frac{N}{4}$ non-hot switches will have an input data rate of $p_{a,1}$ since the data rates at the input of a non-hot switch are same. Also according to theorem 6, both the outputs of these switches will have the same data rate say $p_{1,2}$. Note that $p_{1,2}$ depends only on $p_{a,1}$ whereas $p_{1,0}$ and $p_{1,1}$ depend on $p_{a,0}$ and the distribution of the requests to the two outputs of a hot switch.
From the foregoing discussion, we observe that the output links of $S_0$ and $S_1$ have two and three different data rates respectively. Let the set of data rates at the output links of stage $i$ be denoted by $P_i$. For example, 

$P_1 = \{p_{1,0}, p_{1,1}, p_{1,2}\}$. Thus at the output of $S_1$ we have:

$\frac{N}{2}$ non-hot links with data rates $p_{1,x}$. These are produced by non-hot switches having non-hot input links with data rates $p_{a,1}$.

$\frac{N}{4}$ non-hot links with data rates $p_{1,1}$. These are produced by hot switches having hot input links with data rates $p_{a,0}$.

$\frac{N}{4}$ hot links with data rates $p_{1,0}$. These are produced by hot switches having hot input links with data rates $p_{a,0}$.

As an example, for an $8 \times 8$ Omega network the different sets of data rates at the different stage outputs are:

$P_0 = \{p_{a,0}, p_{a,1}\}$

$P_1 = \{p_{1,0}, p_{1,1}, p_{1,2}\}$

$P_2 = \{p_{2,0}, p_{2,1}, p_{2,2}, p_{2,3}\}$

The above mentioned concepts about the data rates are illustrated in Figure 3.1 on page 59.
Following the above mentioned procedure, one can recursively calculate the data rates at the output links of all the stages for any network size. In general we note that the size of the set of data rates at successive stages is related by the following relation

\[ |P_{r+1}| = |P_r| + 1, \quad |P_0| = 2 \]  

(3.3)

where \( |P_r| \) denotes the number of elements of \( P_r \). This equation is explained as follows. At each stage there is at least one hot switch. This hot switch generates two different output data rates. This generation leads to an increase in the number of data rates as soon as a hot switch is encountered.

At the output of stage \( S_r \), let us denote as \( p_{r,0} \) and \( p_{r,1} \) to be the data rates on the hot output link and non-hot output link of a hot switch. The input data rates of this switch is denoted by \( p_{r-1,0} \). Let the other data rates at the outputs of \( S_r \) be denoted as \( p_{r,2} \) \( p_{r,3} \) ............... \( p_{r,r-1} \) which are dependent on the data rates \( p_{r-1,1} \) \( p_{r-1,2} \) ............... \( p_{r-1,r-1} \), respectively at the non-hot input links of the non-hot switches. This input-output data rate dependency for the different stages can be expressed as a data rate dependency tree as shown in Figure 3.2. The figure shows which output data rates of a stage depend on which input data rates of that stage. The data rates in boxes are those at the hot links. The data rates in bold typeface are those which depend not only on the data rate indicated in the tree but also on the distribution of the requests (this aspect of dependence is explained below). The superscript associated with a data rate shows the number of links at that stage having that rate. For example, in an 8 X 8 Omega network at stage \( S_2 \)
depends on $p_{1,0}$

depends on $p_{1,0}$

depends on $p_{1,1}$

depends on $p_{1,2}$

and we have 1 link each with data rates $p_{2,0}$ and $p_{2,1}$, 2 links with data rate $p_{2,2}$ and 4 links with data rate $p_{2,3}$.

Based on the discussion presented above the following theorem is presented:

**Theorem 8** If $P_i = i+2$ i.e., at the output of $S_n$, there will be a total of $i+2$ different data rates from $p_{k,0}$ to $p_{k,i+1}$. The number of output links having non-hot data rate $p_{k,p}$, $0 \leq s \leq i+1$, is given by $\left( \frac{N}{2^{i+2}} \right)$. The number of links having hot data rate $p_{k,q}$, $0 \leq s \leq i+1$ is $\left( \frac{N}{2^{i+2}} \right)$.

The proof of this theorem follows from the data rate dependency tree. Added explanation to the proof is Figure 3.1 on page 59, which shows the data rates at different links of a omega network. As we move towards the root of the hot tree, the number of hot switches decrease. Also, the number of hot links decrease (since a hot output link by definition exists only at the output of a hot switch). Simultaneously the number of non-hot switches increases as the last stage is approached giving an increase in the number of non-hot links at the output of successive stages.
Figure 3.2: DATA RATE DEPENDENCY TREE FOR AN OMEGA NETWORK
Ipso facto, the following analysis is made to determine the different data rates at the links of the different stages. The data rates at the output links of \( S_0 \) and \( S_1 \) will first be determined which will be used to generalize the data rates at the output link of any stage. Consider a switch at stage \( S_0 \) shown in Figure 3.1 on page 59. The inputs of the switch are labelled as \( x_0 \) and \( x_1 \) while the outputs are labelled as \( y_0 \) and \( y_1 \). The input and the output data rates are also marked on the respective links. For the purpose of analysis select \( h \geq \frac{N}{2} \). This does not lead to any limitation on the analysis. The analysis holds for any \( h \). If \( h \geq \frac{N}{2} \), then for the network shown, one among the memory modules \( \{MM_a, MM_b, MM_c, MM_d\} \) is the hot module. From theorem 6, therefore, we have that at a switch of stage \( S_0 \) (shown in Figure 3.1 on page 59), the lower output viz., \( y_1 \) is the hot output link and hence it is labelled with \( p_{a,0} \). (If \( h < \frac{N}{2} \) then \( y_0 \) will be the hot output link and would be labelled with \( p_{a,0} \).)

The data rate at \( y_1 \) is equal to the probability that there will be a request routed at the \( y_1 \) output and is given by

\[
p_{a,0} = Pr[x_0 \rightarrow y_1] Pr[x_1 \rightarrow y_1] + Pr[x_0 \rightarrow y_1] Pr[x_1 \rightarrow y_1] + Pr[x_1 \rightarrow y_1] Pr[x_0 \rightarrow y_1]
\]

\[+ Pr[x_1 \rightarrow y_1] Pr[x_0 \rightarrow y_1] \tag{3.4}\]

where \( Pr[x_i \rightarrow y_j] \) is the probability that a memory request is routed to \( y_j \) from \( x_i \) and \( Pr[x_i \rightarrow y_j] \) denotes the probability that no request is routed to \( y_j \) from \( x_i \).
The above expression is explained as follows:

A request is routed to the output $y_1$ in three cases:

1. A request at $x_0$ wanting the $y_1$ output and a request at $x_1$ wanting the $y_1$ output, then at least one goes through.
2. When request at $x_0$ is routed to $y_1$ and at $x_1$ is not routed to $y_1$.
3. Request at $x_1$ is routed to $y_1$ and at $x_0$ is not routed to $y_1$.

The probability $Pr[x_1 \rightarrow y_1]$ is split as follows: probability that a request at an input is routed to an output is the probability that a request is present at the input multiplied by the probability that it is routed to the particular output i.e.,

$$Pr[x_1 \rightarrow y_1] = Pr[\text{that a message is present at } x_1] \cdot Pr[\text{message at } x_1 \text{ is routed } y_1]$$

In stage $S_0$ from the hot output of the hot switch, $(\frac{N}{2})$ memories can be reached. Of these $(\frac{N}{2})$ memories, $(\frac{N}{2} - 1)$ are requested with a probability $\hat{q}$ and the hot memory is requested with a probability $q$. From the non-hot output $(\frac{N}{2})$ non-hot memories can be reached, each of which is requested with an equal probability equal to $\hat{q}$. Therefore by lemma 3, provided that a message has arrived at $x_0$, the probability that it is routed to the non-hot output ($y_0$) is given by

$$P_{\text{non}} = \frac{\left(\frac{N}{2}\right) \hat{q}}{(N-1) \hat{q} + q} \quad (3.5)$$
and the probability that a message at \( x_0 \) is routed to the hot output \( (y_1) \) is given by

\[
P_{h0} = \frac{\left(\frac{N}{2} - 1\right) \hat{q} + q}{(N-1) \hat{q} + q}
\]

(3.6)

The probability that a message is present at \( x_0 \) is the load offered at a network input link i.e., \( p_{-1,0} \). Therefore

\[
P_{x0 \rightarrow y_0} = P_r[a \text{ message is present at } x_0] P_r[\text{Message at } x_0 \text{ is routed } y_0]
\]

\[
P_{x0 \rightarrow y_0} = [p_{-1,0}] \left[ \frac{\left(\frac{N}{2}\right) \hat{q}}{(N-1) \hat{q} + q} \right]
\]

(3.7)

and

\[
P_{x0 \rightarrow y_1} = P_r[a \text{ message is present at } x_0] P_r[\text{Message at } x_0 \text{ is routed } y_1]
\]

\[
P_{x0 \rightarrow y_1} = [p_{-1,0}] \left[ \frac{\left(\frac{N}{2} - 1\right) \hat{q} + q}{(N-1) \hat{q} + q} \right]
\]

(3.8)

Similarly we can derive the expressions for the probability that a request at \( x_1 \) is routed to \( y_0 \) and \( y_1 \) as:

\[
P_{x1 \rightarrow y_0} = [p_{-1,0}] \left[ \frac{\left(\frac{N}{2}\right) \hat{q}}{(N-1) \hat{q} + q} \right]
\]

(3.9)
\[ P[r_{i \rightarrow y_i}] = [p_{i,0} \left( \frac{\frac{N}{2} - 1}{N-1} \hat{q} + q \right)] \]  

(3.10)

Probability that a message at \( x_i \) is not routed to \( y_o \) denoted by \( P[r_{i \rightarrow y_o}] \) is defined as

\[ P[r_{i \rightarrow y_o}] = 1 - P[r_{i \rightarrow y_i}] \]

\[ P[r_{i \rightarrow y_o}] = 1 - [p_{i,0} \left( \frac{N}{2} \hat{q}}{(N-1) \hat{q} + q} \right)] \]  

(3.11)

Similarly, we can derive the expressions for other probabilities like \( P[r_{o \rightarrow y_i}] \), \( P[r_{o \rightarrow y_o}] \), and \( P[r_{i \rightarrow y_i}] \). Substituting the above expressions into the expression for \( p_{a,0} \) we get

\[ p_{a,0} = [p_{i,0} \left( \frac{\frac{N}{2} - 1}{N-1} \hat{q} + q \right)] [p_{i,0} \left( \frac{\frac{N}{2} - 1}{N-1} \hat{q} + q \right)] \]

\[ + [p_{i,0} \left( \frac{\frac{N}{2} - 1}{N-1} \hat{q} + q \right)] [1 - p_{i,0} \left( \frac{\frac{N}{2} - 1}{N-1} \hat{q} + q \right)] \]

\[ + [p_{i,0} \left( \frac{\frac{N}{2} - 1}{N-1} \hat{q} + q \right)] [1 - p_{i,0} \left( \frac{\frac{N}{2} - 1}{N-1} \hat{q} + q \right)] \]

\[ = 2 [p_{i,0} \left( \frac{\frac{N}{2} \hat{q} + q - \hat{q}}{(N-1) \hat{q} + q} \right] - [p_{i,0} \frac{\frac{N}{2} \hat{q} + q - \hat{q}}{(N-1) \hat{q} + q} \right]^2 \]  

(3.12)
Next we derive the expression for $p_{a,1}$ which is the probability that a message is routed to the output $y_0$ and is given by the expression

$$p_{a,1} = Pr[x_0 \to y_0] Pr[x_1 \to y_0] + Pr[x_0 \to y_0] Pr[x_1 \not\to y_0]$$

$$+ Pr[x_1 \to y_0] Pr[x_0 \not\to y_0]$$ \hspace{1cm} (3.13)

An explanation for this similar to the explanation for $p_{a,0}$. Substituting the various probabilities (from equations 3.7, 3.11 and 3.13), the expression for $p_{a,1}$ reduces to

$$p_{a,1} = 2[p_{1,0} \left( \frac{N}{2} \right) \hat{q} \left( \frac{N}{2} \right) \hat{q} + q] - [p_{1,0}]^2 \left( \frac{N}{2} \right) \hat{q} + q \right]$$ \hspace{1cm} (3.14)

Having derived the output data rates of $S_o$, we now proceed to derive the output data rates at $S_1$ in a similar fashion. For evaluating this the topology of the omega network is again the basis. From Figure 3.1 on page 59, we see that at stage $S_1$ an output link can access a set of $\left( \frac{N}{4} \right)$ memory modules only as opposed to the $\left( \frac{N}{2} \right)$ accessed by the output of $S_o$, irrespective of whether a switch is hot or non-hot. The set of $\left( \frac{N}{4} \right)$ memory modules accessed by a hot output link includes the hot memory module $MM_k$ and $\left( \frac{N}{4} - 1 \right)$ non-hot memory modules which are accessed equiprobably. The non-hot output link is
also capable of accessing \( \left( \frac{N}{4} \right) \) memory modules with equal probability. In the case of a non-hot switch, since both the outputs are non-hot, any one of them is capable of accessing a memory module from a set of \( \left( \frac{N}{4} \right) \) with equal probability.

The above mentioned factors influence the data rate calculation as is evidenced below from the expressions for the three different data rates at the output of \( S' \). These data rates are again calculated using the elementary probability techniques and the methodology used to obtain \( p_{a,0} \) and \( p_{a,1} \) earlier. Thus the data rates at the output of \( S' \) are:

\[
p_{i,0} = 2 [p_{a,0}] \left[ \frac{\left( \frac{N}{4} \right) \hat{q} + q - \hat{q}}{\left( \frac{N}{2} - 1 \right) \hat{q} + q} \right] - [p_{a,0}]^2 \left[ \frac{\left( \frac{N}{4} \right) \hat{q} + q - \hat{q}}{\left( \frac{N}{2} - 1 \right) \hat{q} + q} \right]^2 \tag{3.15}
\]

\[
p_{i,1} = 2 [p_{a,0}] \left[ \frac{\left( \frac{N}{4} \right) \hat{q}}{\left( \frac{N}{2} - 1 \right) \hat{q} + q} \right] - [p_{a,0}]^2 \left[ \frac{\left( \frac{N}{4} \right) \hat{q}}{\left( \frac{N}{2} - 1 \right) \hat{q} + q} \right]^2 \tag{3.16}
\]

\[
p_{i,2} = 2 [p_{a,1}] \left[ \frac{\left( \frac{N}{4} \right) \hat{q}}{\left( \frac{N}{2} \right) \hat{q}} \right] - [p_{a,1}]^2 \left[ \frac{\left( \frac{N}{4} \right) \hat{q}}{\left( \frac{N}{2} \right) \hat{q}} \right]^2
\]

\[= [p_{a,1}] - [p_{a,1}]^2 \left( \frac{1}{2} \right)^2 \tag{3.17}\]
Using the expressions for \( p_{i,0}, p_{i,1}, p_{i,2}, p_{i,3} \) and \( p_{i,2} \) derived earlier, the general expression for any data rate \( p_{i,j} \) are formulated as follows:

\[
p_{i,j} = p_{i-1,j-1} - [p_{i-1,j-1}]^2 \left( \frac{1}{2} \right)^2 \quad \text{for } 1 \leq i \leq n - 1, 2 \leq j \leq i + 1
\]

\[
p_{i,j} = 2[p_{i-1,0}] \left[ \frac{\left( \frac{N}{2^{i+1}} \right) \hat{q} + (q - \hat{q})(1 - j)}{\left( \frac{N}{2^i} - 1 \right) \hat{q} + q} \right]
\]

\[
- [p_{i-1,0}]^2 \left[ \frac{\left( \frac{N}{2^{i+1}} \right) \hat{q} + (q - \hat{q})(1 - j)}{\left( \frac{N}{2^i} - 1 \right) \hat{q} + q} \right]^2
\]

for \( 0 \leq i \leq n - 1, \ 0 \leq j \leq 1 \)

(3.19)

Using the above mentioned formula for \( p_{i,j} \), the data rate at any link of any stage can be calculated recursively. Note that equations 3.18 and 3.19 give the data rates at the output of non-hot and hot switches respectively. All the data rate sets \( P_i, i = 0, 1, \ldots, n - 1 \) can be formed. In particular, the set \( P_{n-1} \) determines the average memory bandwidth. The average memory bandwidth is the sum of all the data rates at the output of the network. In other words if \( P_{n-1} = \{p_{n-1,0}, p_{n-1,1}, p_{n-1,2}, \ldots, p_{n-1,n} \} \) then from the properties and theorems about the network we have that there is a single output link (the hot link) with data rate \( p_{n-1,0} \), there is one link with data rate \( p_{n-1,1} \), two links with data rate \( p_{n-1,2} \), four links with data rate \( p_{n-1,3} \) etc. Generalizing this, we have the expression for the average memory bandwidth as
\[ AMBW = p_{n-1,0} + \sum_{i=1}^{s} 2^{i-1} p_{n-1,i} \]  \hfill (3.20)

where \( 2^{i-1} \) is the number of output links of \( S_{n-1} \) having a data rate \( p_{n-1,i} \). The above mentioned simple analysis is extendable to any network size, any initial request rate or any rate of request for the hot module.

**Calculation of Blocking Probability**

Blocking probability, by the definition given in chapter 1, is the probability that a request is blocked. This probability depends on the initial load offered to the network. From the definition of \( q \) and \( p_{1,0} \), we have the total number of hot requests generated as \( N p_{1,0} q \) where \( N p_{1,0} \) is the total number of requests generated in any cycle. Of these \( N p_{1,0} q \) requests to the hot memory module, the number of requests reaching the destination is \( p_{n,0} \) (as there is no buffering in the network and hence blocked requests within the network are lost). Thus \( (N p_{1,0} q - p_{n,0}) \) requests will be blocked. Hence the probability that a hot request is blocked is given by

\[ P_{h,0} = \frac{\text{Number of hot requests blocked in a cycle}}{\text{Total number of hot requests generated in a cycle}} \]

\[ P_{h,0} = \frac{N p_{1,0} q - p_{n,0}}{N p_{1,0} q} \]  \hfill (3.21)

(For an 8 X 8 omega network, of the requests reaching the destination, we have three types. There is one request with data rate \( p_{2,1} \), two with \( p_{2,2} \) and four with \( p_{3,3} \). In general, the number of non-hot requests generated for a particular non-
hot module during a cycle is given by \( N p_{1,0} \hat{q} \). Therefore the total number of non-hot requests at the output are then given by \( \sum_{i=1}^{n} p_{x-i,1} 2^{i-1} \) Blocking probability for the module closest to the hot module (with a data rate of \( p_{x-1,1} \)) is given by

\[
P_{a,1} = \frac{N p_{1,0} \hat{q} - p_{x-1,1}}{N p_{1,0} \hat{q}}
\]  \hspace{1cm} (3.22)

Similarly the blocking probabilities for the links with the other data rates are

\[
P_{a,2} = \frac{N p_{1,0} \hat{q} - p_{x-1,2}}{N p_{1,0} \hat{q}}
\]  \hspace{1cm} (3.23)

\[
P_{a,x-1} = \frac{N p_{1,0} \hat{q} - p_{x-1,x-1}}{N p_{1,0} \hat{q}}
\]  \hspace{1cm} (3.24)

\[
P_{a,x} = \frac{N p_{1,0} \hat{q} - p_{x-1,x}}{N p_{1,0} \hat{q}}
\]  \hspace{1cm} (3.25)

Hence the average blocking probability for the network can be written as

\[
\frac{N p_{1,0} \hat{q} p_{a,0} + \left( \sum_{i=1}^{n} 2^{i-1} p_{a,i} \right) N p_{1,0} \hat{q}}{N p_{1,0}}
\]

(3.26)

The numerator represents the sum of number of hot requests blocked and the number of non-hot requests blocked. This gives the total number of requests
blocked in the network. The ratio of the total number of requests blocked to the total number of requests generated thus gives the overall blocking probability.

Probability of acceptance depends on the total number of requests accepted in the network which is given by AMBW in equation 3.20 Thus the total probability of acceptance is the ratio of AMBW to the total number of requests generated. Thus the overall probability of acceptance and the overall probability of blocking can also be found by

\[
P_A = \frac{\text{AMBW}}{N_{1,0}}
\]  \hspace{1cm} (3.27)

\[
P_B = 1 - P_A = 1 - \frac{\text{AMBW}}{N_{1,0}}
\]  \hspace{1cm} (3.28)

**Example for calculating bandwidth and blocking probability**

Let us take a network of size \(N = 8\). The probability of requesting a module uniformly is thus 0.125. Let the hot memory module be accessed with a probability \(q = 0.3\).

\[
\hat{q} = \frac{1 - q}{N - 1} = \frac{1 - 0.3}{8 - 1} = 0.1
\]

Let the initial request rate be \(p_{1,0} = 1.0\). Then all other data rates can be calculated as follows

\[
p_{0,0} = 2 \left[ 1.0 \right] \left[ \left( \frac{8}{2^0 + 1} \right) 0.1 + (0.3 - 0.1) (1 - 0) \right] \left( \frac{8}{2^0 - 1} \right) 0.1 + 0.3
\]
\[
- [1.0]^2 \left[ \frac{\left( \frac{8}{2^{0+1}} \right) 0.1 + (0.3 - 0.1) (1 - 0)}{\left( \frac{8}{2^1 - 1} \right) 0.1 + 0.3} \right]^2
\]

\[= 0.84\]

\[p_{0,1} = 2[1.0] \left[ \frac{\left( \frac{8}{2^{0+1}} \right) 0.1 + (0.3 - 0.1) (1 - 1)}{\left( \frac{8}{2^1 - 1} \right) 0.1 + 0.3} \right]
\]

\[= \left( \frac{8}{2^{0+1}} \right) 0.1 + (0.3 - 0.1) (1 - 1)
\]

\[= 0.64\]

Similarly we get the other data rates at the outputs of \(S_1\) as follows:

\[p_{1,0} = 0.8064\]

\[p_{1,1} = 0.4816\]

\[p_{1,2} = 0.5376\]

At the output of \(S_2\) the rates are as follows:

\[p_{2,0} = 0.8438\]

\[p_{2,1} = 0.36256\]
\[ p_{2,2} = 0.4236 \]

\[ p_{2,3} = 0.4653 \]

The average memory bandwidth (given by equation 3.20) is

\[ AMBW = p_{2,0} + p_{2,1} + 2p_{2,2} + 4p_{2,3} = 3.9148 \]

For the example (by equation 3.21)

\[ P_{h,0} = \frac{2.4 - 0.8438}{2.4} = 0.6468 \]

where 2.4 = 8 \times 1 \times 0.3 is the number of requests generated for the hot memory in a cycle and 0.8438 is the actual data rate at the hot output link. Similarly we have number of requests generated for a non-hot memory as \( N_{p,1,h} \) i.e., 0.8 here. Since the output data rates are varying, we have different blocking probabilities for different memories. By equation 3.22,

\[ P_{h,1} = \frac{0.8 - 0.36256}{0.8} = 0.5468 \]

\[ P_{h,2} = \frac{0.8 - 0.4236}{0.8} = 0.4705 \]

\[ P_{h,3} = \frac{0.8 - 0.4653}{0.8} = 0.418375 \]

Thus the overall blocking probability by equation 3.26, is
\[ P_0 = \frac{(2.4)(0.6468) + (0.8)[0.5468 + (2 \times 0.4705) + (4 \times 0.418375)]}{8 \times 1} \]

\[ = \frac{1.55232 + 2.52904}{8} = 0.51017 \]

This can also be obtained by using equation 3.28, where

\[ P_0 = 1 - \frac{3.9148}{8} = 0.51064 \]

Inferences:

As is evident from the above figures, for a 3-stage network, \( P_{2,0} > P_{2,3} > P_{2,2} > P_{2,1} \). This is as expected. This equation implies that the data rate at the hot output link is the highest while the data rate at the non-hot output link of the hot switch is the lowest. The data rates at the outputs of non-hot switches, which are in a set disjoint from the hot module, are the second highest while the switch which is in the same set as the hot module but is a non-hot switch has the second lowest data rate. In general, for an n-stage network, we have at the final stage

\[ P_{n-1,0} > P_{n-1,n} > P_{n-1,n-1} > \ldots > P_{n-1,2} > P_{n-1,1} \]  \hspace{1cm} (3.29)
3.4 Simulations

Explicit modelling of an N x N omega network was done in the simulations. Each time step corresponds to one network cycle (the time taken to transmit a packet from one stage to another). The actual switch operations are assumed to take zero time units. The simulated network operation with regards to an unbuffered network is summarized below:

1. At most one request is generated at the beginning of any clock cycle. The initial request rate was varied from 0.1 to 1.0 to provide a detailed account of the behaviour of the network.

2. All the requests were moved forward one stage in each cycle.

3. Once a request is blocked at any switch in the stage, including the input switches it was assumed to be lost.

4. The omega network is built using the construction outlined in sections 2.1 and 2.2. Separate routines are provided for effecting the shuffle operation and for detecting the bit to be tested at each stage.

The following input data values were varied each time to have a comprehensive picture of the network behaviour:

1. Length of the Simulation: The number of cycles for which the simulations were performed were large, typically 50,000.

2. Seed for the random number generator: The inbuilt random number generator provided in the 'C' language library was used (the 'C' lan-
guage supports a single random number generator). After a detailed survey, a seed was chosen for generating the random numbers. The accuracy of the seed was determined by taking out the mean of the numbers generated between '0' and '1'. (This was found to be 0.49953, the best of the lot that was surveyed). The simulation required two independent streams of numbers one for the generation of the request and the other for the resolution of the conflicts. The language supports only one generator and using the same stream for both could introduce dependency in the form of periodicity of the numbers. To take this into account, a large independent stream of random numbers were generated using another seed (which was found to be nearly as accurate as the other seed used for generating the numbers) and stored in a separate file. A number from this file was used to resolve the conflicts. The range of numbers is from 0-1 and there are two packets between which this range is divided — the upper and the lower packets contending for the same output link at any switch. If the generated number was between 0 and 0.5, the packet on the upper link was selected, otherwise the packet on the lower input link won the contention.

3. System Size: As the network simulated was an omega network, the number of processors equalled the number of memory modules.

4. Probability of packet generation at the beginning of each cycle.

5. Probability that a packet generated is destined for the hot module.
Having specified the above input parameters, the complete procedure used is mentioned below as an algorithm.

0- : declaration

0+ : initialization

1 : request generation

2 : calculation of different parameters

3 : iterating steps 1-3 over large number of cycles

4 : outputting the results

In order to facilitate analysis effective requests at the input of each stage were stored in a separate array, while the generated requests were stored in a separate array. The following explanation pertains to the steps mentioned above.

3.4.1 Request Generation

As mentioned earlier, the inbuilt random number generator in the 'C'-language library was used to have random requests at the beginning of each cycle. Implemented in a separate routine, this generation process was actually the allotment of generated requests according to the input parameters -- rate of request generation and the probability of accessing the hot module. The actual demarcation process is portrayed in Figure 3.3. Explanation of this process follows in the next paragraph.
Figure 3.3: DEMARCATION OF REQUESTS GENERATED BY A PSEUDO-RANDOM GENERATOR

RANGE OF RESULTS GENERATED

0.0 0.2 1.0

RANGE IN WHICH REQUESTS FOR HOT MODULE FALL

PORTION WHICH IS DELETED. THIS IMPLIES A REQUEST RATE OF 0.8

REMAINING PORTION WHICH IS UNIFORMLY DISTRIBUTED AMONG (N-1) NON-HOT MEMORIES
The random number generated is between 0 and 1. If the request rate is 1, then the whole range of 0-1 to represents the range of valid requests. To simulate a rate of request less than unity (say by a amount \( x \) where \( 0 \leq x \leq 1 \) ) the range of requests \( 0 - x \) were marked as the invalid requests and no memory module was assigned to this range. The remaining range \( x - 1 \) represents the range of valid requests with a request generation probability of \( (1-x) \). This range of requests was to be distributed -- for hot module and for the non-hot modules. The hotpot probability \( p_h \), given as the input, is the probability that a message requests the hot module, given that a request is generated. Hence if a request is not generated with a rate of 1, then the effective hotpot probability equals \( e = p_h (1-x) \). This method was used to allocate a request to the hot module. Thus if the generated number \( k \) is such that \( x \leq k \leq (x+e) \), then the request is for the hot module. If the request falls outside the above range, then it is for any one of the \( (N-1) \) non-hot modules. The distribution of requests over these \( (N-1) \) non-hot modules should be uniform. Hence \( \left( \frac{1-x-e}{N-1} \right) \) gives the probability of referencing a non-hot module. An example illustrating the above procedure is shown below:

- range of random numbers generated: 0.0 - 1.0
- range of valid requests: 0.2 - 1.0
- range in which the request to hot module falls: 0.2 - 0.6

(implies that the request rate is 0.8)
(implies that the hotspot probability = \( \frac{0.4}{0.8} = 0.5 \))

range in which request to non-hot modules fall \( 0.6 - 1.0 \)

(implies that the non-hot module request probability for a 8 x 8 network = \( \left( \frac{0.4}{0.8} \right) \frac{1}{7} = 0.0714 \))

3.4.2 Switching Operation

Destination tag routing was assumed (this is explained in sec 2.2.3). In case of a conflict, the conflict was resolved randomly (as explained earlier in this section) and the packet which lost the contention was discarded. A tag was associated with each valid packet at the end of the switching operation.

3.4.3 Parameters evaluated

The various parameters evaluated for unbuffered networks include the average memory bandwidth and blocking probability. At the output of the network, the number of valid packets were counted at the end of each cycle. These were averaged over the total number of cycles to give the average memory bandwidth. For the calculation of the blocking probability, the number of requests blocked in each cycle were calculated, summed and averaged over the total number of cycles to give the average blocking probability for a request generated.

The actual simulation program is listed in the appendix A. Comments are provided at appropriate places to aid in understanding the procedure.
3.5 Results and Discussion

In this section, a discussion on the performance of unbuffered delta networks is presented based on the results obtained in the previous section. To encompass all the performance peculiarities, the memory bandwidth and blocking probability were determined for parameters such as network size, hotspot probability and request rate.

The variation of memory bandwidth with the hotspot probability is of prime importance. This plot is shown in Figure 3.4 for various network sizes ranging from 4 - 256 for an initial request rate of 1. Plots for other initial request rates of 0.8, 0.6, 0.4 and 0.2 are shown in Figure 3.5, Figure 3.6, Figure 3.7, and Figure 3.8 respectively.

From Figure 3.4 we note that as the hotspot probability increases the bandwidth decreases for all network sizes. This is because of increasing number of requests being directed to a single memory module and of all these requests only one may be accepted. The non-hot requests have to contend with hot requests in the switches at different stages, reducing their probability of reaching the output. Also evident from the figure is the sharp decrease in the bandwidth for larger network sizes (above 4). For \( N \leq 64 \) the decrease is uniform as the hotspot probability increases but for \( N > 64 \), the rate of decrease rises with the network size. This also follows from the discussion presented above. For other request rates the pattern followed is similar. At lower requests rates the decrease in memory bandwidth follows a linear pattern.

The variation of memory bandwidth with the request rate is shown in Figure 3.9 for different network sizes with hotspot probability of 0.2. The variation
Figure 3.4: BANDWIDTH vs HOTSPOT PROBABILITY FOR DIFFERENT NETWORK SIZES, REQUEST RATE = 1.0
Figure 3.5: BANDWIDTH vs HOTSPOT PROBABILITY FOR DIFFERENT NETWORK SIZES, REQUEST RATE = 0.8
Figure 3.6: BANDWIDTH vs HOTSPOT PROBABILITY FOR DIFFERENT NETWORK SIZES, REQUEST RATE = 0.6
Figure 3.7: BANDWIDTH vs HOTSPOT PROBABILITY FOR DIFFERENT NETWORK SIZES, REQUEST RATE = 0.4
Figure 3.8: BANDWIDTH vs HOTSPOT PROBABILITY FOR DIFFERENT NETWORK SIZES, REQUEST RATE = 0.2
against the request rate is highly significant in the sense that not all real systems involve generation of ‘N’ requests in each cycle i.e., the initial request rate is less than 1. From Figure 3.9, it is evident that as the request rate rises the bandwidth increases. This is because of increased requests entering the network. The rate of increase is not uniform. It is higher at lower request rates and at higher request rates it is lower. The bandwidth is maximum for a request rate of 1.0 for all network sizes. Other hotspot probabilities also follow the same pattern as is evident from Figure 3.10, Figure 3.11, Figure 3.12, and Figure 3.13 for hotspot probabilities of 0.4, 0.5, 0.8, and 1.0 respectively.

The percentage error between the analytical and simulation results is shown in Table 3.1. This table shows the errors for a range of hotspot probabilities for various network sizes. As is evident, the errors are reasonably small.

When all the requests are directed towards the hot memory (a highly hypothetical case) the bandwidth degradation is the highest and all network sizes give the same performance for higher request rate.

For multiprocessing systems, there is always a chance of expanding the system or reducing it. In order to facilitate this, a study is made to evaluate the bandwidth against increase in the network size for various hotspot probabilities. One of these graphs is shown in Figure 3.14 for a initial request rate of 1.0. (Plots for requests rates of 0.8, 0.6, 0.4 and 0.2 are shown in Figure 3.15, Figure 3.16, Figure 3.17, and Figure 3.18 respectively).

Obviously the bandwidth increases with the network size for all hotspot probabilities except 1.0. The increase in bandwidth for all network sizes is higher for lower hotspot probabilities. For any network size, the URM represents the
Figure 3.9: BANDWIDTH vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.2
Figure 3.10: BANDWIDTH vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.4
Figure 3.11:
BANDWIDTH vs. REQUEST RATE FOR DIFFERENT
NETWORK SIZES, HOTSPOT PROBABILITY = 0.6
Figure 3.12: BANDWIDTH vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.8
Figure 3.13: BANDWIDTH vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 1.0
<table>
<thead>
<tr>
<th>( N )</th>
<th>( p_a = 0.2 )</th>
<th>( p_a = 0.4 )</th>
<th>( p_a = 0.6 )</th>
<th>( p_a = 0.8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.082</td>
<td>-0.0298</td>
<td>0.10696</td>
<td>0.1578</td>
</tr>
<tr>
<td>4</td>
<td>0.133</td>
<td>-0.4958</td>
<td>0.08955</td>
<td>0.01039</td>
</tr>
<tr>
<td>8</td>
<td>0.00049</td>
<td>0.21179</td>
<td>0.21902</td>
<td>0.21902</td>
</tr>
<tr>
<td>16</td>
<td>0.2841</td>
<td>0.07466</td>
<td>0.33078</td>
<td>0.07624</td>
</tr>
<tr>
<td>32</td>
<td>-0.01225</td>
<td>-0.05007</td>
<td>-0.32316</td>
<td>-0.13787</td>
</tr>
<tr>
<td>64</td>
<td>-0.23501</td>
<td>-0.04071</td>
<td>0.2860</td>
<td>0.16657</td>
</tr>
<tr>
<td>128</td>
<td>-0.20695</td>
<td>0.02320</td>
<td>0.23804</td>
<td>0.16657</td>
</tr>
<tr>
<td>256</td>
<td>-0.16347</td>
<td>0.01467</td>
<td>0.17856</td>
<td>0.12376</td>
</tr>
</tbody>
</table>

Table 3.1: Percentage errors between analytical and simulation results for different \( N \).

Asymptotic value. The increase in memory bandwidth follows a linear pattern.
Figure 3.14: BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES, REQUEST RATE = 1.0
Figure 3.15: BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES, REQUEST RATE = 0.8
Figure 3.16: BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES, REQUEST RATE = 0.6
Figure 3.17: BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES, REQUEST RATE = 0.4
Figure 3.18: BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES, REQUEST RATE = 0.2
The degradation in bandwidth is evident for larger hotspot probabilities for any network size. The trend of linear increase in bandwidth with network size is maintained for all requests rates.

Blocking probability is an important measure as regards to performance evaluation. This gives direct information as to the probability with which a generated request is blocked. Figure 3.19 shows the variation of average blocking probability against request rate for different network sizes with a hot spot probability of 0.1. An interesting result is that the blocking probability is highest for larger network sizes which means that increasing the network size does not result in an improvement in the overall system performance. At full load more than 65% of the requests generated are blocked for a 256 × 256 network. It is evident from Figure 3.20, Figure 3.21, Figure 3.22 and Figure 3.23 that as the hotspot probability increases, the blocking probability also increases.

From the above mentioned discussions it is evident that larger network sizes are not a good means of improving the system performance. Besides increasing the cost ($O(N \log_2 N)$), they result in an increase in blocking probability even for low hotspot probabilities. An optimum network size has to take all these factors into consideration.
Figure 3.19: BLOCKING PROBABILITY vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.1
Figure 3.20: BLOCKING PROBABILITY vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.3
Figure 3.21: BLOCKING PROBABILITY VS REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.5
Figure 3.22: BLOCKING PROBABILITY vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.7
Figure 3.23: BLOCKING PROBABILITY vs REQUEST RATE FOR DIFFERENT NETWORK SIZES, HOTSPOT PROBABILITY = 0.9
3.6 Comparison of different INs

For making a comparison with other architectures such as multiple-bus and crossbar, the results given in [52] are taken. The comparison is shown in Table 3.2 and Table 3.3. The results are for a $N=8$ and $M=8$ with $B=4$ and $B=8$ (crossbar).

The tables show that for higher request rates MSINs perform better under a hotspot while the percentage degradation in a crossbar is the highest. (This is because the maximum attainable bandwidth for a crossbar under a hotspot is $N$).
Table 3.2: COMPARISON OF DIFFERENT INs FOR DIFFERENT HOTSPOT PROBABILITIES FOR A REQUEST RATE OF 0.5.

<table>
<thead>
<tr>
<th>IN</th>
<th>$p_k = 0.2$</th>
<th>$p_k = 0.4$</th>
<th>$p_k = 0.6$</th>
<th>$p_k = 0.8$</th>
<th>$p_k = 1.0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-Bus</td>
<td>3.0685</td>
<td>2.8306</td>
<td>2.3706</td>
<td>1.7426</td>
<td>0.9900</td>
</tr>
<tr>
<td>MSIN</td>
<td>2.7925</td>
<td>2.5659</td>
<td>2.1584</td>
<td>1.6253</td>
<td>0.9900</td>
</tr>
<tr>
<td>Crossbar</td>
<td>3.1977</td>
<td>2.9015</td>
<td>2.3912</td>
<td>1.7443</td>
<td>0.9900</td>
</tr>
</tbody>
</table>
Table 3.3: COMPARISON OF DIFFERENT INs FOR DIFFERENT HOTSPOT PROBABILITIES FOR A REQUEST RATE OF 1.0.

<table>
<thead>
<tr>
<th>IN</th>
<th>$p_k = 0.2$</th>
<th>$p_k = 0.4$</th>
<th>$p_k = 0.6$</th>
<th>$p_k = 0.8$</th>
<th>$p_k = 1.0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-Bus</td>
<td>3.9727</td>
<td>2.8408</td>
<td>2.3828</td>
<td>1.4239</td>
<td>1.0000</td>
</tr>
<tr>
<td>MSIN</td>
<td>4.0870</td>
<td>3.6540</td>
<td>2.9570</td>
<td>2.0787</td>
<td>1.0000</td>
</tr>
<tr>
<td>Crossbar</td>
<td>5.1810</td>
<td>4.5654</td>
<td>3.6275</td>
<td>2.4488</td>
<td>1.0000</td>
</tr>
</tbody>
</table>
3.7 Summary

This chapter gave the detailed performance analysis of an unbuffered MSIN under the presence of hotspot. The properties and the behaviour of the network were outlined. Parameters such as memory bandwidth and blocking probability were evaluated. Results were compared with simulation results and found to be accurate.

The next chapter starts with an introduction to the concept of tree saturation which is the fundamental cause of performance degradation in buffered MSINs under non-uniform references. Then a detailed analysis of single-buffered MSINs is done by developing three models.
CHAPTER IV

PERFORMANCE OF SINGLE-BUFFERED NETWORKS

4.1 Introduction

In the previous chapter, a detailed analysis of the unbuffered MSINs was presented. The results indicate a substantial degradation in the performance of the network under a single hotspot. One of the drawbacks of the unbuffered type of networks is that a packet can be lost within the network; to avoid this buffering of blocked packets is done.

This chapter starts with an introduction to the causes and effects of non-uniform references in buffered MSINs. Then the properties of the networks having a buffer capacity of one along with the assumptions necessary are presented in section 4.3. Based on the assumptions, three models have been developed for analysis. The first model is described in section 4.4. First the model is developed for uniform references and then extension is done to non-uniform references in section 4.4.1. Simulation of buffered networks is described in section 4.4.2. Results and discussion is presented in section 4.4.3. The second model is described in section 4.5. Again, the model is first presented for uniform references and then the extension to non-uniform references is given in section 4.5.1. The results of this model are given in section 4.5.2. The model 3 is described in section 4.6. At the beginning the analysis for URM is given followed by the
extension to non-uniform references in section 4.6.1. Then a discussion of results obtained is given in section 4.6.2.

4.2 Non-Uniform Memory References:

The degradation in performance of buffered packet switched networks because of non-uniform referencing (hotspot) is due to the occurrence of a phenomenon called Tree saturation [6]. An explanation of this phenomena is given below:

4.2.1 Tree Saturation

Consider the omega network shown in figure 3.1. Assuming this is a packet switched network, the following description is valid. A tree is shown in the figure with its root at the shared memory module, \( MM_x \). As this is the hot module, traffic is directed towards it at an increased rate. In a particular cycle, only one request for the hot module can be satisfied with all other requests being queued up. If the switch that is connected to the module has all its requests to \( MM_x \), then only one is satisfied. The rest queue up. This forces the switches connected to the input of this switch to queue up their respective messages for the same link. After a while the full queue at this stage forces the first stage queues to fill up. The whole process now forces the requests to be blocked from entering the network because of the saturation of all the links leading up to the hot module. This effect will lead to a formation of a tree as shown in figure 3.1. This is called as tree saturation. Tree saturation cripples the network. An abnormal delay is caused to non-hot packets and the final bandwidth is reduced significantly. This delay is due to the fact that, even though these packets are for non-
hot modules, they have to pass through the tree which has a congestion of packets for hot module.

4.2.2 Methods to control Tree Saturation

Tree saturation is not uncontrollable, but the cost at which the prevention of tree saturation is done is a relevant factor. The different methods proposed to control tree saturation are:

Feedback [46] As the length of the queue is the sole affecting factor on tree saturation in buffered MSIN's, queue length control offers a viable means to prevent tree saturation. A queue length is fixed as a threshold. If at any time a queue at a memory module exceeds the desired length, a feedback mechanism forces the processors to stop issuing requests for the hot module. In this way the effect of blocked queues is prevented from propagating further back into the network. Once the threshold is reached, until the queue length decreases, the processors will be idle. This reduces the processor utilization. The improvement in bandwidth is substantial and more than offsets the other drawbacks.

Combining [6,41,42,45] This is the method originally proposed by Pfister and Norton [6] to alleviate the problem of tree saturation. Message combining implies the combining of messages at switches. The combinable messages are those which request identical memory modules. If combining has taken place at a switch then it is recorded in some form of a tag. When the handshaking signal from the memories to processors comes at the switch where combining took
place, the multiple requests are satisfied. Various studies have studied the effectiveness of various degrees of combining. Combining prevents tree saturation by preventing the occurrence of full queues. The additional hardware required for combining (like an extra buffer) can be eliminated by having software combining.

As outlined in the preceding chapter, the effect of hotspots on the system performance is too significant to be overlooked. An accurate analysis of buffered networks under this non-uniformity is presented in this chapter. In the next section the model to be used for analyzing the network is described.

4.3 Analysis of Buffered Networks

Under the uniform reference model, it has been observed that the buffering of the requests in the network increases the system bandwidth. In this chapter the effect of buffering on an omega network is analyzed under hotspots. In the literature various models have been presented to evaluate a buffered network. Based on these models, three models have been developed for the evaluation of the buffered networks under hotspots. In the section 4.3.2 of this section an introduction to the buffering action and the general model are outlined. In section 4.3.3, a brief introduction is given to Markov processes while the properties are presented in section 4.3.4.

Various buffers sizes have been studied in literature. Most of the work done models the buffering action by means of a buffer at either the input or the out-
put port of each switch [33,34,49] The capacity of each buffer in a network is equal but not necessarily one. The different types of buffering strategies used are First-in First-out (FIFO) buffers, Last-in First-out (LIFO) buffers. Mostly Markov chains have been used to derive the different states of the buffer and attempt is directed towards obtaining a steady state solution. Our analysis of buffered omega networks is based on the following general model. (The nuances of each of the three models developed along with associated properties are presented in separate sub-sections):

4.3.1 General Assumptions
1. The network to be analyzed is a $N \times N$ omega network. The stage connected to the processors is referred to as the first stage, denoted by 1 and the stage connected to the memory side is referred to as the output stage denoted by $n$.

2. A single hotspot is present. This means memory module $MM_j$, $0 \leq j \leq n - 1$ is accessed more frequently by all processors.

3. Network operation is assumed to be synchronous and packet switched.

4. The network contains buffers at the input links of all switches i.e., if the network size is $N \times N$ then we have $\left( \frac{N}{2} \log_2 N \right)$ switches. Each switch has two input buffers. Thus we have a total of $(N \log_2 N)$ buffers in the network. These buffers can be visualized as shown in Figure 4.1. The representation of buffer as outside the actual switch becomes clear on the perusal of the list of assumptions below. There are no buffers at the
output ports of a switch.

5. The buffer has a capacity of one i.e., the network is single-buffered. This is done to simplify the analysis. Exhaustive and accurate analysis of larger sized buffers is difficult, if not impossible, and hence the assumption of single capacity buffers is made. The approach developed for all the three models can be extended to multibuffered networks.

6. All the buffers in a switch are assumed to be identical in nature. The equivalence stems from the mode of operation and the size of the buffer.

7. The enqueuing and dequeuing of packets in different buffers of a stage is assumed to be take place at the same instant. It is also assumed that this process does not involve time delay i.e., they are instantaneous. These are valid assumptions made to simplify the analysis.
Figure 4.1: BUFFERS AT THE INPUTS OF SWITCHES
4.3.2 Description of Network operation:

The primary purpose of buffers in any network is to prevent loss of valuable data. Apart from crossbar, all IN's are capable of losing packets in the switching fabric if there is no buffering. This is because, once at any switch at any stage, if the destination bits of the destinations of both input packets require the same output, then only one of them goes ahead while the other is rejected/ discarded. With the help of buffers, this loss of data is avoided. As we have buffers at the input ports of all switches, it implies that in case of a conflict there is some “store room” where the rejected request can go. This concept is utilized here.

The operation of the network is assumed to be synchronized to a global clock. This implies that at time \( t = 0 \) we have all empty buffers in the network. The switching operation starts once requests are generated by the processors. If there is a conflict, the packet losing the conflict is stored in the buffer. The winning packet is forwarded if only the output link to which it is routed is able to accept the packet. Otherwise it is buffered at this stage itself. Now in the next cycle we have two possibilities, if any input contains a blocked packet:

1. random resolution of conflicts which implies that a blocked packet may get blocked again.

2. prioritized resolution of conflicts where in buffered request has a priority over the other contending request. When both the inputs contain blocked packets, then in case of conflict, one of the two is selected at random.
This dissertation has covered both these strategies and has demonstrated the results (from simulations) for both the cases in section 4.2.5 and section 4.7 respectively.

Time delay, as defined in section 1.3, becomes meaningful in the analysis of buffered networks. Statistically speaking, time delay is as important a parameter as the number of packets delivered in a cycle. This is because there is no meaning in getting larger number of packets, if the time duration between initiation of requests and between satisfaction of requests is very large. Hence the delay a packet experiences is a vital factor in the performance evaluation of the network. Before going to the properties of a buffered network in general, a brief account of Markov processes and chains is given below. The description presented is brief but relevant to the analysis. A detailed account on Markov processes can be found in [51].

4.3.3 Markov Process

Markovian principles form the basis for the analysis of all modern queuing systems. A family of random variables \( \{X(t), t \in T\} \) is called a stochastic process. Here \( t \) is modelled as time. The state space of this process is the set of all values which the random variable \( X(t) \) can assume. Each of these values is called a state. A stochastic process in which the future of the process depends only on the present state and not on the history of the process is called as a Markov process. Formally speaking, "a stochastic process \( \{X(t), t \in T\} \) is a Markov process if for any set of \( (n + 1) \) values \( t_1 \ldots t_i \ldots t_{n+1} \) for \( t \) and any set of \( (n + 1) \) states \( x_1 \ldots x_i \ldots x_{n+1} \), we have
\[ Pr[X(t_{n+1}) = x_{n+1} | X(t_1) = x_1, X(t_2) = x_2, \ldots, X(t_n) = x_n] \]

\[ = Pr[X(t_{n+1}) = x_{n+1} | X(t_n) = x_n] \quad (4.1) \]

Any Markov process wherein the state space is discrete is called as a discrete Markov chain. In a discrete time Markov chain, the process is thought of as making state transitions at times \( t_n = 1, 2, 3, \ldots \). It is these transition probabilities, along with the state probabilities themselves, that define a Markov chain. The one step transition probabilities from state \( i \) to state \( j \) are of primary importance in modelling using a Markov chain. Exhaustive analysis using Markov chains for the operation of a switch is complex. Studies [31] have shown that the number of states grow abnormally large with the increase in \( N \). If the modelling is tried for non-uniform references the states will almost double. Hence there is a need for an iterative solution which gives accurate results. The property of Markovian chains is still utilized but states are reduced by approximations. With the above introduction to the Markov process, we now postulate the different properties of the buffered networks.

4.3.4 Properties of Buffered Networks

1. Once a packet enters a buffered network, the packet cannot be lost. This is because of the presence of a buffer at every switch in each stage. Only at the first stage there is a possibility of a request being lost. A request is generated independent of the availability of the buffer at the first stage. This request is accepted into the network if the first stage buffers are not full.
2. The whole operation of the network is modelled in terms of stage cycles i.e., a stage cycle is the time required to implement the switching operations for a switch in a particular stage. (The operations of all switches in a particular stage are in parallel). Hereafter, a stage cycle is synonymous with a clock cycle. Therefore the total number of stage cycles that make up a complete network cycle is equal to \( \log_2 N \) — the number of stages in the network.

3. The buffers which are at the inputs of hot switches are called as "hot buffers", while the buffers at the inputs of non-hot switches are called as non-hot buffers.

4. At a hot switch, both the buffers are hot buffers. Since hot links carry an increased percentage of traffic, the likelihood of a hot buffer containing a blocked packet is higher and increases with an increase in the number of request directed to hot modules. It is this increase which leads to tree saturation resulting in a reduction in system performance.

5. As the network is assumed to be synchronous and packet switched, the operation of all stages overlaps. (A fine distinction is made in the models presented below to help in the analysis of the network).

In the following section the first model for performance analysis is presented in detail:
4.4 Model 1

This model is motivated by the work of Jenq [33] and Yoon et al [34]. The model developed is presented for uniform reference model. The extension to non-uniform reference model is presented in subsection 1.2.5.1. If a packet wins contention (from contending packets if any), then that packet is able to move forward if the next stage buffer is empty or the next stage buffer is full and the next stage packet is able to move forward in this cycle. This follows from the modelling of operations by stage cycles. In URM the state of a SE at stage $k$ is statistically not distinguishable from that of another SE in the same stage $k$. Hence the state of a stage can be reduced to the state of a single buffer. This assumption does not hold for non-uniform reference model. To simplify the analysis, any clock cycle $t$ is assumed to be made of two phases:

1. In phase 1, control signals are assumed to have passed across the network to determine whether a packet is able to move forward one stage or whether it is to be blocked. This operation should be done from the output stage to the input stage, one stage at a time.

2. The actual routing of packets is assumed to take place in phase 2 of clock cycle $t$. This routing is based on the destination address bit $i$ of each packet at stage $S_r$. (This type of routing is described in section 2.2.3. The occurrence of conflict when the two input packets have same $d_i$'s is explained in section 2.2.4).
The above distinction in a single clock cycle is made to facilitate the analysis. This distinction is not possible in practice. At the start of any cycle distinction is made in the states of a buffer as follows: a buffer is in state '1' if it contains a packet and is in state '0' if it is empty. Let \( n = \log_2 N \) represent the number of stages in the network. The state probabilities are defined as:

\[
p_0(k, t) = \Pr[\text{that a buffer of a SE at stage } k \text{ is in state 0 at the beginning of the } t^\text{th} \text{ stage cycle}] \]

\[
p_1(k, t) = \Pr[\text{that a buffer of a SE at stage } k \text{ is in state 1 at the beginning of the } t^\text{th} \text{ stage cycle}] \]

Lemma: Since there are only two possible states for any buffer, at any given time \( t \)

\[
p_0(t, k) + p_1(t, k) = 1.0 \quad (4.2) \]

This follows from elementary laws of probability.

**Transition Probabilities**

\[
q(k, t) = \Pr[\text{that a packet is ready to come to a buffer of a SE at the beginning of the } t^\text{th} \text{ cycle}] \]

\[
r(k, t) = \Pr[\text{that a packet in a buffer of a SE at stage } k \text{ is able to move forward during the } t^\text{th} \text{ stage cycle, given that there is a packet in that buffer}] \]
\( \rho = \) the bandwidth on a single link i.e., the average number of packets received on a network output link in one clock cycle

\( d = \) average delay a packet experiences before reaching the output (or the average number of cycles taken to reach the output)

*Relations*

Consider the diagram shown in Figure 4.2. This shows the illustration of \( q(k, t) \) and \( r(k, t) \). Based on this, the following relations are derived.

\( q(k, t) \), the probability that a packet is ready to come to a buffer of a SE at stage \( k \) during \( t^\text{th} \) stage cycle is related to the number of packets in buffers of stage \( (k-1) \), which feed the buffer under consideration and destinations of those packets. It is assumed that the packets in different buffers are independent of one another.

As a buffer can have only two states, the probability that a \( 2 \times 2 \) SE at stage \( k \) has \( j \) packets in its 2 input buffers, \( 0 \leq j \leq 2 \), can be obtained by Bernoulli's distribution as equal to \( \binom{2}{j} p_j(k, t) p_{j}(k, t)^{2-j} \). Since a \( 2 \times 2 \) SE is considered and the probability that any one of the output link is referenced is equal to 0.5, the probability that at least one of the \( j \) packets is destined to the upper buffer in stage \( k \) under consideration is equal to \( 1 - \left(1 - \frac{1}{2}\right)^j \). Here \( \left(1 - \frac{1}{2}\right) \) represents the probability that a packet is not destined for the output under consideration and \( \left(1 - \frac{1}{2}\right)^j \) is the probability that none of the \( j \) packets are destined for
Figure 4.2: A CLOCK CYCLE IN MODEL 1
that output link. Hence, the probability that at least one of them is destined for
the output link is given by \( 1 - \left(1 - \frac{1}{2}\right)^j \). Hence from the definition,
\( q(k+1, t) \) is the product of the two probabilities summed over the total number
of packets.

\[
q(k+1, t) = Pr[\text{having } j \text{ packets in stage } k-1 ]
\times Pr[\text{at least one of the packets is destined for the SE under consideration}]
\]

Since \( j \) cannot be greater than 2, we have

\[
q(k+1, t) = \sum_{j=0}^{2} \binom{2}{j} p_i(k, t)^j p_o(k, t)^{2-j} \left[1 - \left(1 - \frac{1}{2}\right)^j\right] \tag{4.3}
\]

\[
= p_o(k, t) p_i(k, t) + \frac{3 p_i(k, t)^2}{4}
\]

Utilizing equation 4.2 we can simplify the above expression to yield

\[
q(k, t) = \left[1 - \left(1 - \frac{p_i(k, t)}{2}\right)^j\right] \tag{4.4}
\]

Let us define \( \hat{H}(k, t) \) as the probability that a packet in a buffer of a SE at stage
\( k \) is able to pass that SE to the desired output port of that SE during the \( t^\text{th} \)
stage cycle by winning the contention among the competing packets (if any), given
that the buffer is in state 1. The other buffer can contain \( j \) packets, \( 0 \leq j \leq 1 \).

Probability that the other buffer contains \( j \) packets = \( \binom{1}{j} p_i(k, t)^j p_o(k, t)^{1-j} \tag{4.5} \)
For any \( j \), let \( i \) be the number of packets which contend with the packet under consideration, \( 0 \leq i \leq j \). Therefore the given packet can win the contention from a total of \( (i+1) \) packets in \( \left( \frac{1}{i+1} \right) \) ways. The probability of accessing a particular output link is 0.5 (URM). Hence \( \hat{r}(k, t) \) can be expressed as

\[
\hat{r}(k, t) = \sum_{j} \Pr[j \text{ packets being present}] \times \\
\sum_{i} \Pr[i \text{ packets accessing the same output link as the given packet}] \times \\
Pr[\text{the given packet wins the contention}] \times \\
Pr[\text{given packet accessing a specific output}] \\
\] (4.6)

\[
\hat{r}(k, t) = \sum_{j=0}^{i} \binom{i}{j} p_i(k, t) p_0(k, t)^{i-j} \sum_{i=0}^{j} \binom{j}{i} \left( \frac{1}{2} \right)^i \left( 1 - \frac{1}{2} \right)^{i-j} \left( \frac{1}{i+1} \right) \left( \frac{1}{2} \right)
\]

This simplifies to

\[
\hat{r}(k, t) = \frac{1}{2} \left[ 1 - \frac{p_i(k, t)}{4} \right] \\
\] (4.7)

Once a packet at stage \( k \) wins the contention, it can move forward if there is an available buffer in stage \( (k+1) \), i.e., if the buffer at stage \( (k+1) \) is empty at the start of cycle \( t \) or it is full and the packet moves forward to stage \( k+2 \). The probability that it moves forward to stage \( k+2 \) in cycle \( t \) is given by \( r(k+1, t) \). Therefore, the probability that a buffer of stage \( k+1 \) is able to accept a packet is given by the expression
\[ p_d(k+1, t) + p_i(k+1, t) \, r(k+1, t) \] \hspace{1cm} (4.8)

From the definition of \( r(k, t) \) mentioned earlier in the chapter, it can be expressed as

\[ r(k, t) = Pr[\text{the packet under consideration wins the conflict}] \times Pr[\text{next stage is able to accept the packet}] \]

Using equation 4.7 and 4.8 we thus get

\[ r(k, t) = \hat{r}(k, t) \left[ p_d(k+1, t) + p_i(k+1, t) \, r(k+1, t) \right], \quad 1 \leq k \leq (n-1) \] \hspace{1cm} (4.9)

At the final stage a packet passes the SE if it wins the conflict, because there is no blocking at the output due to full buffers at succeeding stages. Hence the probability of a message being routed given that there is a packet in the buffer, at stage \( n \) is given by

\[ r(n, t) = \hat{r}(n, t) = \frac{1}{2} \left[ 1 - \frac{p_1(n, t)}{4} \right] \] \hspace{1cm} (4.10)

At time \( t = 0 \), all the buffers are empty and the load is applied at \( t = 1 \). (Load is defined as the requests offered to the network at the start of any clock cycle.) Hence \( q(1, t) \) is the load applied to the network. The probability that a buffer in stage \( k \) is empty at the beginning of next cycle \( t + 1 \) is given by

\[ p_0(k, t+1) = \{ Pr[\text{it was in state 0 at time } t] \times Pr[\text{no packet was received in time cycle } t] \} \]
\[ p_d(k, t+1) = p_d(k, t)(1 - q(k, t)) + p_s(k, t) r(k, t)(1 - q(k, t)) \]

\[ p_d(k, t+1) = (1 - q(k, t))[p_d(k, t) + p_s(k, t) r(k, t)] \]  \hspace{1cm} (4.11)

Based on the state and transition probabilities mentioned above, a state diagram can be drawn as shown in Figure 4.3.

Using the value of \( q(1, t) \) and equations 4.2, 4.4, 4.9, 4.10 and 4.11 one can recursively calculate all the probabilities till a steady state is reached in which all probabilities in each stage becomes time-independent. This calculation is based on the splitting up of the clock cycle, with the operation of each phase as described at the beginning of section 4.3. The stabilized values are then used to calculate the total network bandwidth as follows:

\[ AMBW = Pr[\text{message is received on a output link}] \times (\text{Number of outputs}) \]

which gives

\[ AMBW = p(t, k) \times N \]

which simplifies to

\[ AMBW = p_s(n) r(n) N \]  \hspace{1cm} (4.12)
\[
\bar{q} = 1.0 - q \\
\bar{r} = 1.0 - r
\]

Figure 4.3: STATE DIAGRAM FOR MODEL I
where \( p_1(n) \) and \( r(n) \) are stabilized values. The reciprocal of the probability that a packet at stage \( k \) is successfully routed gives the delay at that stage. Thus the total delay \( d \) for the whole circuit is

\[
d = \sum_{k=1}^{n} \frac{1}{r(k)} \tag{4.13}
\]

The algorithm for calculation of the bandwidth is as follows:

Given \( q(1, t) \):

Begin

\[\text{do 60 (for large number of cycles, typically 50 for 64 x 64 network)}\]

\[\text{do 50 } k = 1, n \text{ in steps of 1} \]

\[\text{calculate } p_0(k, t), p_1(k, t) \]

\[\text{calculate } q(k, t) \]

50 continue

\[\text{calculate } r(n, t) \]

\[\text{do 55 } k = (n-1), 1 \text{ in steps of -1} \]

\[\text{calculate } r(k, t) \]

55 continue

60 continue
calculate the steady state values of \( \rho \) and \( d \)

End

The results of Model 1 for URM are given in section 4.6 as a means of comparison.

4.4.1 Extension Of Model 1 to Non-Uniform references

The above mentioned analysis for URM is now extended to the case for non-uniform references. Consider the 8 x 8 omega network shown in figure 3.1. From the properties and theorems mentioned in chapter 3, it is obvious that we have three different types of switches:

1. Hot switches. Note that each hot switch has two outputs having different data rates.
2. Non-hot switches of type 1, which receive inputs from non-hot outputs of the hot switches in the previous stage.
3. Non-hot switches of type 2, which receive input from the outputs of non-hot switches stage.

In general a network containing \( \log_2 N \) stages will have \( \log_2 N \) types of switches. In order to analyze the networks, each of the statistically different paths in the network has to be evaluated with respect to data rates. Based on the model presented for the URM in the section 4.4, the following notations will be used to modify the URM to the non-URM case.

Acceptance Probabilities
\[ q_{n,n}(k, t) = Pr[\text{a packet is offered a hot switch in stage } k] \]

\[ q_{n,n}(k, t) = Pr[\text{a packet is offered a switch of type 2 in stage } k] \]

\[ q_{n,n}(k, t) = Pr[\text{a packet is offered a switch of type 1 in stage } k] \]

*State Probabilities*

\[ p_{n,n}(k, t) = Pr[\text{that a buffer at the input of a hot switch in stage } k \text{ is empty at the beginning of } t^{th} \text{ cycle}] \]

\[ p_{n,n}(k, t) = Pr[\text{that a buffer at the input of a hot switch in stage } k \text{ is full at the beginning of } t^{th} \text{ cycle}] \]

\[ = 1 - p_{n,n}(k, t) \]

\[ p_{o,n,n}(k, t) = Pr[\text{buffer at the input of a non-hot switch of type 2 is empty at the beginning of } t^{th} \text{ cycle}] \]

\[ p_{o,n,n}(k, t) = Pr[\text{a buffer at the input of a non-hot switch of type 2 is full at the beginning of } t^{th} \text{ cycle}] \]

\[ = 1 - p_{o,n,n}(k, t) \]

\[ p_{o,n,n}(k, t) = Pr[\text{a buffer at the input of a non-hot switch of type 1 is empty at the beginning of } t^{th} \text{ cycle}] \]
\[ p_{n,n,n}(k, t) = \Pr[ \text{a buffer at the input of a non-hot switch of type 1 is full at the beginning of } t^{th} \text{ cycle} ] \]

\[ = 1 - p_{n,n,n}(k, t) \]

**Transition Probabilities**

\[ r_{h}(k, t) = \Pr[ \text{a packet in a buffer at the input of a hot switch in stage } k, \text{ is able to win the contention and move forward to the hot output link, given that there is a packet in the buffer at the start of the cycle} ] \]

\[ r_{n}(k, t) = \Pr[ \text{a packet in a buffer at the input of a hot switch in stage } k, \text{ is able to win the contention and move forward to the non-hot output link, given that there is a packet in the buffer at the start of the cycle} ] \]

\[ r_{n,n}(k, t) = \Pr[ \text{a packet in a buffer at the input of a non-hot switch of type 2 is able to win the contention and move forward to a specific output link, given that there is a packet in the buffer at the start of the cycle} ] \]

\[ r_{n,h}(k, t) = \Pr[ \text{a packet in a buffer at the input of a non-hot switch of type 1 is able to win the contention and move forward to a specific output link, given that there is a packet in the buffer at the start of the cycle} ] \]
The throughput on the input links of a hot switch are defined as

\[ p^h_n(k, t) = Pr[\text{a message is received on hot output link of a hot switch at stage } k] \]

\[ p^\text{n,n}_n(k, t) = Pr[\text{a message is received on the non-hot output link of a hot switch at stage } k] \]

Given a packet at the input of a hot switch, let the probability of requesting a hot output link (at stage \( k \)) be \( P^h_n(\omega) \) while the probability of requesting a non-hot output link (at stage \( k \)) be \( P^\text{n,n}_n(\omega) \). Hence the probability of a hot switch receiving a packet is the probability that in the previous stage the buffer is full and able to route the packet to the hot link. Following in the same way as we did for the URM (equation 4.4), we get expressions for \( q^\text{n,n}_n(k, t) \) and \( q^h_n(k, t) \) as

\[ q^\text{n,n}_n(k, t) = \left[ 1 - (1 - P^h_n(k-1) p^\text{n,n}_n(k-1, t))^T \right] \]  \hspace{1cm} (4.14)

\[ q^h_n(k, t) = \left[ 1 - (1 - P^\text{n,n}_n(k-1) p^h_n(k-1, t))^T \right] \] \hspace{1cm} (4.15)

As derived in the previous section (equation 4.4), \( q^\text{n,n}_n(k, t) \) is given by

\[ q^\text{n,n}_n(k, t) = \left[ 1 - \left( 1 - \frac{p^\text{n,n}_n(k-1, t)}{2} \right) \right]^T \] \hspace{1cm} (4.16)

The expression for a buffer being empty at the beginning of a cycle is evaluated as follows. Any buffer is empty at the beginning of a cycle \( t \), if it was empty at the start of cycle \((t-1)\) and it did not receive any packet in \((t-1)\) or if it was full at the beginning of cycle \((t-1)\) and it routed the packet successfully to any of the
output links, but did not receive any packet during phase 2 in the previous cycle. Expressing this in terms of the probabilities defined above, we have

\[ p_{a,n}(k, t) = [1 - q_{n,n}(k, t-1)] [p_{a,n}(k, t-1) \]

\[ + p_{1,n}(k, t-1) (r^a_n(k, t-1) + r^b_n(k, t-1))] \quad (4.17) \]

\[ p_{a,n,n}(k, t) = [1 - q_{n,n}(k, t-1)] [p_{a,n,n}(k, t-1) \]

\[ + p_{1,n,n}(k, t-1) 2 r_{n,n}(k, t-1)] \quad (4.18) \]

The multiplying factor of '2' in the second term comes from the fact that \( r_{n,n}(k, t-1) \) is the probability of being routed to a specific output link which can be any one of the two output links.

\[ p_{a,n,n}(k, t) = [1 - q_{n,n}(k, t-1)] [p_{a,n,n}(k, t-1) \]

\[ + p_{1,n,n}(k, t-1) 2 r_{n,n}(k, t-1)] \quad (4.19) \]

The transition probabilities are obtained as follows:

A buffer in stage \( k+1 \) is able to accept a hot packet from stage \( k \) during a cycle if it is empty during that cycle or it is full but is able to route its own packet successfully. A packet in stage \( k \) wins the contention from \( j \) packets before getting routed. The probability that a hot packet at the hot switch wins the contention among the contending packets and passes on to the hot output link during cycle \( t \) is denoted by \( p^a_n(k, t) \) and is given by the following equation
\[
\hat{\tau}_h^{(k,t)} = \sum_{j=0}^{i} \binom{i}{j} p_{1,h,n}(k, t)^j p_{0,h,n}(k, t)^{i-j} \sum_{i=0}^{\infty} \binom{i}{i} \frac{1}{i+1} p_{\text{h}}(k)^i P_{\text{h}}(k)^{i-j} P_{\text{h}}(k)
\]

which reduces to

\[
\hat{\tau}_h^{(k,t)} = P_{\text{h}}(k) \left[ 1 - \frac{P_{\text{h}}(k) p_{1,h,n}(k, t)}{2} \right]
\] (4.20)

Probability that a non-hot packet at a hot switch wins the contention and passes on to the non-hot output link, denoted by \(\hat{\tau}_n^{(k,t)}\), is given by

\[
\hat{\tau}_n^{(k,t)} = \sum_{j=0}^{i} \binom{i}{j} p_{1,h,n}(k, t)^j p_{0,h,n}(k, t)^{i-j} \sum_{i=0}^{\infty} \binom{i}{i} \frac{1}{i+1} P_{\text{n}}(k)^i P_{\text{n}}(k)^{i-j} P_{\text{n}}(k)
\]

which reduces to

\[
\hat{\tau}_n^{(k,t)} = P_{\text{n}}(k) \left( 1 - \frac{P_{\text{n}}(k) p_{1,h,n}(k, t)}{2} \right)
\] (4.21)

Hence the probability that a packet in a hot switch is routed to the hot output link is given by

\[
r_h^{(k,t)} = P_{\text{h}}(k) \left[ 1 - \frac{P_{\text{h}}(k) p_{1,h,n}(k, t)}{2} \right] \times \\
\left[ p_{0,h,n}(k+1, t) + (p_{1,h,n}(k+1, t) (r_h^{(k,t)}(k+1, t) + r_n^{(k+1,t)})) \right]
\] (4.22)

and the probability that a packet is routed to the non-hot output link is given by

\[
r_n^{(k,t)} = P_{\text{n}}(k) \left[ 1 - \frac{P_{\text{n}}(k) p_{1,h,n}(k, t)}{2} \right] \times \\
\left[ p_{0,h,n}(k+1, t) + (p_{1,h,n}(k+1, t) (r_h^{(k,t)}(k+1, t) + r_n^{(k+1,t)})) \right]
\]
\[
[p_{\text{a}, \text{H}}(k+1, t) + p_{\text{t}, \text{H}}(k+1, t)]
2 r_{\text{H}, \text{H}}(k+1, t)\]
\]

(4.23)

For the two types of non-hot switches, the corresponding expressions for the routing probabilities are just a simple extension of the derivation of \(r(k, t)\) in the section 4.3 (equation 4.7). They can be explicitly written as:

\[
r_{\text{H}, \text{H}}(k, t) = \frac{1}{2} \left[ 1 - \frac{p_{\text{t}, \text{H}}(k, t)}{4} \right] \times
\]

\[
[p_{\text{a}, \text{H}}(k+1, t) + p_{\text{t}, \text{H}}(k+1, t)] 2 r_{\text{H}, \text{H}}(k+1, t)\]
\]

(4.24)

and

\[
r_{\text{H}, \text{H}}(k, t) = \frac{1}{2} \left[ 1 - \frac{p_{\text{t}, \text{H}}(k, t)}{4} \right] \times
\]

\[
[p_{\text{a}, \text{H}}(k+1, t) + p_{\text{t}, \text{H}}(k+1, t)] 2 r_{\text{H}, \text{H}}(k+1, t)\]
\]

(4.25)

The term \(\left(\frac{1}{2}\right)\) comes from the fact that \(p_{\text{H}}(k, t)\) and \(p_{\text{H}}(k, t)\) have been defined as the probability of being routed to a single specific output link. At the final stage the different transition probabilities are defined as

\[
r_{\text{H}}^t(n, t) = P_{\text{H}}(n) \left[ 1 - \frac{P_{\text{H}}(n) p_{\text{t}, \text{H}}(n, t)}{2} \right]
\]

(4.26)

\[
r_{\text{H}}^l(n, t) = \frac{1}{2} \left[ 1 - \frac{p_{\text{t}, \text{H}}(n, t)}{4} \right]
\]

(4.27)

\[
r_{\text{H}, \text{H}}(n, t) = \frac{1}{2} \left[ 1 - \frac{p_{\text{t}, \text{H}}(n, t)}{4} \right]
\]

(4.28)
\[ r_{n,n}(n, t) = \frac{1}{2} \left[ 1 - \frac{p_{1,n,n}(n, t)}{4} \right] \]  

The throughput at the output links of the different types of switches are defined as follows

\[ p^h_R(k, t) = Pr[\text{a message is received on hot output link of a hot switch at stage } k] \]

\[ = 2.0 \left[ p_{1,n,n}(k, t) \cdot r^h_R(k, t) \right] \]  

The multiplying factor of 2 comes from the fact that a message received on an output link can come from any of the two input links.

\[ p^n_R(k, t) = Pr[\text{a message is received on the non-hot output link of a hot switch at stage } k] \]

\[ = 2.0 \left[ p_{1,n,n}(k, t) \cdot r^n_R(k, t) \right] \]  

\[ p_{n,n}(k, t) \text{ and } p_{n,n}(k, t) \text{ for a specific output link of non-hot switches are given by} \]

\[ p_{n,n}(k, t) = 2.0 \left[ p_{1,n,n}(k, t) \cdot r_{n,n}(k, t) \right] \]  

\[ p_{n,n}(k, t) = 2.0 \left[ p_{1,n,n}(k, t) \cdot r_{n,n}(k, t) \right] \]

Thus with the closed form of equations presented above from equations 4.14-4.32 and with the initial load specified, a steady state is reached by iterating the equations. At steady state the bandwidth of a 8 x 8 network is computed as
\[ AMBW = \rho_H^1(n) + \rho_H^5(n) + 2\ \rho_{H,N}(n) + 4\ \rho_{N,H}(n) \] (4.33)

The above mentioned method can be extended to a network of arbitrary size if the topology of the network is known and the number of different type of switches in each stage are known. The algorithm to be followed is given by

Step 0:  identification of the different types of switches in a network.
         This entirely depends on the number of stages in a network.
         In a 8 x 8 network we have switches of three types as explained at the beginning of section 4.4.1

Step 1:  The acceptance and state probabilities for the switches at the first stage, which are all hot switches, are evaluated.

Step 2:  The acceptance and state probabilities of the different type of switches in different stages are now evaluated for this cycle.
         For a 8 x 8 omega network we evaluate only \( q_{H,N} \), \( q_{N,H} \), \( p_{0,H,N} \) and \( p_{0,N,H} \) at the second stage while at the final stage, since we have all the three types of switches, all acceptance and state probabilities are found.

Step 3:  Once all the acceptance and state probabilities of all the stages are obtained, \( r(k, t) \) is calculated for the final stage for all types of switches.

Step 4:  Proceeding backwards, all the other transition probabilities in other stages are calculated.
Step 5: After all the probabilities in a cycle are obtained we go back to step 2 to start the next cycle. The above iterations are done till a steady state is reached. At steady state, the bandwidth is evaluated from equation 4.33.

The time delay experienced by a packet once it enters the network can also be computed. Delay at a stage is reciprocal of the probability that the packet can pass that stage. For an 8 x 8 omega network, four different types of delay are possible: \( d^h_n \) – delay for the packet destined for the hot module, \( d^n_h \) – delay for the packet destined for the non-hot module which is connected to the hot switch at the final stage; \( d_{n,n} \) – delay for the packet arriving on the output of type 1 switch in stage n and \( d_{n,n} \) – for packets arriving on output links of switches of type 2 at stage n.

\[
d^h_n = \sum_{k=1}^{s} \frac{1}{r^h_n(k)}
\]

\[
d^n_h = \frac{1}{r^h_n(1)} + \frac{1}{r^h_n(2)} + \frac{1}{r^h_n(3)}
\]

\[
d_{n,n} = \frac{1}{r^h_n(1)} + \frac{1}{r^h_n(2)} + \frac{1}{r_{n,n}(3)}
\]

\[
d_{n,n} = \frac{1}{r^h_n(1)} + \frac{1}{r_{n,n}(2)} + \frac{1}{r_{n,n}(3)}
\]
4.4.2 Simulation

Simulations were carried out to validate the results obtained from the analytical model described in section 4.3. The generation of requests according to various request rates and hotspot probabilities followed the same method as explained in sec 3.4. A single capacity buffer was assumed at the input of every switch in every stage. Separate arrays at each switch input link facilitated the demarcation of fresh requests from the blocked requests. A record of the cycle in which the packet was accepted into the network was kept by appending the cycle number to the packet. At the final stage the cycle in which the request is satisfied was used to get the delay the packet experienced in reaching the output. The whole operation was modelled as stage cycles. The process proceeds from stage n to stage 1, doing the switch operation in successive stages. After stage 1 is reached, new requests are generated. Then the cycle starts again from stage n to stage 1. Destination tag routing was assumed and the resolution of conflicts was random. The number of valid packets at the output of the network at the end of any cycle gives the memory bandwidth of the network. The simulations were performed for various network sizes against varying degrees of hotspots and also for various request rates. A comprehensive set of data was obtained for memory bandwidth and time delay. The number of cycles for simulations were large (typically 50000). Results obtained from the first 500 cycles were discarded in order to allow the network to reach steady state. A flowchart illustrating the complete simulation procedure for buffered networks is shown in Figure 4.4.
Figure 4.4: FLOWCHART ILLUSTRATING THE SIMULATION PROCEDURE (contd...)
Figure 4.4: (.... contd)
4.4.3 Results

With the model mentioned in section 4.4.1, the memory bandwidth was computed for an 8 x 8 omega networks. Simulations were also performed as mentioned in section 4.4.2. The results obtained are shown in Figure 4.5 and Figure 4.6 for increasing request rates for different hotspot probabilities. At lower loads the difference between the analytical and simulation results is reasonably small, but at higher loads the difference is higher as can be seen from Figure 4.5. Also evident is the fact that for higher hotspot probabilities the analytical and simulation results show a high degree of difference. The reasons for the large errors are described in section 4.4.3.1. The delay obtained by analytical model for a 8 x 8 network under URM is shown in Figure 4.7. In this figure, the delay at various stages is also given. This figure illustrates the fact that for URM, the model gives accurate results. The delay at first stage is highest and is lowest at the last stage.
Figure 4.5: Bandwidth obtained from model 1 against request rate for different hotspot probabilities
Figure 4.6: PERCENTAGE ERROR IN BANDWIDTH OBTAINED FROM MODEL 1 AGAINST REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES
Figure 4.7: DELAY OBTAINED FROM MODEL 1 COMPARED WITH SIMULATION RESULTS, FOR AN 8 x 8 OMEGA NETWORK UNDER URM
4.4.3.1 Drawbacks

The analytical results presented are highly optimistic. This is because of the assumptions made. The following explanation gives an insight as to why the results from analytical modelling are different from those of simulations.

The intrinsic assumption made in the model is that the requests in consecutive clock cycles are independent and that the states of the buffers in adjacent stages are independent too. These assumptions are not realistic. When a packet is blocked in any particular cycle, the above assumption implies that its destination in the next cycle can still be any one of the two output links. This is not true in a real environment, since a packet blocked in this cycle always requests the same output link again in the next cycle. Also, it was assumed that irrespective of what happened to the packet in this stage \((k)\), the probability that the next stage \((k+1)\) buffer is able to accept it is given by the expression \(p_a(k+1, t) + p_r(k+1, t) r(k+1, t)\). This is again not true in all cases. If at a switch in any stage, two packets contend for the same output link in this cycle, one of them wins the contention and goes to the next stage buffer if the next stage buffer is not full or if it is full and the packet can go through in this cycle. Otherwise, it stays in the buffer at this stage. The packet which lost the contention will see a full buffer in the next cycle irrespective of whether the winning packet went through or was blocked. Thus, for the packet which lost the contention, the probability that it sees a full buffer in the next stage in the next cycle is always unity. At higher input loads, because of higher degree of blocking in successive stages, the inaccuracy due to the above mentioned reasons will be high. The degradation is even higher in the case of a hotspot. This is because of
the increased traffic on the hot links in the network. Due to this, a hot packet in
the buffer of stage \( k \) always has to contend with a hot packet and will find a hot
packet in succeeding stage buffer if it wins the contention.

To verify the above concepts by simulation, a separate simulation study was
performed as follows. At each stage, if a request is blocked during cycle \( t \), the
destination of the blocked requests were regenerated at cycle \( t + 1 \). The new des-
tination is one from the set of memory modules accessible by the switch. The
regenerated requests followed the same hotspot distribution. An example is giv-
en as follows:

Let the request rate be unity for a 8 x 8 omega network with a hotspot probabil-
ity of 0.3. Say a request is blocked at switch SW5 in figure 3.1. The set of
memory modules accessibe from this switch are \( \{MM_4, MM_5, MM_6 \text{ and } MM_7\} \).
Let the original request be for \( MM_6 \). \( MM_4 \) is the hot memory module. On
regeneration, the hotspot distribution i.e., the ratio of the probability of access-
ing the hot memory module (0.3) to that of accessing the non-hot memory mod-
ule (0.1) is maintained the same. Hence at SW5, the range of generation of the
requests is divided as per the above ratio.

The result obtained for a hotspot probability of 0.3 is shown in Figure 4.8
for an 8 x 8 network. The results show a increased proximity to the results of
the analytical model presented above. Thus the model presented above tends to
be inaccurate at higher loads and needs to be reformed. In the following section
a highly refined model is presented.
Figure 4.8: EFFECT OF HAVING INDEPENDENT REQUESTS IN THE BUFFERS; N = 8, REQUEST RATE = 1 AND HOTSPOT PROBABILITY = 0.3
4.5 Model 2

This model is motivated by the work presented by Theimer et al in [49].

As presented in the last section, the inaccuracy in the results of model 1 is due to the assumption of independence between consecutive cycles and between buffers of adjacent stages. These were also verified by simulations in section 4.4.3.1. To improve the accuracy, the following model is developed. It takes into account the dependencies mentioned above.

All the assumptions outlined in section 4.3.1 hold for this model. Besides that, the operation of the network is also based on the properties mentioned in section 4.3.2 and 4.3.4. The basic difference from model 1 comes from the modelling of the different states of the buffer to account for the independence assumptions.

The underlying principle in the operation of a buffer is still the same. A packet can be forwarded in the current cycle if it wins the conflict (if any) with the packet in the other buffer and if the destination buffer is ready to accept it; otherwise it waits in the current buffer. To take into account the correlation between the buffered packets in consecutive cycles and that between the buffers at adjacent stages, the different states of buffers at the beginning of a cycle are identified as follows:

State '0' buffer is empty
State '1' buffer contains a packet which arrived in the last clock cycle
State '2' buffer contains a packet which has been delayed for at least one clock cycle
Here, as opposed to model 1, we have three states. In model 1, no distinction was made between a blocked packet or a new packet.

The clock cycle is split up into two parts

**phase 1**  
A buffer containing a packet forwards the packet to the next stage if it wins the conflict (if any) with the packet in the other buffer and if the next stage buffer is able to accept it. Hence at the end of phase 1 of any cycle, the intermediate state of a buffer can be

1. state '0' if the packet could pass through in the current cycle or if the buffer was in state '0' at the start of the cycle
2. state 'b' if the packet could not pass through.

**phase 2**  
In this phase, buffers at stage (k-1) offer packets to buffers at stage k. The offered packets are accepted by only those buffers having intermediate state '0' in stage k. The probability of a packet being offered is independent of the state of the buffer in the succeeding stage.

The above mentioned demarcation of a stage cycle into two phases is only to help the analysis. In a real parallel processing system, all activities are in parallel.

The state diagram for this model is shown in Figure 4.9.

With the above mentioned guidelines, the following probabilities are defined:

\[ q(t,k) = Pr\{ \text{that a packet is offered to a buffer of a SE at stage } k \text{ during phase 2 of the } t^t \text{ cycle} \} \]
Figure 4.9: STATE DIAGRAM FOR MODEL 2
State Probabilities:

\[ p_x(t,k) = \Pr[ \text{that an input buffer in stage } k \text{ is in state } x \text{ at the beginning of phase 1 of } t^k \text{ cycle, } x \in \{0, n, b\}] \]

Intermediate State Probabilities:

\[ \tilde{p}_x(t,k) = \Pr[ \text{that an input buffer at stage } k \text{ is in state } x \text{ at the end of phase 1 (or at the beginning of phase 2) of } t^k \text{ cycle, } x \in \{0, b\}] \]

Transition Probabilities:

\[ r_x^l(t,k) = \Pr[ \text{that a packet in buffer of a SE at stage } k \text{ is able to move forward during phase 1 of } t^k \text{ cycle, to a specific output link, given that the buffer is in state } x \text{ at the beginning of phase 1 of } t^k \text{ cycle, } x \in \{n, b\}] \]

\[ R_x^l(t,k) = \Pr[ \text{that a packet in buffer of a SE at stage } k \text{ is unable to move forward (to either output links) during phase 1 of } t^k \text{ cycle, given that the buffer is in state } x \text{ at the beginning of phase 1 of } t^k \text{ cycle, } x \in \{n, b\}] \]

With the above probabilities defined, the state transitions in a clock cycle are as shown in Figure 4.9 on page 161. In order to be consistent with the explanation, the clock cycle has been demarcated into two phases. In addition to the
probabilities defined above some other probabilities need to be defined as follows:

\[ p_s(t,k) = \Pr[ \text{a buffer at stage } k \text{ is able to accept a packet during phase } 2 \text{ of the } t^\text{th} \text{ cycle}] \]

\[ p_s^x(t,k) = \Pr[ \text{a buffer at stage } k \text{ is able to accept a packet during phase } 2 \text{ of } t^\text{th} \text{ cycle, given that the buffer is in state } x \text{ at the start of phase } 1, x \in \{0, n, b\}] \]

\[ p(t,k) = \Pr[ \text{that a packet is received on a specific input link of a SE at stage } k \text{ during phase } 2 \text{ of the } t^\text{th} \text{ clock cycle}] \]

Based on the state diagram shown in Figure 4.9 on page 161 the following derivations for intermediate state probabilities are made. In the \(t^\text{th}\) stage cycle

\[ \tilde{p}_0(t,k) = \Pr[\text{a buffer at stage } k \text{ is in state } 0 \text{ at the end of phase } 1]\]

\[ = \Pr[\text{buffer was in state } 0 \text{ at the beginning of phase } 1]\]

\[ + \Pr[\text{buffer was in state } n \text{ at the beginning of phase } 1] \times \]

\[ \Pr[\text{packet was routed to either output links given that the buffer was in state } n]\]

\[ + \Pr[\text{that buffer was in state } b \text{ at the beginning of phase } 1] \times \]

\[ \Pr[\text{packet was routed to either output links given that the buffer was in state } b]\]

\[ = p_0(t,k) + p_n(t,k) 2 r_0(t,k) + p_b(t,k) 2 r_b(t,k) \quad (4.34) \]
\[ \tilde{p}_b(t,k) = Pr[\text{a buffer at stage } k \text{ is in state } b \text{ at the end of phase 1}] \]

\[ = Pr[\text{that buffer was in state } n \text{ at the beginning of phase 1}] \times \]

\[ Pr[\text{packet was unable to go to either output link given that the buffer was in state } n] \]

\[ + Pr[\text{that buffer was in state } b \text{ at the beginning of phase 1}] \times \]

\[ Pr[\text{packet was unable to go to either output link given buffer was in } b] \]

\[ = p_a(t,k) R_b^2(t,k) + p_b(t,k) R_b^0(t,k) \quad (4.35) \]

The state probabilities at the end of phase 2 of the \( t \)-th cycle: ( or at the beginning of the \( (t+1) \)-th cycle )

\[ p_0(t+1,k) = Pr[\text{buffer was in state } 0 \text{ at the end of phase 2 of } t \text{-th cycle}] \]

\[ = Pr[\text{it was in state } 0 \text{ at the end of phase 1 of } t \text{-th cycle}] \times \]

\[ Pr[\text{no fresh packet was offered in phase 2 of } t \text{-th cycle}] \]

\[ = \tilde{p}_0(t,k) [1 - q(t,k)] \quad (4.36) \]

\[ p_n(t+1,k) = Pr[\text{buffer was in state } n \text{ at the end of phase 2 of } t \text{-th cycle}] \]

\[ = Pr[\text{it was in state } 0 \text{ at the end of phase 1 of } t \text{-th cycle}] \times \]

\[ Pr[\text{a fresh packet was offered in phase 2 of } t \text{-th cycle}] \]

\[ = \tilde{p}_0(t,k) q(t,k) \quad (4.37) \]
\[ p_b(t + 1, k) = Pr[\text{buffer was in state } b \text{ at the end of phase } 2 \text{ of } t^k \text{ cycle}] \]

\[ = Pr[\text{the buffer was in state } b \text{ after phase } 1 \text{ of } t^k \text{ cycle}] \]

\[ = \tilde{p}_b(t, k) \tag{4.38} \]

Acceptance Probabilities:

\[ p^a_a(t, k) = Pr[\text{a buffer at stage } k \text{ is able accept a packet during phase } 2 \text{ of } t^k \text{ cycle} \]

\[ \text{given that the buffer was in state } n \text{ at the beginning of } t^k \text{ cycle}] \]

\[ = Pr[\text{a packet is routed to either output links in phase } 1, \text{ given buffer was in state } n \text{ at begining of phase } 1] \]

\[ = 2 \mu_a(t, k) \tag{4.39} \]

\[ p^b_a(t, k) = Pr[\text{a buffer at stage } k \text{ is able accept a packet during phase } 2 \text{ of } t^k \text{ cycle} \]

\[ \text{given that the buffer was in state } b \text{ at the beginning of } t^k \text{ cycle}] \]

\[ p^b_b(t, k) = Pr[\text{a packet at stage } k \text{ is routed to either output links in phase } 1, \text{ given buffer was in state } b \text{ at the beginning of phase } 1] \]

\[ = 2 \mu_b(t, k) \tag{4.40} \]

\[ p_a(t, k) = Pr[\text{buffer is empty after phase } 1 \text{ of } t^k \text{ cycle}] \]
\[ r_k^1(t, k) = Pr[ \text{the packet in a buffer of a SE at stage } k \text{ is able to move} \]
\[ \text{forward during phase 1 of the } t^\text{th} \text{ cycle to a specific output link} \text{ given that the buffer is in state } n ] \]

\[ = y_1 + y_2 + y_3 + y_4 + y_5 \quad (4.41a) \]

where the terms \( y_1, y_2, y_3, y_4 \) and \( y_5 \) are given as follows (If one of the buffers at a switch is in state \( n \) and the other is in either state 0 or state \( n \), then the next stage buffer can be in any one of the three states 0, \( n \) or b.):

\[ y_1 = Pr[\text{the given packet accesses a particular output link}] \times \]

\[ Pr[\text{other buffer is in state } n] \times \]

\[ Pr[\text{other buffer accesses the other output link}] \times \]

\[ Pr[\text{next stage is able accept the given packet}] \]

\[ = (0.5) [p_x(t, k) (0.5)] p_x(t, k+1) \quad (4.42) \]

\[ y_2 = Pr[\text{the given packet accesses a particular output link}] \times \]

\[ Pr[\text{other buffer is in state } n] \times \]

\[ Pr[\text{other buffer accesses the same output link}] \times \]
\[ Pr[\text{the given packet wins the conflict}] \times \]

\[ Pr[\text{next stage is able accept the given packet}] \]

\[ = (0.5) [p_{*t,k}(0.5)] (0.5) p_{*t,k+1} \]  

(4.43)

\[ y_3 = Pr[\text{the given packet accesses a particular output link}] \times \]

\[ Pr[\text{other buffer is in state } b] \times \]

\[ Pr[\text{other buffer does not access the same link}] \times \]

\[ Pr[\text{next stage is able accept the given packet}] \]

\[ = (0.5) p_{*t,k}(0.5) (0.5) p_{*t,k+1} \]  

(4.44)

\[ y_4 = Pr[\text{the given packet accesses a particular output link}] \times \]

\[ Pr[\text{other buffer is in state } b] \times \]

\[ Pr[\text{other buffer accesses the same output link}] \times \]

\[ Pr[\text{given packet wins the conflict}] \times \]

\[ Pr[\text{next stage buffer is able accept it}] \]

If a SE contains a new packet at one of its links and the other link contains a packet blocked for the same link, then the next stage buffer is in either state \( n \) or state \( b \). It cannot be in state \( 0 \) because a packet may have gone from one of
the links in stage $k$ (because stage $k$ contains a blocked packet), in which case it will be in state 'n' or it may contain some blocked packet which could not be forwarded, due to which the packet at stage $k$ was blocked, in which case stage $(k+1)$ is in state 'b'. The next stage $(k+1)$ buffer is in state 'n' if it received a packet in the previous cycle from stage $k$. The probability that it received a packet from stage $k$ during phase 2 of $(t-1)^{th}$ and which came from a specific input, is given by $0.5 \rho(t-1,k+1)$. Buffer at stage $(k+1)$ is in 'b' if it did not receive a packet from stage $k$. Let the probability that the next stage buffer is able to accept the given packet in this case be denoted as $y_\epsilon$. Then

$$y_\epsilon = Pr[\text{next stage buffer is able accept the given packet}]$$

$$= \{Pr[\text{next stage is in state n}] \times Pr[\text{next stage can accept a packet given it is in state n}] + \{Pr[\text{next stage is in state b}] \times Pr[\text{next stage can accept a packet given it is in state b}]\right)$$

$$= 0.5 \rho(t-1,k+1) \rho_a^o(t,k+1) + [1 - 0.5 \rho(t-1,k+1)] \rho_a^o(t,k+1)$$

(4.45)

Therefore,

$$y_\epsilon = (0.5) p_a(t,k) (0.5) (0.5) y_\epsilon$$

(4.46)

$$y_\delta = Pr[\text{the given packet accesses a particular output link}] \times Pr[\text{the other buffer is in state 0}]$$
Pr[next state buffer is able accept the given packet]

\[ \begin{align*}
&= 0.5 p_a(t,k) p_a(t,k+1) \\
&+ 0.125 p_a(t,k) \left( p_a(t,k+1) p(t-1,k+1) \right) \\
&+ p_a(t,k+1) \left[ 1 - 0.5 p(t-1,k+1) \right] \\
&= 1 - 2 r_a(t,k)
\end{align*} \]  \hspace{1cm} (4.47)

Substituting \( y_1, y_2, y_3, y_4 \), \( y_5 \), \( y_6 \), and \( y_7 \) in equation 4.41a, we get the simplified form as

\[ r_a'(t,k) = p_a(t,k+1) \left[ 0.5 - 0.125 p_a(t,k) - 0.5 p_a(t,k) \right] + 0.125 p_a(t,k) \left( p_a(t,k+1) p(t-1,k+1) \right) + p_a(t,k+1) \left[ 1 - 0.5 p(t-1,k+1) \right] \]  \hspace{1cm} (4.48)

Since the packet under consideration is a blocked packet, the next stage buffer must be in either state \( n \) or state \( b \) as was explained in the derivation of equation 4.45.

\[ x_i = Pr[\text{the given packet accessing a particular output link}] \times \]

\[ Pr[\text{other buffer is in state } n] \times \]

\[ Pr[\text{other packet is not accessing the same output link}] \times \]
\( Pr[\text{next stage buffer being able accept the given packet}] \)

\[ = (0.5) p_n(t, k) (0.5) y_6 \]  (4.50)

\[ x_2 = Pr[\text{the given packet accesses a particular output link}] \times \]

\[ Pr[\text{other buffer is in state n}] \times \]

\[ Pr[\text{other buffer accesses the same output link}] \times \]

\[ Pr[\text{given packet wins the conflict}] \times \]

\[ Pr[\text{next stage buffer accepts the given packet}] \]

\[ = (0.5) p_n(t, k) (0.5) (0.5) y_6 \]  (4.51)

\[ x_3 = Pr[\text{the given packet accesses a particular output link}] \times \]

\[ Pr[\text{other buffer is in state b}] \times \]

\[ Pr[\text{other packet does not access the same output link}] \times \]

\[ Pr[\text{next stage is able accept the packet}] \]

When both the buffers of a SE at a stage are blocked the next stage buffer is also in a blocked state. Therefore,

\[ x_3 = (0.5) p_n(t, k) (0.5) p_s(t, k + 1) \]  (4.52)

\[ x_4 = Pr[\text{the given packet accessing a particular output link}] \times \]
\[ P[\text{other buffer is in state } b] \times \]

\[ P[\text{other packet accesses the same output link}] \times \]

\[ P[\text{given packet wins the conflict}] \times \]

\[ P[\text{next stage buffer accepts the given packet}] \]

\[ = (0.5) p_\delta(t,k)(0.5)(0.5) p_\eta(t,k+1) \quad (4.53) \]

\[ x_3 = P[\text{given packet accessing a particular output link}] \times \]

\[ P[\text{other buffer is empty}] \times \]

\[ P[\text{next stage buffer accepts the given packet}] \]

\[ = (0.5) p_\delta(t,k) y_6 \quad (4.54) \]

Substituting the values of \( x_1, x_2, x_3, x_4 \) and \( x_3 \) in equation 4.49a, we get

\[ r_1(t,k) = \{ [0.5 - 0.125 (p_\delta(t,k) + p_\delta(t,k))] \times \]

\[ [p_\eta(t,k+1)(0.5) p(t-1,k+1) + p_\eta(t,k+1)(1 - 0.5 p(t-1,k+1))] \}

\[ + 0.1875 p(t-1,k+1) p_\delta(t,k) [p_\eta(t,k+1) - p_\eta(t,k+1)] \quad (4.55) \]

\[ R_\delta = 1 - 2 r_1(t,k) \quad (4.56) \]

\[ p(t,k) = P[\text{a packet is received on an input link of a SE at} \]

stage k during phase 2 of \( t^\delta \) cycle]
\[ = Pr[\text{a packet is offered an input link of a SE at stage } k \text{ during phase 2}] \times Pr[\text{packet is accepted by the SE}] \]

\[ = q(t,k) p_s(t,k) \quad (4.57) \]

Equation 4.57 is used to find out \( q(t, k) \). For finding out \( p(t,k) \) at any stage, the state and the transition probabilities of the previous stage are needed as shown below:

\[ p(t,k+1) = Pr[\text{a packet is received on an input link of a SE at stage } k+1 \text{ during phase 2 of } t^{th} \text{ cycle}] \]

\[ = Pr[\text{a packet is transmitted to a output link of a SE at stage } k] \]

\[ = 2\{ Pr[\text{buffer was in state n}] \times \]

\[ Pr[\text{packet was routed to a particular output link}], \quad 1 \leq k \leq (n-1) \]

\[ + Pr[\text{buffer was in state b}] \times \]

\[ Pr[\text{packet was routed a output link}] \} \]

\[ = 2[p_s(t,k) r_1(t,k) + p_s(t,k) r_2(t,k)] \quad (4.58) \]

(The 'multiplying factor of 2 in equation 4.58 comes from the fact that the packet can come from either input links of stage } k \).}

Equations 4.42 - 4.58 are valid for all the stages except the last stage.
At the final stage, there is no blocking at the output links. Hence a packet has to only win the contention to go through. Moreover, at the final stage, both the buffers cannot be in state b as the network outputs are assumed to empty instantaneously. Hence substituting the probability of acceptance by a next stage as one in equations 4.42 to 4.55, we get the corresponding expressions for \( r^t_n(t,n) \) and \( r^r_n(t,n) \) as

\[
\begin{align*}
  r^t_n(t,n) &= 0.5[1 - 0.25(p^t_n(t,n) + p^r_n(t,n))] \\
  r^r_n(t,n) &= 0.5[1 - 0.25(p^t_n(t,n) + p^r_n(t,n))]
\end{align*}
\]

With the above expressions, a number of iterations are done till a steady state value is reached for throughput on input links i.e., \( \rho(t,k) \). The whole procedure is summarized below:

1. **Initial load** = \( q(1,1) \)

2. **Step 1**: find \( p^t_n(t,k) \), \( r^t_n(t,k) \), \( R^t_n(t,k) \) from equations 4.36-4.38, 4.48, 4.49, 4.55, 4.56

3. **Step 2**: find \( \tilde{p}_2(t,k) \), \( p^t_n(t,k) \), \( p^r_n(t,k) \) from equations 4.34, 4.35, 4.41, 4.39, 4.40

4. **Step 3**: if \( k \neq n \) find \( \rho(t,k+1) \) from equation 4.58, find \( q(t,k+1) \) from equation 4.57

5. **Step 4**: if \( k = 1 \) find \( \rho(t,k) \) from equation 4.57

6. **Step 5**: if \( \rho(t,k) \) is not stabilized, repeat the whole procedure from step 1
The results of model 2 for URM are given in section 4.6. The above mentioned procedure can be extended to non-uniform reference model as shown below.

4.5.1 Extension Of Model 2 to Non-Uniform Reference Model

The equations obtained for the uniform reference model can now be extended to the non-uniform reference case following the same type of approach as followed in extending model 1 to non-URM in section 4.3.1. The procedure involves identification of the different types of switches present at each stage in the network and also the different types of paths existing in the network. The analysis presented below is for a 8 x 8 omega network. This can be extended to any network of any arbitrary size. The notation to be used for different probabilities are as follows:

A hot switch at any stage can be in any one of the following three states:

- state '0': empty state
- state 'n': new state
- state 'b': blocked state

The different probabilities for hot switches are denoted with a prefix h while for switches of type 1 they are are denoted with a prefix n. For the switches of type 2, the probabilities are denoted with a prefix o. The different types of switches are described at the beginning of section 4.3. Based on the logic presented in the section 4.4* and with the same identification of different probabilities, we have the following equations. Equations 4.66 - 4.71 are true for all stages.
except the last stage. For the last stage, separate expressions are given in equations 4.78 - 4.81. Note that at a hot switch a packet can be routed to either the hot output or the non-hot output; hence correspondingly we have \( hr_{th}^j \) and \( hr_{nh}^j \) for the routing probabilities to the hot link and the non-hot link respectively. The probability of accessing a hot link is \( P_{th} \) and accessing a non-hot link is \( P_{nh} \), at a hot switch as opposed to 0.5 each for non-hot switches.

Intermediate state probabilities at hot Switches:

\[
\tilde{h}p_{0}(t,k) = hp_{0}(t,k) + hp_{a}(t,k) [hr_{th}^j(t,k) + hr_{nh}^j(t,k)]
\]

\[
\quad + hp_{a}(t,k) [hr_{th}^j(t,k) + hr_{nh}^j(t,k)] \tag{4.61}
\]

\[
\tilde{h}p_{a}(t,k) = hp_{a}(t,k) hR_{th}(t,k) + hp_{a}(t,k) hR_{nh}(t,k) \tag{4.62}
\]

Intermediate state probabilities for type 1 switch

\[
n\tilde{p}_{0}(t,k) = np_{0}(t,k) + np_{a}(t,k) [nr_{th}^j(t,k) + nr_{nh}^j(t,k)]
\]

\[
\quad + np_{a}(t,k) [nr_{th}^j(t,k) + nr_{nh}^j(t,k)] \tag{4.61a}
\]

\[
n\tilde{p}_{a}(t,k) = np_{a}(t,k) nR_{th}(t,k) + np_{a}(t,k) nR_{nh}(t,k) \tag{4.62a}
\]

Intermediate state probabilities for type 2 switch

\[
o\tilde{p}_{0}(t,k) = op_{0}(t,k) + op_{a}(t,k) [or_{th}^j(t,k) + or_{nh}^j(t,k)]
\]

\[
\quad + op_{a}(t,k) [or_{th}^j(t,k) + or_{nh}^j(t,k)] \tag{4.61b}
\]
\[ o\tilde{p}_d(t,k) = o p_d(t,k) o R_d^q(t,k) + o p_d(t,k) o R_d^o(t,k) \]  
\hspace{10cm} (4.62b)

State probabilities at hot switches:

\[ h p_d(t+1,k) = h \tilde{p}_d(t,k) [1 - h q(t,k)] \]  
\hspace{10cm} (4.63)

\[ h p_d(t+1,k) = h \tilde{p}_d(t,k) h q(t,k) \]  
\hspace{10cm} (4.64)

\[ h p_d(t+1,k) = h \tilde{p}_d(t,k) \]  
\hspace{10cm} (4.65)

State probabilities at switches of type 1:

\[ n p_d(t+1,k) = n \tilde{p}_d(t,k) [1 - n q(t,k)] \]  
\hspace{10cm} (4.63a)

\[ n p_d(t+1,k) = n \tilde{p}_d(t,k) n q(t,k) \]  
\hspace{10cm} (4.64a)

\[ n p_d(t+1,k) = n \tilde{p}_d(t,k) \]  
\hspace{10cm} (4.65a)

State probabilities at switches of type 2:

\[ o p_d(t+1,k) = o \tilde{p}_d(t,k) [1 - o q(t,k)] \]  
\hspace{10cm} (4.63b)

\[ o p_d(t+1,k) = o \tilde{p}_d(t,k) o q(t,k) \]  
\hspace{10cm} (4.64b)

\[ o p_d(t+1,k) = o \tilde{p}_d(t,k) \]  
\hspace{10cm} (4.65b)

Transition probabilities at hot switches:

\[ h t_{ak}^1(t,k) = P_{ak}(k) h p_d(t,k+1) [1 - (0.5) P_{ak}(k) h p_d(t,k)] \]

\[ + (0.5) P_{ak}(k)^2 h p_d(t,k) \{ h p_d^o(t,k+1) (0.5) h p_d(t-1,k+1) \} \]
\[ h^1_{\text{in}}(t,k) = P_{\text{in}}(k) \, n_p(t,k+1) \, [1 - (0.5) \, P_{\text{in}}(k) \, h_p(t,k)] \]

\[ + \, (0.5) \, P_{\text{in}}(k)^2 \, h_p(t,k) \, \{ n_p^*(t,k+1) \, (0.5) \, n_p(t-1,k+1) \}
\]

\[ + \, n_p^*(t,k+1)[1 - (0.5) \, n_p(t-1,k+1)] - 2 \, n_p(t,k+1) \]  \hspace{1cm} \hspace{1cm} (4.67)

\[ hR^0_p(t,k) = 1.0 - [h^1_{\text{in}}(t,k) + h^1_{\text{in}}(t,k)] \]  \hspace{1cm} \hspace{1cm} (4.68)

\[ h^1_{\text{in}}(t,k) = \{ P_{\text{in}}(k) \, [h^*_p(t,k+1) \, 0.5 \, h_p(t-1,k+1) \, + \, h^*_p(t,k+1)[1 - 0.5 \, h_p(t-1,k+1)] \} \times \]

\[ [1 - (0.5) \, P_{\text{in}}(k) \, h_p(t,k) - h_p(t,k)] \}

\[ + \, [P_{\text{in}}(k) \, h_p(t,k) \, h^*_p(t,k+1) \, (1 - 0.5P_{\text{in}}(k))] \]  \hspace{1cm} \hspace{1cm} (4.69)

\[ h^1_{\text{in}}(t,k) = \{ P_{\text{in}}(k) \, [n_p^*(t,k+1) \, 0.5 \, n_p(t-1,k+1) \, + \, n_p^*(t,k+1)[1 - 0.5 \, n_p(t-1,k+1)] \} \times \]

\[ [1 - (0.5) \, P_{\text{in}}(k) \, h_p(t,k) - h_p(t,k)] \}

\[ + \, [P_{\text{in}}(k) \, h_p(t,k) \, n_p^*(t,k+1) \, (1 - 0.5P_{\text{in}}(k))] \]  \hspace{1cm} \hspace{1cm} (4.70)

\[ hR^0_p(t,k) = 1.0 - [h^1_{\text{in}}(t,k) + h^1_{\text{in}}(t,k)] \]  \hspace{1cm} \hspace{1cm} (4.71)

Transition probabilities at switches of type 1:

\[ nr^1_p(t,k) = 0.25 \, op_p(t,k+1) \, [1 - (0.25) \, n_p(t,k)] \]

\[ + \, (0.125) \, n_p(t,k) \, \{ op^*_p(t,k+1) \, (0.5) \, op(t-1,k+1) \]
\[ + \ op_\delta^h(t,k+1) [1 - (0.5) \ op(t-1,k+1)] - 2 \ op_\delta(t,k+1) \] (4.66a)

\[ nR_\delta^h(t,k) = 1.0 - [2.0 \ nr_\delta(t,k)] \] (4.67a)

\[ nr_\delta(t,k) = \{ 0.25 [op_\delta^h(t,k+1) 0.5 op(t-1,k+1) + op_\delta^h(t,k+1)[1 - 0.5 op(t-1,k+1)] \times \]

\[ [1 - (0.25) np_\delta(t,k) - np_\delta(t,k)] \} + [0.375 np_\delta(t,k) \ op_\delta^h(t,k+1)] \] (4.68a)

\[ nR_\delta^\gamma(t,k) = 1.0 - [2.0 \ nr_\delta(t,k)] \] (4.69a)

Note that, for a 8 x 8 network, we do not have switches of type 2 in the first and second stages.

Acceptance probabilities for hot switches:

\[ hp_\delta^h(t,k) = hr_\delta^h(t,k) + hr_\delta^\gamma(t,k) \] (4.72)

\[ hp_\delta^g(t,k) = hr_\delta^h(t,k) + hr_\delta^\gamma(t,k) \] (4.73)

\[ hp_\delta(t,k) = \tilde{hp}_\delta(t,k) \] (4.74)

Acceptance probabilities for switches of type 1:

\[ np_\delta^h(t,k) = 2.0 \ nr_\delta^h(t,k) \] (4.72a)

\[ np_\delta^g(t,k) = 2.0 \ nr_\delta^h(t,k) \] (4.73a)

\[ np_\delta(t,k) = \tilde{np}_\delta(t,k) \] (4.74a)
Acceptance probabilities for switches of type 2:

\[ op_x^s(t,k) = 2.0 \, or_x^s(t,k) \]  \hspace{1cm} (4.72b)

\[ op_x^d(t,k) = 2.0 \, or_x^d(t,k) \]  \hspace{1cm} (4.73b)

\[ op_x(t,k) = \tilde{op}_x(t,k) \]  \hspace{1cm} (4.74b)

\[ hp(t,k+1) = 2.0 \left[ hp_x(t,k) \, hr_x^0(t,k) + hp_x(t,k) \, hr_x^1(t,k) \right] \] \hspace{1cm} (4.75)

\[ np(t,k+1) = 2.0 \left[ hp_x(t,k) \, hr_x^0(t,k) + hp_x(t,k) \, hr_x^1(t,k) \right] \] \hspace{1cm} (4.76a)

\[ op(t,k+1) = 2.0 \left[ np_x(t,k) \, nr_x^1(t,k) + np_x(t,k) \, nr_x^0(t,k) \right] \] \hspace{1cm} (4.76b)

\[ hq(t,k+1) = \frac{hp(t,k+1)}{hp_x(t,k+1)} \] \hspace{1cm} (4.77)

\[ nq(t,k+1) = \frac{np(t,k+1)}{np_x(t,k+1)} \] \hspace{1cm} (4.77a)

\[ oq(t,k+1) = \frac{op(t,k+1)}{op_x(t,k+1)} \] \hspace{1cm} (4.77b)

The multiplying factor of 2 in equations 4.75, 4.76a and 4.76b comes from the fact that at stage \( k+1 \), an input link can receive a packet from any of the two buffers of the SE in stage \( k \).
For the final stage, following the same reasoning followed in the derivation of equations 4.59 and 4.60, we get

At the hot switches:

\[ h_{p_{\text{hot}}}(t,n) = p_{\text{hot}}(n) - (0.5) p_{\text{hot}}(n)^2 h_{p_{\text{hot}}}(t,n) - 0.5 p_{\text{hot}}(n)^2 h_{p_{\text{hot}}}(t,n) \]  \hspace{1cm} (4.78)

\[ h_{p_{\text{hot}}}(t,n) = p_{\text{hot}}(n) - (0.5) p_{\text{hot}}(n)^2 h_{p_{\text{hot}}}(t,n) - 0.5 p_{\text{hot}}(n)^2 h_{p_{\text{hot}}}(t,n) \]  \hspace{1cm} (4.79)

\[ h_{p_{\text{hot}}}(t,n) = p_{\text{hot}}(n) [h_{p_{\text{hot}}}(t,n) + h_0(t,n) - 0.5 p_{\text{hot}}(n) h_{p_{\text{hot}}}(t,n)] \]  \hspace{1cm} (4.80)

\[ h_{p_{\text{hot}}}(t,n) = p_{\text{hot}}(n) [h_{p_{\text{hot}}}(t,n) + h_0(t,n) - 0.5 p_{\text{hot}}(n) h_{p_{\text{hot}}}(t,n)] \]  \hspace{1cm} (4.81)

At the switches of type 1 and type 2:

\[ n_{p_{1}}(t,n) = (0.5) - (0.125) (n_{p_{1}}(t,n) + n_{p_{2}}(t,n)) \]  \hspace{1cm} (4.78a)

\[ o_{p_{1}}(t,n) = (0.5) - (0.125) (o_{p_{1}}(t,n) + o_{p_{2}}(t,n)) \]  \hspace{1cm} (4.78b)

\[ n_{p_{2}}(t,n) = (0.25) [n_{p_{2}}(t,n) - 0.75 n_{p_{1}}(t,n)] \]  \hspace{1cm} (4.80a)

\[ o_{p_{2}}(t,n) = (0.25) [o_{p_{2}}(t,n) - 0.75 o_{p_{1}}(t,n)] \]  \hspace{1cm} (4.80b)

The expressions for different probabilities are derived in a manner similar to the one for URM in 4.5; only the appropriate prefix is needed to distinguishing the type of switch. Thus with the initial load \( h_q(1,1) \) specified, the systematic proce-
procedure outlined in section 4.5 can be iteratively extended to yield a stabilized value for the network throughput. As mentioned in the procedure for model 1 in section 4.4.1, the procedure is individually evaluated for a hot switch and all types of non-hot switches in a cycle. The algorithm for the evaluation of the bandwidth for a $N \times N$ omega network is:

- **step 0**: initial load to the network is specified
- **step 1**: calculation of all the state and transition probabilities except $q(.)$ and $\rho(.)$ for the final stage, for all types of switches.
- **step 2**: calculation of all probabilities except $q(.)$ and $\rho(.)$ for the $(n-1)^{th}$ stage, for all types of switches at that stage. $q(n)$ and $\rho(n)$ are calculated now. Repetition of the above step till the first stage is reached
- **step 3**: calculation of all probabilities at first stage given the applied load. This completes a cycle. The above cycle is repeated till a steady state value for the probability of a message at each of the network outputs is obtained. These values are then summed up according to equation 4.33 to give the memory bandwidth of the IN

### 4.5.1.1 Delay

For finding out the delay, the delay for each type of packet received at the output are found out. For finding out the delay for a particular packet type, the delay at each stage is computed. Little's law is used to find out the delay. According to this law, the time spent in a queue multiplied by the arrival rate of
requests gives the length of the queue under steady state. Hence,

\[
\text{Delay at the first stage} = \frac{Pr[\text{Buffer is full}]}{hp(t,k)} = \frac{1 - hp_0(1)}{hp(1)}
\]

\[
\text{Delay at the second stage: for a hot packet} = \frac{1 - hp_0(2)}{hp_0(2)}
\]

\[
\text{for a non-hot packet} = \frac{1 - np_0(2)}{hp_0(2)}
\]

\[
\text{Delay at the third stage: for a hot packet} = \frac{1 - hp_0(3)}{hp_0(3)}
\]

\[
\text{for a packet at non-hot switch of type 1} = \frac{1 - np_0(3)}{np_0(3)}
\]

\[
\text{for a packet at non-hot switch of type 2} = \frac{1 - op_0(3)}{op_0(3)}
\]

Thus total delay is the sum of the delays at the various stages in the network.

The delay experienced is different for different packets and as expected depends on the path a packet follows in the network. Delay for the hot packets is the highest. This is because of increased congestion on hotlinks. The delay for the packet directed towards the non-hot module nearest to the hot module is high as compared to the other non-hot modules. The delay for the memory module, which lies in a set disjoint from the set of modules containing the hot module, is the lowest.
4.5.2 Results

The parameter of interest i.e., the bandwidth was evaluated for various input loads and various hotspot probabilities. The result is shown in Figure 4.10 against request rate for various hotspot probabilities for a network size of 8 x 8. The results show that as the hotspot probability is increased, the bandwidth decreases rapidly. These results are compared with the simulation results in Figure 4.11. The results show appreciable error at high request rates though the error is much less than the one obtained with model 1. This error is due to the fact that model 2 doesn't distinguish between a packet blocked for upper or lower link at a switch. This affects the derivation of the routing probabilities as can be seen from the next section. In the next section a model is presented which takes into account the above mentioned deficiencies and gives very accurate results, even at high loads.
Figure 4.10: RESULTS OBTAINED FROM MODEL 2 FOR AN 8x8 NETWORK FOR DIFFERENT HOTSPOT PROBABILITIES
Figure 4.11: COMPARISON OF THE RESULTS OBTAINED FROM MODEL 2 WITH SIMULATIONS FOR 8x8 NETWORK WITH A HOTSPOT PROBABILITY OF 0.3
4.6 Model 3

The model 2 presented in the previous section though accurate to a higher degree than the model 1, is still not accurate enough for non-uniform reference case, as seen from Figure 4.9. Model 2 in itself, can be applied to any MSIN of arbitrary size. That paradigm is the motivation for the derivation of the following model.

4.6.1 Features

1. In model 2, if a packet was blocked it was assumed that this constituted a whole state in itself which is clearly an approximation because, if a packet has been blocked during a cycle for any particular link, say the upper link, then the probability that it will access the lower link in the next cycle is zero. Hence, there needs to be some information present as regards to the link for which a packet was blocked.

2. If a packet is blocked in any buffer during a cycle, the probability that it will access the same link again in the next cycle is unity. In model 2, it was assumed that a blocked packet requests any output link with a probability of 0.5 for URM and the hot or non-hot output link with a probability $p_{ho}$ or $p_{nho}$ respectively for non-URM.

To incorporate the above features, a "memorized" model is presented below. The model preserves the history of the blocked packet.

IN this model, at any cycle $t$, a buffer at any switch in stage $k$ can be in any one of the following states:

State 0 Buffer is empty
**State n** Buffer contains a new packet which arrived in the phase 2 of the previous cycle i.e., during cycle \((t-1)\)

**State bu** Buffer contains a packet which was previously blocked when it requested the upper output link

**State bl** Buffer contains a packet which was previously blocked when it requested the lower output link

This implies that we have actually added a memory to the model just to hold the information regarding the blocked packet. From the above discussion it is clear that at any given time a blocked packet can be either in state bu or state bl. The new state diagram along with the transition probabilities is as shown in Figure 4.12 for the case of URM. Extension to non-URM is given in section 4.6.1.

The basic strategy of splitting up a clock cycle into two phases is still the same (these two phases are still identical to those mentioned in section 4.1).

From the state diagram, a few observations can be made. A buffer can be in state bu at the end of a cycle if it was in state n at the start of a cycle and the packet was not routed to the output during the cycle or if it was in state bu at the start of a cycle and the packet was blocked again. Similarly, a buffer can be in state bl if it was in state n at the beginning of a cycle and the packet was not routed to the output during the cycle, or if it was in state bl and the packet was blocked again. At the end of phase 1 i.e., at the beginning of phase 2, a buffer can only be in one of the three states -- 0, bu or bl.
Figure 4.12: STATE DIAGRAM FOR MODEL 3
The method of analysis is still the same as in section 4.5. All the different probabilities to be used are defined below following which the derivations of the various probabilities are given for an URM. Then the analysis for the non-URM will be presented in section 4.6.1.

Definitions

\textit{State probabilities}

\[ p_x(t,k) = \Pr[ \text{an input buffer at stage } k \text{ is in state } x \text{ at the beginning of phase 1 of the } t^{\text{th}} \text{ cycle, } x \in \{0,n,bu,bl\}] \]

\textit{Intermediate state probabilities}

\[ \tilde{p}_x(t,k) = \Pr[ \text{an input buffer at stage } k \text{ is in state } x \text{ at the end of phase 1 of the } t^{\text{th}} \text{ cycle, } x \in \{0,nu,bl\}] \]

\textit{Transition probabilities}

\[ r_x^1(t,k) = \Pr[ \text{packet in an input buffer at stage } k \text{ is able to move forward to the upper output link during phase 1 of the } t^{\text{th}} \text{ cycle, given that the buffer was in state } x \text{ at the beginning of phase 1 of the } t^{\text{th}} \text{ cycle, } x \in \{0,n,nu,bl\}] \]

\[ r_x^1(t,k) = \Pr[ \text{packet in an input buffer at stage } k \text{ is able to move} \]


forward to the lower output link during phase 1 of the \( t \)th cycle, given that the buffer was in state \( x \) at the beginning of phase 1 of the \( t \)th cycle, \( x \in \{0, n, bu, bl\} \)

\[
\tau^o_{\mathfrak{R}}(t,k) = Pr[ \text{packet in an input buffer at stage } k \text{ is unable to move forward to the upper output link during phase 1 of the } t \text{th cycle, given that the buffer was in state } x \text{ at the beginning of phase 1 of the } t \text{th cycle, } x \in \{0, n, bu, bl\} ]
\]

\[
\iota^o_{\mathfrak{R}}(t,k) = Pr[ \text{packet in an input buffer at stage } k \text{ is unable to move forward to the lower output link during phase 1 of the } t \text{th cycle, given that the buffer was in state } x \text{ at the beginning of phase 1 of the } t \text{th cycle, } x \in \{0, n, bu, bl\} ]
\]

(Note that in the model 2 in section 4.5, \( R^o_x \) was used to denote the probability of a request being blocked for either output links as opposed to the probability of being blocked for a single output link. Hence, there is change in the notation in this model.)

Acceptance probabilities
\[ p_g(t,k) = \Pr[ \text{that a buffer at stage } k \text{ is able to accept a packet during phase 2 of the } t^{th} \text{ cycle}] \]

\[ p_g^x(t,k) = \Pr[ \text{an input buffer at stage } k \text{ is able to accept a packet during phase 2 of the } t^{th} \text{ cycle, given that the buffer was in state } x \text{ at the beginning of phase 1 of the } t^{th} \text{ cycle}, x \in \{0, n, bu, bl\}] \]

Load

\[ q(t,k) = \Pr[ \text{a packet is offered to a buffer at stage } k \text{ during phase 2 of } t^{th} \text{ cycle}] \]

\[ p_a(t,k) = \Pr[ \text{a packet is received on the upper input of a SE at stage } k \text{ during phase 2 of the } t^{th} \text{ cycle}] \]

\[ p_l(t,k) = \Pr[ \text{a packet is received on the lower input of a SE at stage } k \text{ during phase 2 of the } t^{th} \text{ cycle}] \]

With the above mentioned definitions, expressions are derived for the different state and transition probabilities. The derivations are similar to the ones done for model 1 and model 2 in sections 4.4 and 4.5 respectively. Before going to the actual derivations the following lemmas are presented which will be used in the derivations:
Lemma 1 When there is a new packet at one of the inputs of a SE at stage $k$ and the other input contains a packet blocked for the same output link as the one which is accessed by the new packet, then the next stage buffer should contain a packet. The packet in the next stage buffer should have gone from this stage in the previous cycle or it must have been blocked at that stage for a period exceeding one cycle.

Lemma 2 At the final stage, both the input buffers cannot be in state bu (or bl) simultaneously. It is also not possible that one buffer is in state bu and the other is in state bl. This is because the outputs of the network are assumed to empty instantaneously.

$$\hat{P}_b(t,k) = Pr[\text{a buffer at stage } k \text{ is in state 0 after the phase 1 of } t^{th} \text{ cycle}]$$

$$= Pr[\text{it was in state 0 at the start of phase 1}]$$

$$+ Pr[\text{it was in state n}] \cdot Pr[\text{the new packet was routed to upper link}]$$

$$+ Pr[\text{it was in state n}] \cdot Pr[\text{the new packet was routed to lower link}]$$

$$+ Pr[\text{it was in state bu}] \cdot Pr[\text{the blocked packet was routed to upper link}]$$

$$+ Pr[\text{it was in state bl}] \cdot Pr[\text{the blocked packet was routed to lower link}]$$

$$= p_b(t,k) + p_a(t,k)r^1_{in}(t,k)$$

$$+ p_a(t,k)r^1_{in}(t,k) + p_{in}(t,k)r^1_{in}(t,k)$$
\[ + p_u(t,k) r_u(t,k) \]

(4.82)

\[ \tilde{p}_u(t,k) = Pr[\text{a buffer is in state } u \text{ at the end of phase 1}] \]

\[ = Pr[\text{it was in state } n] \cdot Pr[\text{the new packet was not routed to upper link}] \]

\[ + Pr[\text{it was in state } u] \cdot Pr[\text{the new packet was not routed to upper link}] \]

\[ = p_n(t,k) r_u^0(t,k) + p_u(t,k) r_u^0(t,k) \]

(4.83)

\[ \tilde{p}_d(t,k) = Pr[\text{a buffer is in state } d \text{ at the end of phase 1}] \]

\[ = Pr[\text{it was in state } n] \cdot Pr[\text{the new packet was not routed to lower link}] \]

\[ + Pr[\text{it was in state } d] \cdot Pr[\text{the new packet was not routed to lower link}] \]

\[ = p_n(t,k) r_d^0(t,k) + p_d(t,k) r_d^0(t,k) \]

(4.84)

**State probabilities**

At the end of phase 2 of the \( t \)th cycle or at the beginning of the \((t+1)\)th cycle, the state probabilities are derived as follows:

\[ p_0(t+1,k) = Pr[\text{a buffer is in state } 0 \text{ after phase 2 of } t \text{th cycle}] \]

\[ = Pr[\text{it was in state } 0 \text{ after phase 1 of } t \text{th cycle}] \]
no fresh packets came in phase 2]

\[ P_r = \tilde{p}_\delta(t,k) \cdot [1 - q(t,k)] \]  \hspace{1cm} (4.85)

\[ p_n(t+1,k) = Pr[\text{a buffer is in state } n \text{ after phase 2 of } t^\text{th} \text{ cycle}] \]

\[ = Pr[\text{it was in state 0 after phase 1 of } t^\text{th} \text{ cycle}] \]

\[ = \tilde{p}_\delta(t,k) \cdot q(t,k) \]  \hspace{1cm} (4.86)

\[ p_w(t+1,k) = Pr[\text{a buffer is in state } b_u \text{ after phase 2 of } t^\text{th} \text{ cycle}] \]

\[ = \tilde{p}_w(t,k) \]  \hspace{1cm} (4.87)

\[ p_b(t+1,k) = Pr[\text{a buffer is in state } b_l \text{ after phase 2 of } t^\text{th} \text{ cycle}] \]

\[ = \tilde{p}_w(t,k) \]  \hspace{1cm} (4.88)

\textit{Acceptance probabilities}
From the definitions presented in the early part of the chapter, we get

\[ p^*_{(t,k)} = Pr[\text{packet was routed to either output link in phase 1, given buffer was in state } n] \]

\[ = r^i_{(t,k)} + r^r_{(t,k)} \]  \hspace{1cm} (4.89)

\[ p^u_{(t,k)} = Pr[\text{packet was routed to upper output link in phase 1, given buffer was in state } u] \]

\[ = r^u_{(t,k)} \]  \hspace{1cm} (4.90)

\[ p^l_{(t,k)} = Pr[\text{packet was routed to lower output link in phase 1, given buffer was in state } l] \]

\[ = r^l_{(t,k)} \]  \hspace{1cm} (4.91)

\[ p_{(t,k)} = Pr[\text{packet was empty after phase 1 of } t\text{-th cycle}] \]

\[ = \tilde{p}_{(t,k)} \]  \hspace{1cm} (4.92)

Transition Probabilities

When we find out the transition probabilities for a packet in a buffer of a certain stage, all the possibilities regarding the state of the buffer in the succeeding stage are to be considered. When there is a new packet and a packet blocked for the same output link in the two buffers of a SE, then the probability that the next
stage buffer is able to accept a packet is found as follows.

The probability that it is in state n at time t is equal to the probability that it received a packet from the buffer containing a new packet in the preceding stage during cycle (t-1) which is given by \((0.5)p(t-1,k + 1)\). The probability that buffer at the next stage is not in state n i.e., it is either state bu or bl is given by \((1.0 - (0.5)p(t-1,k + 1))\). Note that, since this is URM, it is equally likely to be in bu or bl. Hence, the probability that the next stage buffer, which is in either state n or bu or bl, can accept a packet is given by

\[
P_{accr} = p^*_n(t,k + 1) (0.5) (p(t-1,k + 1))
+ (p^*_u(t,k + 1) + p^*_d(t,k + 1))\left(\frac{1.0 - (0.5)(p(t-1,k + 1))}{2}\right)
\]

(4.93)

When both the packets in a switch are blocked for the same output link, then the probability that the next stage buffer is able to accept a packet is given by:

\[
P_{inbk} = (p^*_u(t,k + 1) + p^*_d(t,k + 1))\left(\frac{1.0 - (0.5)(p(t-1,k + 1))}{2}\right)
\]

(4.94)

With the above mentioned aspects, the following expressions have been derived: (note that equations 4.95 - 4.102 are valid for all stages except the last stage)

\[
r^1_w(t,k) = (0.5)(p_n(t,k))(0.5)(p_n(t,k + 1))
+ (0.5)(p_u(t,k))(0.5)(p_u(t,k + 1)) + (0.5)(p_d(t,k))(p_d(t,k + 1))
+ (0.5)(p_u(t,k))(0.5)(p_{accr})
+ (0.5)(p_d(t,k))(p_n(t,k + 1))
\]

(4.95)
The explanation for the above expression is as follows: The first term and the second term represents the transition probabilities for the case when the other buffer is in state n, for conflict and conflict-free case. The third term represents the probability for the case when the other buffer is in state 0. The fourth term gives the probability for the case when the other buffer contains a packet blocked for the same link while the fifth term gives the probability for the case when the other buffer is in state bl. The above expression simplifies to give

\[ r^1_{m}(t,k) = (0.375 \ p_a(t,k) + 0.5 \ p_d(t,k) + 0.5 \ p_{ad}(t,k)) \ p_d(t,k+1) \]

\[ + 0.25 \ p_{ad}(t,k) \ p_{next} \]  \hspace{1cm} (4.96)

Similarly one can derive the expressions for the other transition probabilites as:

\[ r^1_{m}(t,k) = ((0.375 \ p_a(t,k)) + (0.5 \ p_d(t,k)) + (0.5 \ p_{ad}(t,k))) \ p_d(t,k+1) \]

\[ + (0.25 \ p_{ad}(t,k) \ p_{next}) \]  \hspace{1cm} (4.97)

From the state diagram of Figure 1.10, we have

\[ r^1_{m}(t,k) + r^1_{a}(t,k) + r^0_{m}(t,k) + r^0_{a}(t,k) = 1 \]  \hspace{1cm} (4.97a)

Since for URM we have \( r^0_{m}(t,k) = r^0_{a}(t,k) \) and \( r^1_{m}(t,k) = r^1_{a}(t,k) \) we get

\[ r^0_{m}(t,k) = 0.5 - r^1_{m}(t,k) \]  \hspace{1cm} (4.98)

\[ r^0_{a}(t,k) = 0.5 - r^1_{a}(t,k) \]  \hspace{1cm} (4.99)
\[ r^1_{ln}(t,k) = (p_\Delta(t,k)(0.5))(p_{\text{new}}) \]

\[ + (p_a(t,k)(0.5))(p_{\text{new}}) + (p_\delta(t,k))(p_{\text{new}}) \]

\[ + (p_{\text{new}}(t,k))(0.5)(p_{\text{new}}) \]

\[ + (p_\Delta(t,k))(p_{\text{new}}) \]

which simplifies to

\[ r^1_{ln}(t,k) = p_{\text{new}}((0.75p_a(t,k)) + (p_\delta(t,k))) \]

\[ + p_{\text{new}}((0.5p_{\text{new}}(t,k)) + (p_\Delta(t,k))) \]

(4.100)

Similarly we can get the other transition probabilities as

\[ r^1_{\Delta}(t,k) = p_{\text{new}}((0.75p_a(t,k)) + (p_\delta(t,k))) \]

\[ + p_{\text{new}}((0.5p_{\Delta}(t,k)) + (p_{\text{new}}(t,k))) \]

(4.101)

\[ r^0_{ln}(t,k) = 1.0 - r^1_{ln}(t,k) \]

(4.101a)

\[ r^0_{\Delta}(t,k) = 1.0 - r^1_{\Delta}(t,k) \]

(4.102)

At the final stage, the transition probabilities are slightly different. Since we do not have blocking due to next stage being full, the expressions for these probabilities at the final stage are obtained by making the next stage acceptance probabilities as unity in equations 4.95 - 4.102. For example from equation 4.95, by removing the next stage acceptance probabilities we get
\[ r_{1w}(t,n) = (0.5)(p_\alpha(t,n)(0.5)) \]

\[ + (0.5)(p_\alpha(t,n)(0.5))(0.5) + (0.5)(p_\delta(t,n)) \]

\[ + (0.5)(p_{w0}(t,n))(0.5) \]

\[ + (0.5)(p_\delta(t,n)) \]

which can be simplified by noting that the sum of all state probabilities at any stage and at any time is unity, to yield

\[ r_{1w}^1(t,n) = 0.5 - 0.125p_\alpha(t,n) - 0.25p_{w0}(t,n) \quad (4.103) \]

\[ r_{2w}^1(t,n) = 0.5 - 0.125p_\alpha(t,n) - 0.25p_{\delta}(t,n) \quad (4.104) \]

\[ r_{1w}^0(t,n) = 0.5 - r_{1w}^1(t,n) \quad (4.105) \]

\[ r_{2w}^0(t,n) = 0.5 - r_{2w}^1(t,n) \quad (4.106) \]

\[ r_{1w}^1(t,n) = 1.0 - 0.25p_\alpha(t,n) \quad (4.107) \]

\[ r_{2w}^1(t,n) = 1.0 - 0.25p_\alpha(t,n) \quad (4.108) \]

\[ r_{1w}^0(t,n) = 1.0 - r_{1w}^1(t,n) \quad (4.109) \]

\[ r_{2w}^0(t,n) = 1.0 - r_{2w}^1(t,n) \quad (4.110) \]
The load applied to any stage $k$ can be found after finding the probability that a packet is received on an input of a SE at stage $k$ from the upper or lower input link of the SE of stage $(k-1)$. This can happen if the buffer at stage $(k-1)$ was in state $n$ and it routed the packet or if it was in state $bu$ or $bl$ and it routed the packet. Also, the buffer at stage $k$ can receive a packet from either inputs of the SE of stage $(k-1)$ feeding it. Hence, we have

$$p_o(t,k) = 2(p_o(t,k-1)r_u(t,k-1) + p_o(t,k-1)r_l(t,k-1)) \quad (4.111)$$

and we have

$$p_i(t,k) = 2(p_o(t,k-1)r_u(t,k-1) + p_o(t,k-1)r_l(t,k-1)) \quad (4.112)$$

For URM $p_o(t,k)$ is equal to $p_i(t,k)$. The multiplying factor of 2 comes from the fact that the packet can come from either inputs of the SE in stage $(k-1)$.

The probability that a packet is offered to any stage is the ratio of the probability that a packet is received on a particular link to the probability that the buffer in that link accepted the packet. Thus

$$q(t,k) = \left[ \frac{p_o(t,k)}{p_i(t,k)} \right] \quad (4.113)$$

Thus with the expressions derived the average bandwidth of the system can be evaluated following the same algorithm outlined for model 2 in section 4.5. The delay can also be found in a similar manner.
4.6.1.1 Results For URM for model 1, model 2 and model 3

The above mentioned model for URM is highly accurate as can be seen from Figure 4.13 which shows the comparison of the results obtained for the three models for an 8 x 8 omega network. Model 3 is the most accurate and is more accurate than the existing models for URM in literature.
Figure 4.13: COMPARISON OF RESULTS OBTAINED FROM THE THREE MODELS WITH SIMULATION FOR A 8x8 OMEGA NETWORK UNDER URM.
4.6.2 Extension of Model 3 to non-URM

The extension of these equations to non-URM case is also similar to that done for the model 2 in section 4.5.1. For a hot switch, the different states are:

**State 'V'** Buffer is empty

**State 'w'** Buffer contains a new packet which arrived in the phase 2 of the previous cycle (from a hot switch)

**State 'bh'** Buffer contains a packet blocked for the hot output link

**State 'bnh'** Buffer contains a packet blocked for the non-hot output link

All other definitions of different probabilities also apply to the non-URM case. For the sake of notation the probabilities at the different types of switches, the same convention as that used in model 1 is used. There is a slight variation in the derivation of acceptance probabilities for a hot switch which is detailed below. The other expressions can be derived following the same methodology used in obtaining the results for model 2 under non-URM.

The probability of a hot switch at stage \((k+1)\) accepting a packet depends on whether it successfully routed the already existing packet (if it was in a new state or one of the blocked states) or it was in state 0 at the start of the cycle.

The probability that the packet is successfully routed is derived as follows:

At a hot switch in stage \((k+1)\), a packet is blocked for a hot output link either due to a conflict (probability equals \(P_{bh}(k+1)^2\)) or it is blocked due to stage \((k+2)\) not able to accept it (probability equals \((1.0 - hp_s(t,k+2))\)). Similarly, for a nonhot output link of a hot switch, the probability that a packet is blocked due to a conflict or due to the next stage being unable to accept it is \(P_{bnh}(k+1)^3\) or \((1.0 - np_s(t,k+2))\) respectively. Consequently, we define three
terms as:

\[ v_j(t,k+1) = 1 - hp_j(t,k+2) + P_{\text{ab}}(k+1)^2 \]  \hspace{1cm} (4.114)

\[ v_i(t,k+1) = 1 - np_i(t,k+2) + P_{\text{ab}}(k+1)^2 \]  \hspace{1cm} (4.115)

\[ v_i(t,k+1) = v_i(t,k+1) + v_i(t,k+1) \]  \hspace{1cm} (4.116)

Hence the probability that stage \((k+1)\) it is able to accept a new packet when the other packet is blocked for the same output link, is given by

\[ hp_{\text{max}}(t,k+1) = hp_j(t,k+1)0.5hp(t-1,k+1) \]

\[ + hp_j(t,k+1)(1 - P_{\text{ab}}(k)hp(t-1,k+1))(\frac{v_i(t,k+1)}{v_j(t,k+1)}) \]

\[ + hp_j(t,k+1)(1 - P_{\text{ab}}(k)hp(t-1,k+1))(\frac{v_j(t,k+1)}{v_i(t,k+1)}) \]  \hspace{1cm} (4.117)

The quotient \((\frac{v_i(t,k+1)}{v_j(t,k+1)})\) gives the proportion of the blocked probability to account for the blocking for the hot link, while the quotient \((\frac{v_j(t,k+1)}{v_i(t,k+1)})\) gives the corresponding proportion for the non-hot link.

When both the buffers at stage \(k\) are in a blocked state, the probability reduces to

\[ hp_{\text{max}}(t,k+1) = hp_j(t,k+1)(1 - P_{\text{ab}}(k)hp(t-1,k+1))(\frac{v_i(t,k+1)}{v_j(t,k+1)}) \]
\[ + \, h_{p_1}(t,k+1)(1 - P_{1}(k)h_{p_1(t-1,k+1)))(\frac{v_p(t,k+1)}{v_s(t,k+1)}) \]  

(4.118)

The following equations are derived in a manner similar to the derivation of the expressions for hot switches in section 4.5.1 for model 2.

Intermediate state probabilities

\[ \tilde{h}_{p_1}(t,k) = h_{p_1}(t,k) + h_{p_1}(t,k)h_{r_1}(t,k) \]

\[ + h_{p_1}(t,k)h_{r_2}(t,k) + h_{p_1}(t,k)h_{r_3}(t,k) \]

\[ + h_{p_1}(t,k)h_{r_4}(t,k) \]  

(4.119)

\[ \tilde{h}_{p_2}(t,k) = h_{p_2}(t,k)h_{r_1}(t,k) + h_{p_2}(t,k)h_{r_3}(t,k) \]

(4.120)

\[ \tilde{h}_{p_3}(t,k) = h_{p_3}(t,k)h_{r_1}(t,k) + h_{p_3}(t,k)h_{r_3}(t,k) \]  

(4.121)

State Probabilities

\[ h_{p_1}(t+1,k) = \tilde{h}_{p_1}(t,k) \cdot [1 - h_{q}(t,k)] \]  

(4.122)

\[ h_{p_2}(t+1,k) = \tilde{h}_{p_2}(t,k) \cdot h_{q}(t,k) \]  

(4.123)

\[ h_{p_3}(t+1,k) = \tilde{h}_{p_3}(t,k) \]  

(4.124)
\[ h_{p_{\text{m}}}^{(t+1)k} = \tilde{h}_{p_{\text{m}}}^{(t)k} \]  \hspace{1cm} (4.124)

Acceptance probabilities

\[ h_{p_{\text{a}}}^{(t)k} = hr_{\text{ml}}^{(t)k} + hr_{\text{ml}}^{(t+1)k} \]  \hspace{1cm} (4.125)

\[ h_{p_{\text{a}}}^{(t)k} = hr_{\text{ml}}^{(t)k} \]  \hspace{1cm} (4.126)

\[ h_{p_{\text{a}}}^{(t)k} = hr_{\text{ml}}^{(t)k} \]  \hspace{1cm} (4.127)

\[ h_{p_{\text{a}}}^{(t)k} = h_{p_{\text{d}}}^{(t)k} \]  \hspace{1cm} (4.128)

From equations 4.114 - 4.118, we get the transition probabilities as:

\[ hr_{\text{ml}}^{(t)k} = P_{\text{m}}(k) h_{p_{\text{m}}}^{(t)k + 1} (1 - 0.5 P_{\text{m}}(k)) h_{p_{\text{m}}}^{(t)k} + h_{p_{\text{d}}}^{(t)k} + h_{p_{\text{ml}}}^{(t)k} \]  \hspace{1cm} (4.129)

\[ hr_{\text{ml}}^{(t)k} = P_{\text{ml}}(k) n_{p_{\text{ml}}}^{(t)k + 1} (1 - 0.5 P_{\text{ml}}(k)) h_{p_{\text{ml}}}^{(t)k} + h_{p_{\text{d}}}^{(t)k} + h_{p_{\text{ml}}}^{(t)k} \]
\[ + 0.5 \ P_{\text{sh}}(k) \ hp_{\text{sh}}(t,k) \ np_{\text{next}}(t,k+1) \]  \hspace{1cm} (4.130)

where \( np_{\text{next}}(t,k+1) \) is the probability of acceptance of a non-hot switch in stage \( k+1 \), which is given by

\[ np_{\text{next}}(t,k+1) = np_{2}^{*}(t,k+1) \ 0.5 \ np(t-1,k+1) \]

\[ + (np_{2}^{*}(t,k+1) + np_{2}^{*}(t,k+1) (1 - (0.5np(t-1,k+1)))) \frac{1}{2} \]  \hspace{1cm} (4.131)

\[ hr_{\text{sh}}^{1}(t,k) = hp_{\text{next}}(t,k+1) ((0.5hp_{\text{sh}}(t,k)) + hp_{\text{sh}}(t,k)) \]

\[ + hp_{\text{next}}(t,k+1) ((0.5hp_{\text{sh}}(t,k)) + hp_{\text{sh}}(t,k)) \]  \hspace{1cm} (4.132)

\[ hr_{\text{sh}}^{1}(t,k) = np_{\text{next}}(hp_{\text{sh}}(t,k)(1 - (0.5)P_{\text{sh}}(k))) + hp_{\text{sh}}(t,k)) \]

\[ + np_{\text{next}}(t,k+1) ((0.5hp_{\text{sh}}(t,k)) + hp_{\text{sh}}(t,k)) \]  \hspace{1cm} (4.133)

where

\[ np_{\text{next}}(t,k+1) = np_{\text{next}} - (np_{2}^{*}(t,k+1) \ 0.5 \ np(t-1,k+1)) \]

At the final stage, we do not have blocking due to a the next stage being full; hence, in this case, the equations for transition probabilities reduce to

\[ hr_{\text{sh}}^{1}(t,k) = P_{\text{sh}}(k) - (0.5P_{\text{sh}}(k)^{2}hp_{\text{sh}}(t,k)) - (0.5P_{\text{sh}}(k)hp_{\text{sh}}(t,k)) \]  \hspace{1cm} (4.134)

\[ hr_{\text{sh}}^{1}(t,k) = P_{\text{sh}}(k) - (0.5P_{\text{sh}}(k)^{2}hp_{\text{sh}}(t,k)) - (0.5P_{\text{sh}}(k)hp_{\text{sh}}(t,k)) \]  \hspace{1cm} (4.135)

\[ hr_{\text{sh}}^{2}(t,k) = P_{\text{sh}}(k) - hr_{\text{sh}}^{1}(t,k) \]  \hspace{1cm} (4.136)
\begin{align}
hr_a^0(t,k) &= P_a(k) - hr_a^1(t,k) \\
hr_1^1(t,k) &= 1.0 - (0.5P_a(k)hp_a(t,k)) \\
hr_1^1(t,k) &= 1.0 - (0.5P_a(k)hp_a(t,k)) \\
hr_1^0(t,k) &= 1.0 - hr_1^1(t,k) \\
hr_1^0(t,k) &= 1.0 - hr_1^1(t,k)
\end{align}

Thus the throughput on the input links can be found out as

\begin{align}
hr(t,k) &= 2(hp_a(t,k-1)hr_a^1(t,k-1) + hp_a(t,k-1)hr_a^1(t,k-1)) \\
hp(t,k) &= 2(hp_a(t,k-1)hr_a^1(t,k-1) + hp_a(t,k-1)hr_a^1(t,k-1))
\end{align}

The load offered can be found as

\begin{align}
hq(t,k) &= \left( \frac{hp(t,k)}{hp_a(t,k)} \right)
\end{align}

The equations for the nonhot switch are similar to those for the non-hot switches in model 2 in section 4.5.1. The algorithm for the evaluation of the bandwidth and the network delay is identical to the one used in model 2.
4.6.3 Results

The parameters evaluated for single buffered networks were the bandwidth and the delay. Figure 4.14 shows the variation of bandwidth with request rate for different hotspot probabilities for a network size of 8. This plot demonstrates the effect of increase in hotspot probability on the performance of single buffered omega network. The degradation is higher at higher loads because of increase in probability of tree saturation. The accuracy of the analytical model developed can be seen from Table 4.1 where the percentage error between the results of model 3 and the simulation results is listed for various hotspot probabilities. As is evident, the percentage error is reasonably small. Time delay is plotted against request rate for different hotspot probabilities for a 8 x 8 network in Figure 4.15. The delay increases rapidly because of hotspots and is worse even for moderate hotspot probabilities.

Simulation results are presented for other network sizes. The analytical model is valid for all network sizes; because of the complexity involved in identifying each type of switch in the large sizes and because the simulation results and the analytical results are in close agreement, only simulation results are given for larger network sizes. Plots for network sizes of 16, 32, and 64 are shown in Figure 4.16, Figure 4.17, and Figure 4.18 respectively. Similarly delay for other network sizes of 16, 32, and 64 is shown in Figure 4.19, Figure 4.20, and Figure 4.21 respectively.

The effect on the system performance because of the increase in the network size was also studied. Figure 4.22 shows the variation of bandwidth with the network size for different hotspot probabilities with an initial load of 1.
Figure 4.14: BANDWIDTH vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 8 x 8 NETWORK
<table>
<thead>
<tr>
<th>Request Rate</th>
<th>( p_n = 0.2 )</th>
<th>( p_n = 0.4 )</th>
<th>( p_n = 0.6 )</th>
<th>( p_n = 0.8 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>-0.13</td>
<td>0.05</td>
<td>1.09</td>
<td>0.88</td>
</tr>
<tr>
<td>0.4</td>
<td>-0.38</td>
<td>0.17</td>
<td>0.13</td>
<td>0.17</td>
</tr>
<tr>
<td>0.6</td>
<td>-0.92</td>
<td>0.83</td>
<td>0.82</td>
<td>0.26</td>
</tr>
<tr>
<td>0.8</td>
<td>-0.92</td>
<td>1.79</td>
<td>1.13</td>
<td>0.31</td>
</tr>
<tr>
<td>1.0</td>
<td>-0.99</td>
<td>1.82</td>
<td>1.45</td>
<td>0.63</td>
</tr>
</tbody>
</table>

Table 4.1: Percentage Error Between Results of Model 3 and Simulation Results for \( n = 8 \).

(Plots are given in Figure 4.23, Figure 4.24, Figure 4.25, and Figure 4.26 for a
Figure 4.15: DELAY (HOT PACKET) vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 8x8 NETWORK
Figure 4.16: BANDWIDTH vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 16 x 16 NETWORK
Figure 4.17: BANDWIDTH vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 32 x 32 NETWORK
Figure 4.18: BANDWIDTH vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 64 x 64 NETWORK
Figure 4.19: DELAY (HOT PACKET) vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 16 x 16 NETWORK
Figure 4.20: DELAY (HOT PACKET) vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 32 x 32 NETWORK
Figure 4.21: DELAY (HOT PACKET) vs REQUEST RATE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED 64 x 64 NETWORK
load of 0.8, 0.6, 0.4 and 0.2). The increase in the network size is offset by the percentage degradation in the bandwidth as is evident from the figures.

A comparison of the results obtained from the three models with those obtained from simulation is performed in Figure 4.27 for an initial load of unity and hotspot probability of 0.3. The graph shows the inaccuracies involved in the loose modelling employed in model 1. The model 2 though accurate to a high degree than model 1, is still inaccurate. Model 3 is highly accurate and demonstrates the effectiveness obtained by modelling the interdependence of the states of buffers in consecutive cycles and in adjacent stages.
Figure 4.22: BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED NETWORK WITH AN INITIAL LOAD OF 1.0
Figure 4.23: BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED NETWORK WITH AN INITIAL LOAD OF 0.8
Figure 4.24: BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED NETWORK WITH AN INITIAL LOAD OF 0.6
Figure 4.25: BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED NETWORK WITH AN INITIAL LOAD OF 0.4
Figure 4.26: BANDWIDTH vs NETWORK SIZE FOR DIFFERENT HOTSPOT PROBABILITIES FOR A SINGLE BUFFERED NETWORK WITH AN INITIAL LOAD OF 0.2
Figure 4.27: COMPARISON OF RESULTS OF THE THREE MODELS WITH SIMULATIONS FOR N = 8 WITH FULL LOAD AND HOTSPOT PROBABILITY OF 0.3
4.6.3.1 Prioritized Resolution of Conflicts:

As mentioned in section 2.2.4.1, two types of conflict resolution strategies have been studied in simulations. The resolution of conflicts by randomly selecting a packet (when there is a conflict) has already been described and the results presented. The results of other type of conflict resolution, wherein blocked packets have precedence over fresh packets, are given below.

By prioritized resolution, it is meant that when a new packet is in conflict with a buffered packet for an output link, the buffered packet always wins the conflict and goes through to the next stage if the next stage buffer is able to accept it. Intuitively this implies that the bandwidth should increase because, by clearing the packets in the immediate cycle, the saturation of the tree is delayed. Also the congestion of the traffic is reduced. The results confirm the above intuition. The bandwidth of an 8 x 8 network, with both types of conflict resolutions, is plot in Figure 4.28 against the request rate for various hotspot probabilities. A comparison of the bandwidth obtained with the two conflict resolution schemes for hotspot probabilities of 0.2 and 0.3 (for a 8 x 8 omega network) is given in Figure 4.29. The improvement in the bandwidth is obvious.
Figure 4.28: RESULTS FOR PRIORITISED RESOLUTION OF CONFLICTS N = 8 WITH FULL LOAD FOR DIFFERENT HOTSPOT PROBABILITIES.
Figure 4.29: COMPARISON OF THE RESULTS OBTAINED WITH THE TWO CONFLICT RESOLUTION SCHEMES FOR N = 8
4.7 Summary

In this chapter, single buffered omega networks were analyzed. An introduction to the concept of tree saturation was given in section 4.1. The three models developed were explained in sections 4.4, 4.5 and 4.6 respectively. In each of the three models, the analysis was first done for URM and then extended to non-URM. Results were compared in section 4.6.2.

In the next chapter simulation results are presented for multibuffered networks under non-URM.
CHAPTER V
PERFORMANCE ANALYSIS OF MULTI-BUFFERED NETWORKS

5.1 Introduction
In the last chapter, extensive analytical and simulation results were presented for single-buffered networks. Three analytical models were presented which had varying degree of accuracy. These models can be extended to multibuffered networks. This chapter presents the simulation results for the multibuffered networks which will form a guide to study in this field. In section 5.2, the simulation methodology is described followed by an interpretation of the results in section 5.3.

5.2 Simulation methodology
The simulation of multi-buffered networks was carried out following the same general pattern as for the unbuffered and single-buffered networks, explained in sections 3.4 and 4.2.5 respectively. To facilitate fast processing, a single array was chosen as the buffer. Again, the FIFO buffer policy was used. Requests were generated using the same policy outlined in section 3.4 for unbuffered networks. In each cycle, if the buffer at the first stage was not occupied to full capacity, then the request was accepted into the network. While implementing the switching operations, the queue lengths of the next stage buffer were
checked to determine the availability of free space. Also, the queue lengths at each buffer were taken as the criteria for determining the validity of packets at each stage. A separate array was used to hold the queue length at each link of each stage. The conflict resolution was random. After a complete cycle, the number of valid requests at the output of the network were counted to give the bandwidth in that cycle. The whole procedure was iterated over 50,000 cycles to give a steady estimate of the values. Modelling of delay was also done in a manner similar to the one done for a single-buffered network in section 4.2.5.

5.3 Results
Simulations were performed for various queue sizes ranging from 2 to 8. All the buffers in the network had the same length. Parameters evaluated were bandwidth and delay. Plots of these parameters are presented against request rates, hotspot probabilities etc., for various buffer sizes. Figure 5.1 shows the plot of the bandwidth versus request rate for various buffer sizes for a network size of 64 and hotspot probability of 0.05. It is evident from the graph that just increasing the buffer size doesn't improve the performance. After a certain point, increase in buffer size doesn't increase the bandwidth. This is because, as the buffer size is increased and as the hotspot probability is increased, there are not enough packets coming in the buffers of the non-hot switches; but the buffers at the hot switches are filled up, leading to tree saturation. This underutilization of the buffer capacity at non-hot switches leads to a reduction in the bandwidth. For hotspot probabilities of 0.1, 0.15, 0.2 and 0.25, the plots are shown in Figure 5.2, Figure 5.3, Figure 5.4, and Figure 5.5 respectively.
Figure 5.1: BANDWIDTH vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.05 FOR DIFFERENT BUFFER SIZES
Figure 5.2: BANDWIDTH vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.10 FOR DIFFERENT BUFFER SIZES
Figure 5.3: BANDWIDTH vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.15 FOR DIFFERENT BUFFER SIZES
Figure 5.4: BANDWIDTH vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.20 FOR DIFFERENT BUFFER SIZES
Figure 5.5: BANDWIDTH vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.25 FOR DIFFERENT BUFFER SIZES
The plot of delay for a request directed towards the hot module versus request rate is shown in Figure 5.6 for a hotspot probability of 0.05 for different buffer sizes. As the buffer size increases, the delay increases at higher request rates. This is due to increased probability of buffer being full at higher request rates leading to tree saturation.

A more informative plot is shown in Figure 5.7. This shows the effect of hotspots on the performance of an 8 x 8 omega network with a buffer size of 4. As the hotspot probability increases, the bandwidth decreases rapidly even with a higher buffer size. Plots for network sizes of 16, 32, and 64 are shown in Figure 5.8, Figure 5.9, and Figure 5.10 respectively.

A comparison of the delay for requests for different memory modules is also given in Figure 5.11, which shows the same for an 8 x 8 network having a buffer capacity of 4. The hotspot probability is 0.3. The figure shows the plot against increasing request rate. It is evident from the plot that even the requests that are oriented towards the non-hot memories are affected. The delay for the module connected to the same switch as the hot module, is very high as compared to other non-hot modules.

Given a buffer size, the optimum network size i.e., the network size that gives a cost effective performance, is a significant factor. A plot of bandwidth versus increase in network size is thus made to provide information regarding the same. This plot is shown in Figure 5.12 for a hotspot probability of 0.15 and for a request rate of unity. It is evident from the plot that increase in the network size will be offset by lack of improvement in the bandwidth of even for low hotspot probabilities.
Figure 5.6: DELAY (HOT PACKET) vs REQUEST RATE FOR A 64 x 64 OMEGA NETWORK WITH A HOTSPOT PROBABILITY OF 0.05 FOR DIFFERENT BUFFER SIZES
Figure 5.7: BANDWIDTH vs REQUEST RATE FOR A 4-BUFFERED 8 x 8 OMEGA NETWORK FOR DIFFERENT HOTSPOT PROBABILITIES.
Figure 5.8: BANDWIDTH vs REQUEST RATE FOR A 4-BUFFERED 16 x 16 OMEGA NETWORK FOR DIFFERENT HOTSPOT PROBABILITIES.
Figure 5.9: BANDWIDTH vs REQUEST RATE FOR A 4-BUFFERED 32 x 32 OMEGA NETWORK FOR DIFFERENT HOTSPOT PROBABILITIES.
Figure 5.10: BANDWIDTH vs REQUEST RATE FOR A 4-BUFFERED 64 x 64 OMEGA NETWORK FOR DIFFERENT HOTSPOT PROBABILITIES.
Figure 5.11: DELAY FOR DIFFERENT MEMORY MODULES vs REQUEST RATE FOR A 4-BUFFERED 8x8 OMEGA NETWORK; HOTSPOT PROBABILITY = 0.3.
Figure 5.12: BANDWIDTH vs NETWORK SIZE FOR DIFFERENT BUFFER SIZES. HOTSPOT PROBABILITY = 0.15 AND REQUEST RATE = 1.0.
CHAPTER VI
CONCLUSION

The problem of hotspot contention in shared memory multiprocessor systems is marked enough to be not overlooked while evaluating them. Previous approaches to this problem have been based on extensive simulations — without a basic analytical model which follows the assumptions mentioned earlier.

In this thesis, analytical models have been developed for the performance evaluation of a MSIN under a single hotspot. Section 6.1 gives a summary of the thesis work besides presenting the results in a terse form. In section 6.2, few suggestions are made regarding the extension of this work in future and regrading the analysis of a MSIN in general.

6.1 Summary of the thesis work

In chapter 1, an introduction has been given to the concepts of parallel processing and interconnection networks. A general classification of INs is also given. A major potential pitfall in the performance of a MSIN - a hotspot, has been introduced. A report of the work done in the field of INs of all types has been presented. Significant results obtained for crossbar and multiple-bus INs are listed. The primary purpose of this is to provide a comprehensive literature survey and to aid in comparing the complexities of different INs. Basic performance
measurement criteria have been defined. An insight into the objective of this thesis work has been given.

Chapter 2 gives the details of the construction and operation of MSINs, focussing on delta and omega networks. The various aspects of message passing in the INs have been described. Different switching strategies, routing strategies and conflict resolution schemes have been defined. A survey of the results reported in the literature for unbuffered and buffered MSINs under URM and non-URM has also been presented. The research surveyed indicates an acute shortage of accurate results for a study of MSIn under non-uniform memory references.

Chapter 3 provides a detailed performance analysis of an unbuffered MSIN. At the beginning of the chapter, a summary of the causes and effects of hotspots has been presented. A synchronous, packet-switched $N \times N$ omega network has been used to model a MSIN under a single hotspot. Other assumptions used have been listed in the chapter. A description of the peculiar behavior exhibited by a MSIN under the presence of a single hotspot has also been given. Based on the theorems and lemmas formulated, expressions are developed for memory bandwidth and blocking probability. The results obtained are validated against simulation results. The simulation methodology used is also explained in detail. The results indicate a marked degradation in the bandwidth obtained under the presence of a hotspot. Blocking probability is increased as compared to URM. Thus, the presence of a hotspot affects the network as a whole and cannot be overlooked in its performance evaluation.
Chapter 4 deals with the performance evaluation of single buffered MSIN's. Buffers of capacity one have been assumed at the inputs of the switches. FIFO buffer policy has been used. A clock cycle is split into two clock phases — one for forwarding the packets and the other for receiving new ones. Motivated by the work presented in the literature, three models are presented for performance evaluation of the buffered omega network under hotspot conditions.

In model 1, the buffer can be in any one of the two states '0' or '1'. This model does not take into account the dependencies between states of buffers of adjacent stages and states of buffers of a stage in consecutive cycles. A recursive method is presented to evaluate the various state and transition probabilities. Parameters evaluated include the bandwidth and the network delay. The model is developed for uniform references and then extended to non-uniform references. Simulations are done and the results are compared to that of analytical model developed. The model developed lacks accuracy at higher loads due to the dependencies mentioned above. A separate simulation study was done to verify this and the results were found in close agreement with the analytical ones.

A blocked state 'b' is introduced in model 2. Again the intermediate state, initial state and state transition probabilities are iteratively calculated until a steady state value is obtained for the bandwidth. First the model is explained under URM, then the extension to non-uniform references is presented. This model takes into account the dependency between states of buffers in adjacent stages. Hence the results obtained are closer to simulation results., but still lack a fair degree of accuracy at higher loads. The model developed is accurate for uniform reference model. Based on this model a refined model is developed.
The refined model, model 3, considers a switch to be in any one of the four states -- either empty or containing a new packet or containing packet blocked for the upper link or containing packet blocked for the lower link. A highly accurate model is developed for URM based on the above model and then this model is extended for non-uniform references. The results are still highly accurate at all loads. The degradation in the bandwidth is obvious. There is very small improvement in the bandwidth because of buffering. Simulation results are presented for both types of conflict resolution schemes -- random selection and priority to blocked packets. Results indicate a slight improvement in the bandwidth when the priority is given to the blocked requests. Delay was found to be the highest for the blocked packet. A collection of graphs is presented as an aid for further work. These show the plots of network bandwidth and delay for varying degrees of hotspot probabilities and request rates.

In this thesis, only simulations were performed for multibuffered networks. Chapter 5 deals with the performance of multibuffered networks for various network sizes, hotspot probabilities, request rates and buffer size. The results indicate a degradation in bandwidth as the hotspot probability is increased. Also, increasing the buffer size does not solve the problem of low bandwidth in an omega network. Delay experienced by packets of all types is presented. A comparison of effects of various buffer sizes on bandwidth is also given.

As a conclusion, it is evident that there is a substantial degradation in the performance of the MSIN under the presence of a hotspot.
6.2 Suggestions for future work:

(1) The models presented in the thesis for unbuffered and buffered networks are for a switch size of 2 x 2. A possible extension of the model would be to incorporate any general switch size. For such an extension, at each switch, for each link the set of memory modules that can be accessed need to be identified. Then the effective hotspot probabilities at each stage can be determined and used to develop the model.

(2) The thesis work assumes a single hotspot. This work can also be extended to the case where there are multiple hotspots.

(3) Throughout the thesis, a single hotspot was assumed. The extension of the models developed to incorporate a "warm spot" can be done. A "warm spot" is a module which is requested with a probability that is greater than the probability of requesting a non-hot module, but less than the probability of requesting the hot module. This will create sub trees in the network which can be identified and analyzed individually.

(4) In the analytical study of the buffered networks, the model assumed a buffer capacity of one. The claim that this model holds for all the buffer sizes can be investigated. In doing so, one will need to identify all possible states of all statistically different buffers in the network.

(5) The models developed for buffered networks can be extended to the case where priority is given to blocked requests. The extension can be verified with the simulation results presented in this thesis.
(6) Another possible variation of the models developed for buffered networks is to place an output buffer queue instead of one at the input and analyze the network under hotspots.
REFERENCES


UNBUF.C

/*****************************/
/* PROGRAM TO SIMULATE A UNBUFFERED MULTISTAGE OMEGA NETWORK */
/* WHEN THE CONFLICT RESOLUTION IS RANDOM. THIS PROGRAM *****/
/* COMPUTES MEMORY BANDWIDTH AND THE PROBABILITY OF BLOCKING **/
/***** DESTINATION TAG ROUTING ASSUMED ***********************
/*************************************************************/
/* greq[][] contains the generated requests in any cycle ****/
/* while efreq[][] contains the effective requests at the ****/
/* output of the previous stage. the first subscript of the ***/
/* array refers to the link number while the second subscript */
/* refers to the data and the associated validity tag; ***********/
/* a valid packet is identified by tag bit of 1 while invalid*/
/* packet has a tag bit of 0;bw[] and ctrl[] hold the bandwidth*/
/* and the number of requests blocked in each iteration *******
/*****************************/
/* COMPONENTS OF THE PROGRAM : *******
/* Main routine : controls the flow of the program; *******/
/* count() : to count the number of requests to a memory**/
/* switex() : to implement the switching operation */
/* rotate() : to find out destination bit at each stage */
/* shuf() : to implement the shuffle operation */
/* conflict : to resolve the conflicts randomly */
/*****************************/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

int ca,id;

/* array declarations for the */
/* generated requests and the */
/* effective requests ; n = */
/* the no. of processors */

int greq[256][2],x[256][2],efreq[256][2];
int i1,n,n1,n2,m,it,mxit,bw[6000],ctr[6000];
double abw,ebw,cr,cr1;
FILE *fp6;
/* m = no. of memory modules * mxit = number of iterations*/
/* fp6 is the pointer to the file containinig the random */
/* numbers for conflict resln */
void main(void)
{
  int i,j,p,t,req;
  /* ro = initial request rate; */
  double ro,po,ej,q,l,s,k,ph;
  /* ph = hotspot probability */
  FILE *fpl;
  /* for a request rate of 1; */
  /* ej = effective hotspot */
  fpl = fopen("result","w");/* probability while 'l' is */
mxit = 5000;    /* the probability of request-* /
 srand(19);     /* -ing non-hot modules * /
 for(n = 2;n<32;n = n*2) /* seed for random number gen-* /
 {    /* -erator is chosen as 19 * /
     n1 = log10(n) / log10 (2);/* n1 is the number of stages */
     n2 = n / 2;          /* n2 is the no. of switches */
     m = n;             /* in a stage */

     for(ro=0;ro < 1.01;ro = ro+0.1)
     {
         fprintf(fp1,"
"n");
         po = 1 - ro;
         fprintf(fp1,"%d\t%f\n",n,po);
         for(ph = 0.10;ph<1.0;ph=ph+0.1)
         {
             ebw = 0.0;
             cr = 0.0;
             crl = 0.0;
             abw = 0.0;
             ej = (ph * ro);

             fp6 = fopen("nums","r");
             for(it = 0;it<mxit;it++) /* 'for' loop for the number*/
             {    /* of iterations ( = mxit ) */
                 printf("%d\t%f\t%f\t%d\n",n,ro,ph,it);
                 bw[it] = 0;
                 req = 0;
                 ctr[it] = 0;
                 il = 1;

                /***************************************************************************/
                // GENERATION OF NON-UNIFORM DISTRIBUTED REQUESTS ; MEMORY */
                // MODULE '0' IS ACCESSSED MORE FREQUENTLY THAN OTHER MODULES */
                /***************************************************************************/

                 l = ((1 - po - ej) / (n - 1));
                 for(i=0;i<=(n-1);i++)
                 {
                     greq[i][0] = x[i][0] = efreq[i][0] = 0;
                 }
                 s = 32768.0000;
                 for(i=0;i<=(n-1);i++)
                 {
                     t = rand();
                     k = ( t / s );
                     if(k <= po)
                     {
                         greq[i][0] = x[i][0] = efreq[i][0] = 0;
                         greq[i][1] = 0;
                     }
                 }
else
{
  req++;
  if((k - po) <= ro)
  {
    greq[i][1] = 0;
    greq[i][0] = x[i][0] = efreq[i][0] = 1;
  }
  else
  {
    p = 1;
    q = 1;
    while(greq[i][0] == 0)
    {
      if ((k-po)<= (ro+q))
      {
        greq[i][1] = p;
        greq[i][0] = x[i][0] = efreq[i][0] = 1;
      }
      else;
    q = q+1;
    p = p+1;
  }
}

/**************************************************************************/
/************************************************************************** INITIAL SHUFFLE OF REQUESTS *************/
/**************************************************************************/
for (i = 0; i<=n-1; i++)
{
  j = shuf(i);
  x[i][1] = greq[i][1];
  efreq[j][1] = x[i][1];
  x[i][0] = greq[i][0];
  efreq[j][0] = x[i][0];
}
count();
switex();
for(j=0; j<n; j++)
{
  if(efreq[j][0] == 1)
  {
    bw[it] = bw[it] + 1;
  }
  else ;
}
UNBUF.C

Friday, January 17, 1992 6:17 pm

```c
    ebw = ebw + bw[it];
    if(req != 0)
        cr = cr + (double)(ctr[it])/(req);
    else;
}
fclose(fp6);
abw = (ebw / mxit);       /* average memory bandwidth*/
crl = (cr / mxit);        /* average blocking prob. */
fprintf(fp1,"%f\t%f\t%f\n",ej,abw,crl);
printf("%f\t%f\t%f\n",ej,abw,crl);
}
}
fclose(fp1);
}

/******************************************************************************/
/* function to calculate the number of requests to a module */
/******************************************************************************/

count()
{
    int k, l, y[256];
    for (l = 0; l <= n-1; l++)
        y[l] = 0;
    for (l = 0; l <= n-1; l++)
        {
            for (k = 0; k <= n-1; k++)
                {
                    if (x[k][l] == 1)
                        y[l] += 1;
                    else;
                }
            for (k = 0; k <= n-1; k++)
                x[k][l] = y[k];
        return;
    }
}

/*******************************************************************************
/* FUNCTION TO DO THE OPERATION OF A STAGE OF N/2 SWITCHES */
/*******************************************************************************

switex()
{
    int s, u, temp1, temp2, f1, f2, sh;
    for (s = 1; s <= nl; s++) /* 's' - the stage number */
        {
            /* 'u' - the link number */
```
for (u = 0; u <= (n - 1); u++)
{
    temp1 = efreq[u][1];
    temp2 = efreq[u+1][1];
    f1 = rotate(temp1);
    f2 = rotate(temp2);
    if ((efreq[u][0] == 1) && (efreq[u+1][0] == 1))
    {
        if ((f1 == 0) && (f2 == 0)) /* if there is any */
        {
            /* conflict, increase */
            ctr[itr]++;
            /* the counter for the */
            conflict();
            /* blocked requests */
            if (ca == 0)
            {
                /* call the conflict */
                /* resolving routine */
                efreq[u][1] = temp1; /* If the returned */
                efreq[u][0] = 1; /* no. is 1 then */
                /* request on lower */
            }
            /* is forwarded; else */
            else /* the upper request */
            {
                /* is forwarded */
                efreq[u+1][0] = 1; /* is forwarded */
                efreq[u][1] = temp2;
            }
        }
    }
    else
    {
        if ((f1 == 1) && (f2 == 1))
        {
            ctr[itr]++;
            conflict();
            if (ca == 0)
            {
                efreq[u+1][1] = temp1;
                efreq[u+1][0] = 1;
            }
            else
            {
                efreq[u+1][0] = 1;
                efreq[u+1][1] = temp2;
            }
        }
    }
    else
    {
        if ((f1 == 1) && (f2 == 0))
        {
            efreq[u][1] = temp2;
            efreq[u][0] = 1;
            efreq[u+1][1] = temp1;
            efreq[u+1][0] = 1;
        }
    }
}


```c
{ 
    efreq[u][1] = temp1;
    efreq[u][0] = 1;
    efreq[u+1][1] = temp2;
    efreq[u+1][0] = 1;
}
else
    if ((efreq[u][0] == 1) && (efreq[u+1][0] == 0))
    {
        if (f1 == 1)
            { 
                efreq[u+1][1] = temp1;
                efreq[u+1][0] = 1;
                efreq[u][0] = 0;
            }
        else
            { 
                efreq[u][1] = temp1;
                efreq[u][0] = 1;
                efreq[u+1][0] = 0;
            }
    }
    else
        if ((efreq[u][0] == 0) && (efreq[u+1][0] == 1))
        { 
            if (f2 == 0)
                { 
                    efreq[u][1] = temp2;
                    efreq[u][0] = 1;
                    efreq[u+1][0] = 0;
                }
            else
                { 
                    efreq[u+1][1] = temp2;
                    efreq[u+1][0] = 1;
                    efreq[u][0] = 0;
                }
        }
        else
        { 
            efreq[u][0] = efreq[u+1][0] = 0;
        }
}
if( s != n1 ) /* shuffling of requests in */
    { /* between stages */
        for(u=0;u<n-1;u++)
        {
            greq[u][1] = efreq[u][1];
            greq[u][0] = efreq[u][0];
        }
    }

```
UNBUF.C

for(u=0;u<=n-1;u++)
{
    sh = shuf(u);
    efreq[sh][1] = greq[u][1];
    efreq[sh][0] = greq[u][0];
}
else;
    i1 += 1;
return;
}

/***************************************************************************/
/* FUNCTION TO CALCULATE THE BIT TO BE TESTED AT EACH STAGE ****/
/***************************************************************************/

int rotate(t1)
int t1;
{
    int t2,t3;
    t2 = n1;
    t3 = t1 >> ( t2 - i1 );
    t3 = t3 & 1;
    return(t3);
}

/***************************************************************************/
/* FUNCTION TO EFFECT THE SHUFFLE ********/
/***************************************************************************/

int shuf(pnumb) /* 'pnumb' is the link number*/
int pnumb; /* which is to be shuffled */
{
    int zx,zy.zz;
    zx = pnumb;
    zy = n2;
    zz = ( n1 - 1 );
    zy = zy & zx;
    zy = zy >> zz;
    zx = zx << 1;
    zx = zy | zx;
    if(zx < n)
        return(zx);
    else
    {
        zx = zx ^ n;
        return(zx);
    }
conflict()     /* reading a random number */
{              /* from a file 'nums'; the */
    ca = 0;    /* number is either '0' or '1'*/
    fscanf(fp6,"%d",&ca); /* file length is 30000; if */
    id++;      /* the number of calls to this*/
    if(id == 29990) /* routine exceeds 29990, we */
    {          /* start from the beginning of*/
        fclose(fp6); /* the file again */
        fp6 = fopen("nums","r");
    }
    else;
    return;
}
M BUFL.C

Friday, January 17, 1992 6:26 pm

******************************************************************************
/* PROGRAM TO SIMULATE A SINGLE-BUFFERED OMEGA NETWORK UNDER A*/
/* HOTSPOT. CONFLICT RESOLUTION IS RANDOM. INITIAL RATE OF *****
/* REQUEST VARIES. PROGRAM CAN ALSO BE MODIFIED FOR *************/
/* PRIORITISED REALLOCATION OF CONFLICTS *******************/
******************************************************************************

******************************************************************************
/* Main routine : controls the flow of the program */
/* calculates the bandwidth. */
/* implements the switching operation */
/* generates random requests */
/* resolves the conflicts randomly */
/* gives the shuffled form of a link */
/* gives the destination bit to be */
/* tested at a stage */
******************************************************************************
/* buf[][][] and efreq[][][] contain the buffered and the */
/* effective requests at any stage. the first dimension is for*/
/* the stage number, the second dimension is for the link */
/* number while the third dimension is for storing the data */
/* and the associated tag of validity. n is the network size, */
/* ro the request rate,ph the hotspot probability, ej is the */
/* effective hotspot probability, tl the maximum number of */
/* iterations ; fp6 is a pointer to the file 'n rms' which */
/* contains the random numbers ( 0 or 1 ) for the resolution */
/* of conflicts. bw is the counter for the number of requests */
/* at the output in each cycle while abw gives the average */
/* memory bandwidth; nl is the no. of stages while n2 is the */
/* number of switches in each stage. results are written into */
/* a file pointed by fp4. */
******************************************************************************

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

int n,n1,n4,n2,n3,st,ca,i1,it,temp1,temp2;
int buf[9][256][2],efreq[9][256][2];
double l,po,ro,ej;
FILE *fp6;

void main(void)
{
    unsigned int i,j;o;
    unsigned int cy,t1;
double abw,bw,ph;
FILE *fp4;

fp4 = fopen("HOT64","w");
for(ro = 0.2;ro<1.01;ro = ro + 0.20)
{
  fprintf(fp4,"%f\n",ro);
  /*for(n = 8;n<64;n=2*n)*/
  n = 64;
  
  fprintf(fp4,"%d\t",n);
  temp1 = temp2 = 0;
  t1 = 55000;
  n1 = log10 (n)/log10 (2);
  n2 = n/2;
  n3 = (n1 - 1);
  n4 = n1 + 1;
  for(ph = 0.15;ph<0.51;ph = ph+0.1)
  {
    po = 1-ro;
    ej = ph * ro;
    l = ((1-ph)*ro)/(n-1);
    abw = 0.0;
    srand(19);
    /* for a different run, another*/
    /* seed is necessary */
    for(i = 0;i<n4;i++)
      for(j = 0;j<n;j++)
        for(o = 0;o<2;o++)
        {
          buf[i][j][o] = 0;
          efreq[i][j][o] = 0;
        }
  
  }/**********************************************************************************/

printf("%d\t%f\t%f\t",n,ro,ph);
fp6 = fopen("nums","r");
it = 0;
bw = 0.0;
for(cy = 0;cy<t1;cy++) /*cy - counter for no. of cycles*/
{
  for(j = 0;j<n;j++)
    efreq[n1][j][0] = 0;
  i1 = n1;
st = n3;
  while(st >= 0) /* operation is implemented from */
  {
    /* the last stage to the first */
    switex(); /* stage, one stage at a time */
    if(st != n3)
      shuffle(st+1);
  else ;
i1 = i1 - 1;
st = st - 1;
}
generate();     /* generation of new requests */
shuffle(0);     /* followed by their shuffle */
for(j = 0; j<n; j++)
{
    /* overlooking the first 500 */
    if(cy > 500)     /* cycles and then checking the */
        /* validity bit and counting the */
    if(efreq[n1][j][0] == 1) /* number of valid request*/
        bw++;     /* ts at network output */
    else;
    else;

} fclose(fp6);
abw = (bw / (t1-500)); /* averaging the bandwidth */
printf("average bandwidth is %f \n", abw);
printf(fp4,"%f\t", abw);
} fprintf(fp4,"\n");
} fprintf(fp4,"\n");
} fclose(fp4);

***************************************************************************/
generate()
{
    int t,p,b,c;
    double s,k,q;

    s = 32768.0000;
    for(b = 0; b<n; b++)
    {
        t = rand();
        k = (t/s);
        if(k <= po)     /* generation of requests accord- */
            /* ing to the procedure mentioned*/
        { efreq[0][b][0] = 0;
            efreq[0][b][1] = 0;
        }
        else
            { if((k-po) <= ej)
                {
efreq[0][b][0] = 1;
    efreq[0][b][1] = 0;
}
else
{
p = 1;
q = 1;
while(efreq[0][b][0] == 0)
{
    if((k-po) <= (e_j+q))
    {
        efreq[0][b][0] = 1;
        efreq[0][b][1] = p;
    }
    else;
    q = q+1;
    p = p+1;
}
}
return;

//****************************************************************************

int shuffle(w3)    /* shuffling of requests; */
    int w3;    /* w2[][] - temporary array */
{
    int e,f,w2[256][2];
    for(e = 0; e<n; e++)
    {
        w2[e][0] = 0;
        w2[e][1] = 0;
    }
    for(e = 0; e<n; e++)
    {
        w2[e][0] = efreq[w3][e][0];
        w2[e][1] = efreq[w3][e][1];
    }
    for(e = 0; e<n; e++)
    {
        f = shuf(e);
        efreq[w3][f][0] = w2[e][0];
        efreq[w3][f][1] = w2[e][1];
    }
    return;
}
int shuf(pnumb) 
{
    int zx,zy,zz;
    zx = pnumb;
    zy = n2;
    zz = n3;
    zy = zy & zx;
    zy = zy >> zz;
    zx = zx << 1;
    zx = zy ^ zx;
    if(zx < n)
        return(zx);
    else
    {
        zx = zx ^ n;
        return(zx);
    }
}
if(buf[u+1][0] == 0)
    temp2 = efreq[u+1][1];
else
    
    temp2 = buf[u+1][1];
    efreq[u+1][0] = 0;

f1 = rotate(temp1);
f2 = rotate(temp2);
sh = shuf(u);
sh1 = shuf(u+1);

if(buf[u][0] == 0) && (buf[u+1][0] == 0))
{
    if((efreq[u][0] == 1) && (efreq[u+1][0] == 1))
    {
        /* both requests are new */
        
        if((f1 == 0) && (f2 == 1))
        {
            if(buf[u+1][sh][0] == 0)
            {
                efreq[u+1][u][0] = 1;
                efreq[u+1][u][1] = temp1;
            }
            else
            {
                buf[u][0] = 1;
                buf[u][1] = temp1;
                efreq[u+1][u][0] = 0;
            }
        }
        if(buf[u+1][sh1][0] == 0)
        {
            efreq[u+1][u+1][0] = 1;
            efreq[u+1][u+1][1] = temp2;
        }
        else
        {
            buf[u+1][0] = 1;
            buf[u+1][1] = temp2;
            efreq[u+1][u+1][0] = 0;
        }
    }
    else
    if((f1 == 1) && (f2 == 0))
    {
        if(buf[u+1][sh][0] == 0)
        {
            efreq[u+1][u][0] = 1;
            efreq[u+1][u][1] = temp2;
MBUF1.C     Friday, January 17, 1992 6:26 pm

}
else
{
    buf[st][u+1][0] = 1;
    buf[st][u+1][1] = temp2;
    efreq[st+1][u][0] = 0;
}
if(buf[st+1][sh1][0] == 0)
{
    efreq[st+1][u+1][0] = 1;
    efreq[st+1][u+1][1] = temp1;
} else
{
    buf[st][u][0] = 1;
    buf[st][u][1] = temp1;
    efreq[st+1][u+1][0] = 0;
}
else
if((f1 == 0) && (f2 == 0))
{
    conflict();
    if(ca == 0)
    {
        if(buf[st+1][sh][0] == 0)
        {
            efreq[st+1][u][0] = 1;
            efreq[st+1][u][1] = temp1;
        } else
        {
            buf[st][u][0] = 1;
            buf[st][u][1] = temp1;
            efreq[st+1][u][0] = 0;
        }
        efreq[st+1][u+1][0] = 0;
        buf[st][u+1][0] = 1;
        buf[st][u+1][1] = temp2;
    } else
    {
        if(buf[st+1][sh][0] == 0)
        {
            efreq[st+1][u][0] = 1;
            efreq[st+1][u][1] = temp2;
        } else
        {

```
buf[st][u+1][0] = 1;
buf[st][u+1][1] = temp2;
efreq[st+1][u][0] = 0;
}
efreq[st+1][u+1][0] = 0;
buf[st][u][0] = 1;
buf[st][u][1] = temp1;
}
else
{
    conflict();
    if(ca == 0)
    {
        if(buf[st+1][sh1][0] == 0)
        {
            efreq[st+1][u+1][0] = 1;
            efreq[st+1][u+1][1] = temp1;
        }
        else
        {
            buf[st][u][0] = 1;
            buf[st][u][1] = temp1;
            efreq[st+1][u+1][0] = 0;
        }
        efreq[st+1][u][0] = 0;
        buf[st][u+1][0] = 1;
        buf[st][u+1][1] = temp2;
    }
    else
    {
        if(buf[st+1][sh1][0] == 0)
        {
            efreq[st+1][u+1][0] = 1;
            efreq[st+1][u+1][1] = temp2;
        }
        else
        {
            buf[st][u+1][0] = 1;
            buf[st][u+1][1] = temp2;
            efreq[st+1][u+1][0] = 0;
        }
        efreq[st+1][u][0] = 0;
        buf[st][u][0] = 1;
        buf[st][u][1] = temp1;
    }
}
else
    if((efreq[st][u][0] == 1) && (efreq[st][u+1][0] == 0))
MBUF1.C

Friday, January 17, 1992 6:26 pm

{
    /* upper buffer has new request */
    if(f1 == 0) /* lower buffer empty */
    {
        if(buf[st+1][sh][0] == 0)
            { efreq[st+1][u][0] = 1;
              efreq[st+1][u][1] = temp1;
            }
        else
            { buf[st][u][0] = 1;
              buf[st][u][1] = temp1;
              efreq[st+1][u][0] = 0;
            }
        efreq[st+1][u+1][0] = 0;
    }
else
    {
        if(buf[st+1][sh1][0] == 0)
            { efreq[st+1][u+1][0] = 1;
              efreq[st+1][u+1][1] = temp1;
            }
        else
            { buf[st][u][0] = 1;
              buf[st][u][1] = temp1;
              efreq[st+1][u+1][0] = 0;
            }
        efreq[st+1][u][0] = 0;
    }
}
else
if((efreq[st][u][0] == 0) && (efreq[st][u+1][0] == 1))
    {
        /* upper buffer empty */
        if(f2 == 0) /* lower buffer has new packet */
        {
            if(buf[st+1][sh][0] == 0)
                { efreq[st+1][u][0] = 1;
                  efreq[st+1][u][1] = temp2;
                }
            else
                { buf[st][u+1][0] = 1;
                  buf[st][u+1][1] = temp2;
                  efreq[st+1][u][0] = 0;
                }
            efreq[st+1][u+1][0] = 0;
            efreq[st][u+1][0] = 0;
        }
else
{
    if(buf[st+1][sh1][0] == 0)
    {
        efreq[st+1][u+1][0] = 1;
        efreq[st+1][u+1][1] = temp2;
    }
    else
    {
        buf[st][u+1][0] = 1;
        buf[st][u+1][1] = temp2;
        efreq[st+1][u+1][0] = 0;
    }
    efreq[st+1][u][0] = 0;
}
else ;
}

if((buf[st][u][0] == 1) && (buf[st][u+1][0] == 0))
{
    /* upper buffer has blocked packet*/
    if((efreq[st][u][0] == 0) && (efreq[st][u+1][0] == 1))
    {
        /* lower buffer has new packet */
        if((f1 == 0) && (f2 == 1)) /* for prioritised resln of*/
        {
            /* conflicts, instead of calling */
            if(buf[st+1][sh][0] == 0) /* the conflict() each */
            {
                /* time, the blocked request is */
                efreq[st+1][u][0] = 1; /* forwarded always. */
                efreq[st+1][u][1] = temp1;
                buf[st][u][0] = 0;
            }
            else
            {
                efreq[st+1][u][0] = 0;
            }
        }
    }
    if(buf[st+1][sh1][0] == 0)
    {
        efreq[st+1][u+1][0] = 1;
        efreq[st+1][u+1][1] = temp2;
    }
    else
    {
        buf[st][u+1][0] = 1;
        buf[st][u+1][1] = temp2;
        efreq[st+1][u+1][0] = 0;
    }
    efreq[st][u+1][0] = 0;
}
else
    if((f1 == 1) && (f2 == 0))
    {
        if(buf[st+1][sh][0] == 0)
        {
            efreq[st+1][u][0] = 1;
            efreq[st+1][u][1] = temp2;
        }
        else
        {
            buf[st][u+1][0] = 1;
            buf[st][u+1][1] = temp2;
            efreq[st+1][u][0] = 0;
        }
    }
    if(buf[st+1][sh1][0] == 0)
    {
        efreq[st+1][u+1][0] = 1;
        efreq[st+1][u+1][1] = temp1;
        buf[st][u][0] = 0;
    }
    else
    {
        efreq[st+1][u+1][0] = 0;
    }
    efreq[st][u+1][0] = 0;
else
    if((f1 == 0) && (f2 == 0))
    {
        conflict();
        if(ca == 0)
        {
            if(buf[st+1][sh][0] == 0)
            {
                efreq[st+1][u][0] = 1;
                efreq[st+1][u][1] = temp1;
                buf[st][u][0] = 0;
            }
            else
            {
                efreq[st+1][u][0] = 0;
            }
        }
        efreq[st+1][u+1][0] = 0;
        efreq[st][u+1][0] = 0;
        buf[st][u+1][0] = 1;
        buf[st][u+1][1] = temp2;
else
{
  efreq[st+1][u+1][0] = 0;
  if(buf[st+1][sh][0] == 0)
  {
    efreq[st+1][u][0] = 1;
    efreq[st+1][u][1] = temp2;
  }else
  {
    buf[st][u+1][0] = 1;
    buf[st][u+1][1] = temp2;
    efreq[st+1][u][0] = 0;
  }
  buf[st][u][0] = 1;
  buf[st][u][1] = temp1;
  efreq[st][u+1][0] = 0;
}
else
{
  conflict();
  if(ca == 0)
  {
    efreq[st+1][u][0] = 0;
    if(buf[st+1][sh1][0] == 0)
    {
      efreq[st+1][u+1][0] = 1;
      efreq[st+1][u+1][1] = temp1;
      buf[st][u][0] = 0;
    }else
    {
      efreq[st+1][u+1][0] = 0;
    }
    efreq[st][u+1][0] = 0;
    buf[st][u+1][0] = 1;
    buf[st][u+1][1] = temp2;
  }else
  {
    efreq[st+1][u][0] = 0;
    if(buf[st+1][sh1][0] == 0)
    {
      efreq[st+1][u+1][0] = 1;
      efreq[st+1][u+1][1] = temp2;
    }else
    {
    }
}
buf[st][u+1][0] = 1;
buf[st][u+1][1] = temp2;
efreq[st+1][u+1][0] = 0;
}
buf[st][u][0] = 1;
efreq[st][u+1][0] = 0;
buf[st][u][1] = temp1;
}
else
if((efreq[st][u][0] == 0) && (efreq[st][u+1][0] == 0))
{ /* lower buffer is empty */
  if(f1 == 0)
  {
    if(buf[st+1][sh][0] == 0)
    {
      buf[st][u][0] = 0;
      efreq[st+1][u][0] = 1;
      efreq[st+1][u][1] = temp1;
    }
    else
    {
      efreq[st+1][u][0] = 0;
    }
  
efreq[st+1][u+1][0] = 0;
  }
else
  {
    if(buf[st+1][sh1][0] == 0)
    {
      buf[st][u][0] = 0;
      efreq[st+1][u+1][0] = 1;
      efreq[st+1][u+1][1] = temp1;
    }
    else
    {
      efreq[st+1][u+1][0] = 0;
    }
  }
efreq[st+1][u][0] = 0;
}
else
if((buf[st][u][0] == 0) && (buf[st][u+1][0] == 1))
{ /* lower buffer has blocked packet*/
  if((efreq[st][u][0] == 1) && (efreq[st][u+1][0] == 0))
{ /* upper buffer has new packet */
  if((f1 == 0) && (f2 == 1))
  {
    if(buf[st+1][sh][0] == 0)
    {
      efreq[st+1][u][0] = 1;
      efreq[st+1][u][1] = temp1;
    }
    else
    {
      buf[st][u][0] = 1;
      buf[st][u][1] = temp1;
      efreq[st+1][u][0] = 0;
    }
    if(buf[st+1][sh1][0] == 0)
    {
      efreq[st+1][u+1][0] = 1;
      efreq[st+1][u+1][1] = temp2;
      buf[st][u+1][0] = 0;
    }
    else
    {
      efreq[st+1][u+1][0] = 0;
    }
    efreq[st][u][0] = 0;
  }
  else
  if((f1 == 1) && (f2 == 0))
  {
    if(buf[st+1][sh][0] == 0)
    {
      efreq[st+1][u][0] = 1;
      efreq[st+1][u][1] = temp2;
      buf[st][u+1][0] = 0;
    }
    else
    {
      efreq[st+1][u][0] = 0;
    }
    if(buf[st+1][sh1][0] == 0)
    {
      efreq[st+1][u+1][0] = 1;
      efreq[st+1][u+1][1] = temp1;
    }
    else
    {
      buf[st][u][0] = 0;
    }
buf[st][u][1] = temp1;
  efreq[st+1][u+1][0] = 0;
}
efreq[st][u][0] = 0;
} else
  if((f1 == 0) && (f2 == 0))
  {
    conflict();
    if(ca == 0)
    {
      if(buf[st+1][sh][0] == 0)
      {
        efreq[st+1][u][0] = 1;
        efreq[st+1][u][1] = temp1;
      }
      else
      {
        buf[st][u][0] = 1;
        buf[st][u][1] = temp1;
        efreq[st+1][u][0] = 0;
      }
      efreq[st+1][u+1][0] = 0;
      efreq[st][u][0] = 0;
      buf[st][u+1][0] = 1;
      buf[st][u+1][1] = temp2;
    }
    else
    {
      if(buf[st+1][sh][0] == 0)
      {
        efreq[st+1][u][0] = 1;
        efreq[st+1][u][1] = temp2;
        buf[st][u+1][0] = 0;
      }
      else
      {
        efreq[st+1][u][0] = 0;
      }
    }
  }
efreq[st+1][u+1][0] = 0;
efreq[st][u][0] = 0;
buf[st][u][0] = 1;
buf[st][u][1] = temp1;
}
else
{
  conflict();
  if(ca == 0)
{ 
    if(buf[st+1][sh1][0] == 0) 
    {
        efreq[st+1][u+1][0] = 1;
        efreq[st+1][u+1][1] = temp1;
    } 
    else 
    {
        buf[st][u][0] = 1;
        buf[st][u][1] = temp1;
        efreq[st+1][u+1][0] = 0;
    }
    efreq[st+1][u][0] = 0;
    efreq[st][u][0] = 0;
    buf[st][u+1][0] = 1;
    buf[st][u+1][1] = temp2;
}
else 
{
    if(buf[st+1][sh1][0] == 0) 
    {
        efreq[st+1][u+1][0] = 1;
        efreq[st+1][u+1][1] = temp2;
        buf[st][u+1][0] = 0;
    } 
    else 
    {
        efreq[st+1][u+1][0] = 0;
    }
}
    efreq[st+1][u][0] = 0;
    efreq[st][u][0] = 0;
    buf[st][u][0] = 1;
    buf[st][u][1] = temp1;
}
else 
if((efreq[st][u][0] == 0) && (efreq[st][u+1][0] == 0)) 
{
    /* upper buffer is empty */
    if(f2 == 0)
    {
        if(buf[st+1][sh][0] == 0)
        {
            efreq[st+1][u][0] = 1;
            buf[st][u+1][0] = 0;
            efreq[st+1][u][1] = temp2;
        }
        else 
        {

        }
efreq[st+1][u][0] = 0;
}

efreq[st+1][u+1][0] = 0;
else
{
    efreq[st+1][u][0] = 0;
    if(buf[st+1][sh1][0] == 0)
    {
        buf[st][u+1][0] = 0;
        efreq[st+1][u+1][0] = 1;
        efreq[st+1][u+1][1] = temp2;
    }
    else
    {
        efreq[st+1][u+1][0] = 0;
    }
}
else ;

if((buf[st][u][0] == 1) && (buf[st][u+1][0] == 1))
{ /*both buffers have blocked packets*/
    if((f1 == 0) && (f2 == 1))
    {
        if(buf[st+1][sh][0] == 0)
        {
            efreq[st+1][u][0] = 1;
            efreq[st+1][u][1] = temp1;
            buf[st][u][0] = 0;
        }
        else
        {
            efreq[st+1][u][0] = 0;
        }

        if(buf[st+1][sh1][0] == 0)
        {
            efreq[st+1][u+1][0] = 1;
            efreq[st+1][u+1][1] = temp2;
            buf[st][u+1][0] = 0;
        }
        else
        {
            efreq[st+1][u+1][0] = 0;
        }
    }
else
if((f1 == 1) && (f2 == 0))
{
    if(buf[st+1][sh][0] == 0)
    {
        efreq[st+1][u][0] = 1;
        efreq[st+1][u][1] = temp2;
        buf[st][u+1][0] = 0;
    }
    else
    {
        efreq[st+1][u][0] = 0;
    }
}

if(buf[st+1][sh1][0] == 0)
{
    efreq[st+1][u+1][0] = 1;
    efreq[st+1][u+1][1] = temp1;
    buf[st][u][0] = 0;
}
else
{
    efreq[st+1][u+1][0] = 0;
}
}
else
if((f1 == 0) && (f2 == 0))
{
    conflict();
    if(ca == 0)
    {
        efreq[st+1][u+1][0] = 0;
        if(buf[st+1][sh][0] == 0)
        {
            efreq[st+1][u][0] = 1;
            efreq[st+1][u][1] = temp1;
            buf[st][u][0] = 0;
        }
        else
        {
            efreq[st+1][u][0] = 0;
        }
        buf[st][u+1][0] = 1;
        buf[st][u+1][1] = temp2;
    }
    else
    {
        efreq[st+1][u+1][0] = 0;
        if(buf[st+1][sh][0] == 0)
{  
efreq[st+1][u][0] = 1;
efreq[st+1][u][1] = temp2;
buf[st][u+1][0] = 0;
}
else  
{
efreq[st+1][u][0] = 0;
}
buf[st][u][0] = 1;
buf[st][u][1] = temp1;
}
else  
{
conflict();
if(ca == 0)  
{
efreq[st+1][u][0] = 0;
if(buf[st+1][sh1][0] == 0)  
{
efreq[st+1][u+1][0] = 1;
efreq[st+1][u+1][1] = temp1;
buf[st][u][0] = 0;
}
else  
{
efreq[st+1][u+1][0] = 0;
}
buf[st][u+1][0] = 1;
buf[st][u+1][1] = temp2;
}
else  
{
efreq[st+1][u][0] = 0;
if(buf[st+1][sh1][0] == 0)  
{
efreq[st+1][u+1][0] = 1;
efreq[st+1][u+1][1] = temp2;
buf[st][u+1][0] = 0;
}
else  
{
efreq[st+1][u+1][0] = 0;
}
buf[st][u][0] = 1;
buf[st][u][1] = temp1;  
}
MBUF1.C       Friday, January 17, 1992 6:26 pm

#endif

}
}
#else ;
  efreq[st][u][0] = 0;
  efreq[st][u+1][0] = 0;
}
return;
}
/*----------------------------------------------------------------------------*/

conflict()
{
  ca = 0;
  fscanf(fp6,"%d",&ca);
  it++;
  if(it == 29990)
  {
    fclose(fp6);
    fp6 = fopen("nums","r");
  }
  else ;
  return;
}
/*----------------------------------------------------------------------------*/
MBUF.C   Friday, January 17, 1992 6:46 pm

/*************************************************************************/
/* PROGRAM TO SIMULATE A OMEGA NETWORK UNDER A HOTSPOT WITH */
/* QUEUE CAP OF MORE THAN ONE ; CONFLICT RESOLUTION IS RANDOM*/
/*************************************************************************/
/* buf[]][[] contains the request at each stage. the third */
/* dimension is queue length. req[]][] contains the address of*/
/* the current empty location in the queue. the first */
/* dimension is the stage number while the second is the link*/
/* number. other parameters are the same as they are in */
/* mbuf1.c . sd[] contains the total delay and gets increased*/
/* in each cycle; count[] contains the number of packets for */
/* a particular link. qe is the buffer size *********************/
/*************************************************************************/
/** most of the routines are the same as followed in mbuf1.c */
/** ex : rotate(), switex(), generate(), shuffle(), and shuf()*/
/*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

int n,n1,n4,qe,n2,n3,st,ca,il,it;
int req[9][100];
unsigned long buf[9][100][8],cy;
double ej,1,po,ro;
FILE *fp6;

void main(void)
{
    int i,j,o,a;
    unsigned long t1,my,sd[100],count[100],mv;
    double bw,abw,ph,sumd;
    FILE *fp8,*fp5;

    /*************************************************************************/
    /* mdel contains the final value of the delay while reals */
    /* contains the bw and the average delay for any packet */
    /*************************************************************************/

    fp5 = fopen("mdel","w");
    fp8 = fopen("reals","w");
    n = 64;
    {
        fprintf(fp5,"%d\n",n);
        for(ph=0.05; ph<0.30; ph=ph+0.05)
        {
            fprintf(fp8,"ph = %f\n",ph);
            fprintf(fp5,"%f\n",ph);
            t1 = 35000;
n1 = log10 (n)/log10 (2);
n2 = n/2;
n3 = (n1 - 1);
n4 = n1 + 1;
for(ro=0.1;ro<1.01;ro=ro+0.1)
{
    fprintf(fp5,"%f\n",ro);
    fprintf(fp8,"%f\t",ro);
    for(qe = 2; qe < 9; qe = qe + 2)
    {
        fprintf(fp5,"d\n",qe);
        po = 1.0 - ro;
        ej = ro * ph;
        l = (((1-ph)*ro)/(n-1));
        printf("%f\t%f\t%d\t",ph,ro,qe);
        bw = 0.0;
        abw = 0.0;
        srand(19);

        for(i = 0;i<n4;i++)
            for(a = 0;a<n;a++)
                for(o = 0;o<qe;o++)
                {
                    buf[i][a][o] = 0;
                    req[i][a] = 0;
                    sd[a] = 0;
                    count[a] = 0;
                }

        fp6 = fopen("nums","r");
        it = 0;
        sumd = 0.0;
        for(cy = 0;cy<t1;cy++)
        {
            my = 0;
            mv = 0;
            for(a = 0;a<n;a++)
            {
                buf[n1][a][0] = 0;
            }

            il = n1;
            st = n3;
            while(st >= 0)
            {
                switex();
                il = il - 1;
                st = st - 1;
            }
generate();
if (cy >= 500)
{

/*****************************/
/* when the number of cycles are greater than 500, then for */
/* each link at the output, the corresponding tag at the */
/* output array req[] is checked. if this contains a non- */
/* zero value it means that a request is present at the n/w */
/* ouput. the request is right shifted by the amount */
/* allocated for storing the address of the memory (16 bits */
/* here) and the resulting number which is the cycle in which*/
/* the packet was generated, is subtracted from the present */
/* cycle number cy to give the delay for the packet which is */
/* summed up over the cycles; corresponding count of access */
/* to that module is increased to give the total number of */
/* accesses to it */
/*****************************/

for(a = 0; a < n; a++)
{
    if(req[n1][a] != 0)
    {
        my = 0;
        my = buf[n1][a][0];
        mv = (my >> 16);
        sd[a] = sd[a] + cy - mv;
        count[a]++;
    }
    else
    {
        sd[a] = sd[a];
    }
}
else;
fclose(fp6);
for(a = 0; a < n; a++)
{
    fprintf(fp5,"%d\t", a);
    if(count[a] > 0)
    {

/*****************************/
/* check for zero accesses for the memory module quotient of */
/* the sum delay and count[] gives the delay for each request */
/*****************************/

    sumd = sumd + ((double) sd[a]/count[a]);
MBUF.C

printf("\%f\n",((double) sd[a]/count[a]));
fprintf(fp5,"\%f\n",((double) sd[a]/count[a]));

else
{
    printf("no\n");
fprintf(fp5,"no\n");
}
}

fprintf(fp8,"\%f\t",(sumd/(n*n1))); /* normalized delay*/
abw = (bw / (t1-500));
printf("\%f\t%f\n",ph,abw);
}

fprintf(fp5,"\n");
fprintf(fp8,"\n");
}
fprintf(fp5,"\n");
fprintf(fp8,"\n");
}
}

stdout

generate()
{
    unsigned long mz,mx;
    int t,p,j1,b1,b2,tagp;
    double vs,k,lq;

    mz = 0;
    mx = 0;
    vs = 32768.0000;

    /*******************************************/
    /* the cycle number is shifted left by 16 bits and appended to */
    /* the request generated for calculation of delay *************/
    /*******************************************/
    
    mx = (cy << 16);
    for(j1 = 0;j1<n;j1++)
    {
        b1 = shuf(j1);
        b2 = req[0][b1];
        if(b2 < qe)
        {
        }
t = rand();
k = (t/vs);
if(k <= po)
{
    req[0][b1] = b2;
    buf[0][b1][b2] = 0;
}
else
{
    if((k-po) <= ej)
    {
        buf[0][b1][b2] = 0;
        req[0][b1] = b2 + 1;
    }
    else
    {
        tagp = 0;
p = 1;
lq = 1;
while(tagp == 0)
{
    if((k-po) <= (ej+lq))
    {
        buf[0][b1][b2] = p;
        req[0][b1] = b2 + 1;
        tagp = 1;
    }
    else;
lq = lq+1;
p = p+1;
}
    mz = buf[0][b1][b2];
    buf[0][b1][b2] = (mz ; mx);
}
/*printf("%ld\t%ld\t%ld\n",mz,mx,buf[0][j1][b2]);*/
}
else ;
return;
}

/***********************************************************************************************/
switex()
{
    void shift();
    unsigned long f1,f2,templ,temp2,rotate();
    unsigned int sh,sh1,u;
int s1, s2, s3, s4, g, h;

temp1 = 0;
temp2 = 0;
for (u = 0; u < n; u = u + 2)
{
    if (st != n3)
    {
        sh = shuf(u);
        sh1 = shuf(u + 1);
    }
    else
    {
        sh = u;
        sh1 = u + 1;
        req[n1][sh] = 0;
        req[n1][sh1] = 0;
    }

s1 = req[st][u];
s2 = req[st][u + 1];
s3 = req[st + 1][sh];
s4 = req[st + 1][sh1];

temp1 = buf[st][u][0];
f1 = rotate(temp1);
temp2 = buf[st][u + 1][0];
f2 = rotate(temp2);

if ((s1 != 0) && (s2 != 0))
{
    if (cy != 0)
    {
        if ((f1 == 0) && (f2 == 1))
        {
            if (s3 < qe)
            {
                buf[st + 1][sh][s3] = temp1;
                req[st + 1][sh] = s3 + 1;
                shift(s1, u);
                req[st][u] = s1 - 1;
            } else ;

            if (s4 < qe)
            {
                buf[st + 1][sh1][s4] = temp2;
                req[st + 1][sh1] = s4 + 1;
                shift(s2, u + 1);
                req[st][u + 1] = s2 - 1;
            }
        }
    }
}

else;
else
if((f1 == 1) && (f2 == 0))
{
  if(s4 < qe)
  {
    buf[st+1][sh1][s4] = temp1;
    req[st+1][sh1] = s4 + 1;
    shift(s1,u);
    req[st][u] = s1 - 1;
  }
  else;

  if(s3 < qe)
  {
    buf[st+1][sh][s3] = temp2;
    req[st+1][sh] = s3 + 1;
    shift(s2,u+1);
    req[st][u+1] = s2 - 1;
  }
  else;
}
else
if((f1 == 0) && (f2 == 0))
{
  conflict();
  if(ca == 0)
  {
    if(s3 < qe)
    {
      buf[st+1][sh][s3] = temp1;
      req[st+1][sh] = s3 + 1;
      shift(s1,u);
      req[st][u] = s1 - 1;
    }
    else;
  }
else
  {
    if(s3 < qe)
    {
      buf[st+1][sh][s3] = temp2;
      req[st+1][sh] = s3 + 1;
      shift(s2,u+1);
      req[st][u+1] = s2 - 1;
    }
    else;
  }
}
}  
else  
if((f1 == 1) && (f2 == 1))  
{  
    conflict();  
    if(ca == 0)  
    {  
        if(s4 < qe)  
        {  
            buf[st+1][sh1][s4] = temp1;  
            req[st+1][sh1] = s4 + 1;  
            shift(s1,u);  
            req[st][u] = s1 - 1;  
        }  
        else ;  
    }  
else  
{  
    if(s4 < qe)  
    {  
        buf[st+1][sh1][s4] = temp2;  
        req[st+1][sh1] = s4 + 1;  
        shift(s2,u+1);  
        req[st][u+1] = s2 - 1;  
    }  
    else ;  
}  
else ;  
}  
else ;  
else if((s1 != 0) && (s2 == 0))  
{  
    if(cy != 0)  
    {  
        if(f1 == 0)  
        {  
            if(s3 == 0)  
            {  
                if(s3 < qe)  
                {  
                    buf[st+1][sh][s3] = temp1;  
                    req[st+1][sh] = s3 + 1;  
                    shift(s1,u);  
                    req[st][u] = s1 - 1;  
                }  
                else ;  
            }  
            else  
        }  
    }
if(s4 < qe)
{
    buf[st+1][sh1][s4] = temp1;
    req[st+1][sh1] = s4 + 1;
    shift(s1,u);
    req[st][u] = s1 - 1;
}
else ;
}
else ;
}
else
if((s1 == 0) && (s2 != 0))
{
    if(cy != 0)
    {
        if(f2 == 0)
        {
            if(s3 < qe)
            {
                buf[st+1][sh][s3] = temp2;
                req[st+1][sh] = s3 + 1;
                shift(s2,u+1);
                req[st][u+1] = s2 - 1;
            }
            else ;
        }
        else
        {
            if(s4 < qe)
            {
                buf[st+1][sh1][s4] = temp2;
                req[st+1][sh1] = s4 + 1;
                shift(s2,u+1);
                req[st][u+1] = s2 - 1;
            }
            else ;
        }
    }
    else ;
}
else ;
}
return;
}/* *********************************************************/

void shift(x1,x2)
    int x1;
unsigned int x2;
{
    int f;
    unsigned long temp[8];
    for(f=0; f<8; f++)
        temp[f] = 0;
    for(f = 0; f<x1; f++)
        temp[f] = buf[st][x2][f];
    for(f = 0; f < (x1-1); f++)
        buf[st][x2][f] = temp[f+1];
    return;
}

/*************************************************************/

int shuf(pnumb)
int pnumb;
{
    int zx,zy,zz;
    zx = pnumb;
    zy = n2;
    zz = n3;
    zy = zy & zx;
    zy = zy >> zz;
    zx = zx << 1;
    zx = zy : zx;
    if(zx < n)
        return(zx);
    else
    {
        zx = zx ^ n;
        return(zx);
    }
}

/*************************************************************/

unsigned long rotate(t4)
unsigned long t4;
{
    unsigned long t3;
    t3 = t4 >> (n1 - i1);
    t3 = t3 & 1;
    return(t3);
}

/*************************************************************/

conflict()
MBUF.C

Friday, January 17, 1992 6:46 pm

{
    ca = 0;
    fscanf(fp6,"%d",&ca);
    it++;
    if(it == 29990)
    {
        fclose(fp6);
        fp6 = fopen("nums","r");
    }
    else
    
    return;
}
DIST.C  Friday, January 17, 1992 7:07 pm

/*******************
/* PROGRAM TO SIMULATE A QUEUED NETWORK UNDER HOTSPOTS WITH A */
/* QUEUE CAP OF 1 AND FOR DIFFERENT NETWORK SIZES ; CONFLICT */
/* RESOLUTION IS RANDOM GENERATION OF NEW REQUESTS TAKES PLACE*/
/* IF A REQUEST IS BLOCKED MOST OF THE SYMBOLS AND ROUTINES */
/* ARE COMMON TO EARLIER PROGRAMS. ONLY ROUTINES THAT ARE NEW */
/* ARE EXPLAINED HERE ******************************* */
/* THE ROUTINES GENERATE(), SHUFFLE(), SHUF(), CONFLICT() */
/* ROTATE() ARE EXACTLY THE SAME AS GIVEN IN MBUF1.C AND HENCE*/
/* ARE NOT LISTED HERE. THE MAJOR PART OF SWITEX() IS ALSO THE*/
/* SAME AS IN MBUF1.C; ONLY THAT PORTION THAT IS NEW IS LISTED */
/* SUFFICIENT COMMENTS HAVE BEEN ADDED SO AS TO LINK THINGS */
/*******************

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

unsigned int cy;
int n,n1,n4,bret,n2,n3, st, ca, i1, it, temp1, temp2;
int upper, lower, buf[9][256][2], efreq[9][256][2];
double ej,l,et,el,po,ro;
FILE *fp6;
FILE *fp4;

void main(void)
{
    unsigned int i,j,o,t1,tag1,tag2,m1,m2;
double abw,bw;
    fp4 = fopen("res","w");
    n = 8;
{
    temp1 = temp2 = 0;
t1 = 50500;
n1 = log10 (n)/log10 (2);
n2 = n/2;
n3 = (n1 - 1);
n4 = n1 + 1;
/*for(po = 0.0;po<1.0;po=po+0.1)*/
    po = 0.7;
{
    ro = 1 - po;
printf("%f\t",ro);
    ej = 0.125*ro;
l = (((1-0.125)*ro)/(n-1));
    abw = 0.0;
srand(19);
    for(i = 0;i<n4;i++)
    for(j = 0;j<n;j++)
for(o = 0; o<2; o++)
{
    buf[i][j][o] = 0;
    efreq[i][j][o] = 0;
}
fp6 = fopen("nums","r");
it = 0;
upper = 0;
lower = 0;
et = 0.0;
el = 0.0;
bw = 0.0;
for(cy = 0; cy<t1; cy++)
{
    for(j = 0; j<n; j++)
        efreq[n1][j][0] = 0;
    il = n1;
    st = n3;
    while(st >= 0)
    {
        switch();
        if(st != n3)
            shuffle(st+1);
        else;
        il = il - 1;
        st = st - 1;
    }
generate();
shuffle(0);
for(j = 0; j<n; j++)
{
    if (cy >= 500)
    {
        if(efreq[n1][j][0] == 1)
            bw++;
        else ;
    }
    else ;
}
}
fclose(fp6);
abw = (bw / (t1-500));
printf("average bandwidth is %f\n",abw);
} fprintf(fp4,"\n");
} fclose(fp4);
switex()
{
    int gh, u, f1, f2, sh, sh1;

    for(u = 0; u < n; u = u+2) {
        gh = u;                  /* gh - present link number */
        if(st == 0) {
            temp1 = efreq[st][u][1];
            temp2 = efreq[st][u+1][1];
            buf[st][u][0] = 0;
            buf[st][u+1][0] = 0;
        } else {
            if(buf[st][u][0] == 0)
                temp1 = efreq[st][u][1];
            else {
                temp1 = buf[st][u][1];
                efreq[st][u][0] = 0;
            }
            if(buf[st][u+1][0] == 0)
                temp2 = efreq[st][u+1][1];
            else {
                temp2 = buf[st][u+1][1];
                efreq[st][u+1][0] = 0;
            }
        }
        f1 = rotate(temp1);
        f2 = rotate(temp2);
        sh = shuf(u);
        sh1 = shuf(u+1);
    }
}
/***************

if((buf[st][u][0] == 1) && (buf[st][u+1][0] == 1))
{
    if(st == 1)
    {
        /* the choice of links at each stage depends on the index of */
        /* hot module; et is the new effective hotspot probability */
        /* while el is the new effective probability of requesting the */
        /* non-hot modules accessible from the input link which */
        /* contains a blocked packet. distrib() is the routine which */
        /* generates a new request and distributes it according to the */
        /* hotspot distribution; bret is an index for knowing from */
        /* which half the input buffer is. in the following segment, */
        /* the distribution is for urm; the same can be done for */
        /* non-urm after perusal of the thesis. */
        if((u == 0) && (u == 4))
        {
            et = 0.25*ro;
            el = 0.25*ro;
            bret = 0;
            distrib(gh);
        }
        else
        {
            et = 0.25*ro;
            el = 0.25*ro;
            bret = u;
            distrib(gh);
        }
    }
    else
    {
        if(st == 2)
        {
            if(u == 0)
            {
                et = 0.5*ro;
                el = 0.5*ro;
                bret = 0;
                distrib(gh);
            }
            else
            {
                et = 0.5*ro;
                el = 0.5*ro;
                bret = u;
                distrib(gh);
            }
        }
    }
}
DIST.C

Friday, January 17, 1992 7:07 pm

}
}
else;
}
else;
}
return;
}

/*********************/
/* part calling distribute *****************************/
/*********************/

distrib(jh)
int jh;
{
if(buf[st][jh][0] == 1)
distribute(jh);
else;
if(buf[st][jh+1][0] == 1)
distribute(jh+1);
else;
return;
}

/*********************/
/* request generation and distribution according to et and el;*/
/* this is called from distrib(). a3 is the link number that  */
/* contains blocked packet *****************************/
/*********************/

distribute(a3)
int a3;
{
int pl,t1,ttag;
float s1,k1,q1;
s1 = 32768.00;

t1 = rand();
ttag = 0;
k1 = t1 / s1;
if(k1 <= po)
{
    buf[st][a3][0] = 0;
}
else
{
    if((k1-po) <= et)
    {

buf[st][a3][1] = (0 + bret);
}
else
{
    pl = 1;
    ql = el;
    while(ttag == 0)
    {
        if((k1-po) <= (et+q1))
        {
            buf[st][a3][1] = (pl + bret);
            ttag = 1;
        }
        else;
        ql = ql+el;
        pl = pl+1;
    }
}
return;
CSSTATEMENTS=0

** finding the bandwidth of N x N omega networks under
** hot spot conditions. Basically this program is to
** compute the analytical equations developed by me.

REAL*8 P(0:256,0:256), VAL(0:9), P00, Q, QP, AMBW
INTEGER STAGE, PTYPE, N, I, J, K, NSTAGE, TIMES

** p = probability matrix
** p(0,0)= probability that a processor makes a request
** i.e. PA of Liu and Jou's paper
** P00 = SAME AS P(0,0) =P(-1,0) IN PAPER
** ptype = probability type. Stage n has (n+1) probability
** types. See p. 41 of notes.
** n = no. of procs./ memories
** Q = PROB OF HOT SPOT MEMORY = PH IN PAPER
** QP = PROB OF NON-HOT SPOT MEMORY = PH' IN PAPER
** AMBW = average memory bandwidth
** OR = PERCENTAGE OF REQUESTS THAT ARE HOT

DO 999 Q = 0.1, 1.01, 0.1
  WRITE (1,*) '********** Q= ',Q,' **********'
C WRITE (2,*) '********** P(0,0)= ',P(0,0),',**********'
C DO 999 TIMES = 1, 1
DO 999 P00 = 0.1, 1.01, 0.1
  P(0,0) = P00
DO 35 K = 0.9, 1
35 VAL(K) = 0.0
  K = 0
DO 998 NSTAGE = 2, 8
  N = 2**NSTAGE
  Q = (1.0/REAL(N+(TIMES-1))) *REAL(TIMES)
  qp = (1-q)/(n-1)
do 20 stage = 1, NSTAGE
  do 20 ptype = 0, stage
    IF ( ptype .EQ. 1 ) THEN
      CALL HOTSW(STAGE, PTYPE, N, P, Q, QP)
    ELSE
      CALL NHOTSW(STAGE, PTYPE, P)
    ENDIF
20 CONTINUE

C WRITE (*,*) STAGE, PTYPE, P(STAGE,PTYPE)

20 CONTINUE

AMBW = AMBW + P(NSTAGE,1) * 2**(n-1)

VAL(K) = AMBW
K = K+1
998 CONTINUE

WRITE (1,5) P00, (VAL(J), J = 0, 6)
C    FORMAT(1X,'Q=',F9.5,' AMBW=',F10.5)
5    FORMAT(1X,F3.1,2X,7(F5.2,2X))
999   CONTINUE
      stop
      end

C ***************************************************************
subroutine nhotsv(stage, ptype, p)
      real*8 P(0:256,0:256)
      integer stage, ptype
      p(stage, ptype) = p(stage-1,ptype-1) -
      *     (p(stage-1,ptype-1)**2) * 0.25
      return
      end

C ***************************************************************
subroutine hotsw(stage, ptype, n, p, q, qp)
      integer stage, ptype, n
      real*8 TEMP
      real*8 P(0:256,0:256), Q, QP
      temp = ( (n/2**(stage))\*qp + (q-qp)*((1-ptype) )
      *     ( (n/2**(stage-1))\*qp + q )
      c write (*,*)' hotsw ',stage, ptype, n,q,qp,temp
      p(stage,ptype) = 2*p(stage-1,0) * temp -
      *     ((p(stage-1,0))**2) * temp**2
      return
      end
C PROGRAM TO COMPUTE THE BANDWIDTH AND THE DELAY FROM MODEL 1.

C

1 REAL QH(5,0:15), PON(5,0:15), PH(5)
2 REAL RHN(5,0:15), QNM(5,0:15), PHN(5,0:15)
3 REAL QH1(5,0:15), PON1(5,0:15), PH1(5,0:15), RHN(5,0:15), PHN(5)
4 REAL TEMP1, TEMP2, TEMP3, TEMP4, TEMP5, TEMP6, TEMP7, TEMP8, RHN(5,0:15)
5 REAL TEMP9, TEMP10, TEMP11, TEMP12, TEMP13, Q, Q1, R
6 REAL D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15
7 INTEGER I, J, K, L, M, N, T

C

C THE NUMBER OF CYCLES DEPEND ON THE STEADYING OF THE OUTPUT

C PROGRAM IS FOR 8 x 8 NETWORK

8 T = 12
9 N = 3
10 DO 5 K = 1, N
11 DO 3 I = 0, T
12 QH(K, I) = 0.0
13 PON(K, I) = 0.0
14 PH(K, I) = 0.0
15 RHN(K, I) = 0.0
16 QNM(K, I) = 0.0
17 PHN(K, I) = 0.0
18 PON1(K, I) = 0.0
19 PH1(K, I) = 0.0
20 RHN(K, I) = 0.0
21 QH1(K, I) = 0.0
22 PON1(K, I) = 0.0
23 PH1(K, I) = 0.0
24 RHN(K, I) = 0.0
25 PH(K) = 0.0
26 PHN(K) = 0.0
27 R = 0.0
28 Q = 0.0
29 Q1 = 0.0
30 TEMP1 = 0.0
31 TEMP2 = 0.0
32 TEMP3 = 0.0
33 TEMP4 = 0.0
34 TEMP5 = 0.0
35 TEMP6 = 0.0
36 TEMP7 = 0.0
37 TEMP8 = 0.0
38 TEMP9 = 0.0
39 TEMP10 = 0.0
40 TEMP11 = 0.0
41 TEMP12 = 0.0
42 TEMP13 = 0.0
43 3 CONTINUE
44 5 CONTINUE

C

C PH AND PHN ARE THE PROBABILITIES OF ACCESSING THE HOT AND
C NON-HOT OUTPUT LINKS OF A HOT SWITCH
C
C DO 450 Q = 0.6, 1.01, 0.1
Q = 0.3
Q1 = (1.0 - Q)/(2**M) - 1
DO 100 R = 0.1, 1.01, 0.1
EQ = R * Q
1      EQ = R * Q1
DO 10 K = 1, N
TEMP1 = (((2**M)/(2**K)) - 1) * Q1 + Q
TEMP2 = (((2**M)/(2**K-1)) - 1) * Q1 + Q
PH(K) = TEMP1 / TEMP2
PMH(K) = 1.0 - PH(K)
10 CONTINUE
C
C ******************************************************************************************
C
56      QH(1,0) = R
57      POH(1,0) = 1.0
58      C      DO 20 K = 1, N
59      C      POH(K,0) = 1.00
60      C      CONTINUE
C
C ******************************************************************************************
C
58      DO 40 I = 1, T
59      DO 30 K = 1, N
60      IF ((K,GT,1)) THEN
61      QH(K,1) = 1.0 - ((1.0 - (PH(K-1) * PHH(K-1,1)))**2)
62      QHI(K,1) = 1.0 - ((1.0 - (PMH(K-1) * PHH(K-1,1)))**2)
63      QNH(K,1) = 1.0 - ((1.0 - (PHH(K-1,1)/2.0))**2)
64      ELSE
65      QH(K,1) = R
66      ENDIF
67      TEMP1 = 1.0 - QH(K,1-1)
68      TEMP2 = POH(K,1-1) + (PHH(K,1-1) * (RH(K,1-1)+RHH(K,1-1)))
69      POH(K,1) = TEMP1 * TEMP2
70      PHH(K,1) = 1.0 - POH(K,1)
71      IF ((K,GT,1).AND.(1,GT,1)) THEN
72      TEMP3 = 1.0 - QHI(K,1-1)
73      TEMP4 = POH(K,1-1) + (2.0 * PHH(K,1-1) * RHI(K,1-1))
74      POH(K,1) = TEMP3 * TEMP4
75      PHH(K,1) = 1.0 - POH(K,1)
76      ENDIF
77      IF ((K,GT,2).AND.(1,GT,2)) THEN
78      TEMP5 = 1.0 - QNH(K,1-1)
79      TEMP6 = POH(K,1-1) + (2.0 * PHH(K,1-1) * RHH(K,1-1))
80      POH(K,1) = TEMP5 * TEMP6
81      PHH(K,1) = 1.0 - POH(K,1)
82      ENDIF
83      CONTINUE
84      DO 35 K = 1, N-1
85      IF (K,EQ,N) THEN
86      IF (1,GT,2) THEN
87      RH(N,1) = PH(K) * (1.0 - (PH(K) * PHH(K,1)/2.0))
88      RHH(N,1) = PHH(K) * (1.0 - (PMH(K) * PHH(K,1)/2.0))
89      RHI(K,1) = ((1.0 - (PHH(K,1)/4.0))/2.0)
90      RNH(K,1) = ((1.0 - (PHH(K,1)/4.0))/2.0)
91   ENDIF
92   ELSE
93     IF (((1.EQ.1).AND.(K.EQ.2)) GO TO 33
94        TEMP3 = PH(K) * (1.0 - ( PH(K) * PINH(K,1) / 2.0 )
95        TEMP4 = PINH(K+1,1) + (PINH(K+1,1) * (RH(K+1,1)+RH(K+1,1)))
96        RH(K,1) = TEMP3 * TEMP4
97        TEMP5 = PMH(K) * (1.0 - ( PMH(K) * PINH(K,1) / 2.0 ))
98        TEMP6 = PMH(K+1,1) + (PINH(K+1,1) * 2.0 * RH(K+1,1))
99        RHN(K,1) = TEMP5 * TEMP6
100   CONTINUE
101     IF ((K.GT.1).AND.(1.GT.1)) THEN
102        TEMP6 = 0.5 * (1.0 - (PINH(K,1)/4.0))
103        TEMP7 = PMINH(K+1,1) + (2.0 * PINH(K+1,1) * RHN(K+1,1))
104        RH1(K,1) = TEMP6 * TEMP7
105     ENDIF
106   CONTINUE
107  CONTINUE
108  CONTINUE
109     TEMP10 = PINH(3,12) * 2.0 * RH(3,12)
110     TEMP11 = PINH(3,12) * 2.0 * RHN(3,12)
111     TEMP12 = PINH(3,12) * 2.0 * RH(3,12)
112     TEMP13 = PINH(3,12) * 2.0 * RHN(3,12)
113     TEMP14 = TEMP10 + TEMP11 + (2.0*TEMP12) + (4.0*TEMP13)
114     WRITE (7,38) R,TEMP14
115     C D1 = (1.0 / (PINH(1) * RH(1,T))
116     C D2 = (1.0 / (PINH(2) * RH(2,T))
117     C D3 = (1.0 / (PINH(3) * RH(3,T))
118     C D4 = (1.0 / (PMINH(1) * RHN(1,T))
119     C D5 = (1.0 / (PMINH(2) * RHN(2,T))
120     C D6 = (1.0 / (PMINH(3) * RHN(3,T))
121     C D7 = (2.0 / RH1(2,T))
122     C D8 = (2.0 / RH1(3,T))
123     C D9 = (2.0 / RHN(3,T))
124     C D10 - DELAY FOR A HOT PACKET : OTHER DELAYS FOR THE PACKETS
125     C USING THE OTHER THREE TYPES OF OUTPUT LINKS
126     C D10 = D1 + D2 + D3
127     C D11 = D1 + D2 + D6
128     C D12 = D1 + D5 + D8
129     C D13 = D4 + D7 + D9
130     C D14 = (D10 + D11 + (2.0 * D12) + (4.0 * D13)) / 24.0
131     C WRITE (8,39) R,D10,D11,D12,D13,D14
132 38   FORMAT (2X,F6.4,2X,F7.4)
133 39   FORMAT (2X,F6.4,5(2X,F7.4))
134 100  CONTINUE
135     WRITE(7,37) '     '
136     WRITE(7,37) '     '
137 100  CONTINUE
138     STOP
139     END
C **********************************************************************
C PROGRAM TO EVALUATE MODEL 3
C
C THE FOLLOWING PROGRAM IS VALID FOR AN 8 X 8 NETWORK ONLY.
C THE PROGRAM LOGIC CAN BE USED FOR A NETWORK OF ANY SIZE.
C THE FOLLOWING ARRAY DECLARATIONS ARE MADE ; NUMBER OF ARRAYS
C INCREASE WITH THE SIZE OF THE NETWORK . THE HOT SWITCHES ARE
C IDENTIFIED BY A PREFIX H WHILE THE SWITCHES OF TYPE 1 ARE
C IDENTIFIED BY A PREFIX N ; WHILE THE SWITCHES OF TYPE 2 ARE
C IDENTIFIED BY A PREFIX O . THE FIRST ARGUMENT TO THE ARRAYS IS THE
C CYCLE NUMBER WHILE THE SECOND ARGUMENT IS THE STAGE NUMBER .
C ALL THE PROBABILITIES DEFINED IN THE MODEL ARE DECLARED HERE.
C **********************************************************************

1 REAL HP0(100,5), HPN(100,5), HPBH(100,5), HRBH(100,5), HRBN(100,5)
2 REAL HPNH(100,5), HPNBH(100,5), HRP(100,5), HRNP(100,5)
3 REAL HPO(100,5), HNP(100,5), HBN(100,5), HPBH(100,5), HRP(100,5), HPBH(100,5)
4 REAL HPB(100,5), HPB(100,5), HNB(100,5), HP(100,5), HPB(100,5), HPB(100,5)
5 REAL HBN(100,5), HNB(100,5), HPB(100,5), HPB(100,5)
6 REAL NPO(100,5), NPN(100,5), NPB(100,5), NPB(100,5), NPN(100,5), NPB(100,5)
7 REAL NPBH(100,5), NPBH(100,5), NPBH(100,5), NPBH(100,5), NPBH(100,5), NPBH(100,5)
8 REAL NRP(100,5), NRP(100,5), NRP(100,5), NRP(100,5), NRP(100,5), NRP(100,5)
9 REAL NRPBH(100,5), NRPBH(100,5), NRPBH(100,5), NRPBH(100,5), NRPBH(100,5), NRPBH(100,5)
10 REAL NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5)
11 REAL NRBH(100,5), NRBH(100,5), NRBH(100,5), NRBH(100,5), NRBH(100,5), NRBH(100,5)
12 REAL NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5)
13 REAL NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5)
14 REAL NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5)
15 REAL NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5), NRB(100,5)
16 REAL EQ, EQ1, HBBH(100), HBBH(100), HBBH(100), HBBH(100), HBBH(100), HBBH(100)
17 INTEGER J, K, N, T
C **********************************************************************
C T IS THE NUMBER OF CYCLES WHILE N IS THE NUMBER OF STAGES IN THE
C NETWORK ; Q IS THE HOTSPOT PROBABILITY WHILE Q1 IS THE PROBABILITY
C OF ACCESSING THE NON-HOT MEMORIES. R IS THE REQUEST RATE .
C EQ IS THE EFFECTIVE HOTSPOT PROBABILITY WHILE EQ1 IS THE
C EFFECTIVE PROBABILITY FOR ACCESSING THE NON-HOT MEMORIES .
C **********************************************************************

18 T = 19
19 N = 3
20 DO 30 Q = 0.2, 1.01, 0.2
21 DO 55 R = 1.0, 1.01, 0.1
22 EQ = Q * R
23 Q1 = (1 - Q) / ((2**N) - 1)
24 EQ1 = Q1 * R
C
C **********************************************************************
C INITIALIZATION
C **********************************************************************

C
C **********************************************************************

C
C
C
C
C
C 25 DO 10 I = 1, T
26 DO 9 J = 1, N
<table>
<thead>
<tr>
<th>Row</th>
<th>Equation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>HD(J) = 0.0</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>ND(J) = 0.0</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>OD(J) = 0.0</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>HPO(I,J) = 0.0</td>
<td>HBU00400</td>
</tr>
<tr>
<td>31</td>
<td>HBMH(I) = 0.0</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>HBMN(I) = 0.0</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>HBM(I) = 0.0</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>OBM(I) = 0.0</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>HPM(I,J) = 0.0</td>
<td>HBU00410</td>
</tr>
<tr>
<td>36</td>
<td>HPBH(I,J) = 0.0</td>
<td>HBU00420</td>
</tr>
<tr>
<td>37</td>
<td>HPBN(I,J) = 0.0</td>
<td>HBU00430</td>
</tr>
<tr>
<td>38</td>
<td>HRNTH(I,J) = 0.0</td>
<td>HBU00440</td>
</tr>
<tr>
<td>39</td>
<td>HRNTH(I,J) = 0.0</td>
<td>HBU00450</td>
</tr>
<tr>
<td>40</td>
<td>HRBTH(I,J) = 0.0</td>
<td>HBU00460</td>
</tr>
<tr>
<td>41</td>
<td>HRBTH(I,J) = 0.0</td>
<td>HBU00470</td>
</tr>
<tr>
<td>42</td>
<td>HRNMO(I,J) = 0.0</td>
<td>HBU00480</td>
</tr>
<tr>
<td>43</td>
<td>HRNMO(I,J) = 0.0</td>
<td>HBU00490</td>
</tr>
<tr>
<td>44</td>
<td>HRBCO(I,J) = 0.0</td>
<td>HBU00500</td>
</tr>
<tr>
<td>45</td>
<td>HRBCO(I,J) = 0.0</td>
<td>HBU00510</td>
</tr>
<tr>
<td>46</td>
<td>HPABH(I,J) = 0.0</td>
<td>HBU00520</td>
</tr>
<tr>
<td>47</td>
<td>HPABH(I,J) = 0.0</td>
<td>HBU00530</td>
</tr>
<tr>
<td>48</td>
<td>HPABH(I,J) = 0.0</td>
<td>HBU00540</td>
</tr>
<tr>
<td>49</td>
<td>HPA(I,J) = 0.0</td>
<td>HBU00550</td>
</tr>
<tr>
<td>50</td>
<td>HQ(I,J) = 0.0</td>
<td>HBU00560</td>
</tr>
<tr>
<td>51</td>
<td>HRH0(I,J) = 0.0</td>
<td>HBU00570</td>
</tr>
<tr>
<td>52</td>
<td>HPOT(I,J) = 0.0</td>
<td>HBU00580</td>
</tr>
<tr>
<td>53</td>
<td>HPBNT(I,J) = 0.0</td>
<td>HBU00590</td>
</tr>
<tr>
<td>54</td>
<td>HPBNT(I,J) = 0.0</td>
<td>HBU00600</td>
</tr>
<tr>
<td>55</td>
<td>NPO(I,J) = 0.0</td>
<td>HBU00610</td>
</tr>
<tr>
<td>56</td>
<td>NPM(I,J) = 0.0</td>
<td>HBU00620</td>
</tr>
<tr>
<td>57</td>
<td>NPBU(I,J) = 0.0</td>
<td>HBU00630</td>
</tr>
<tr>
<td>58</td>
<td>NPBBL(I,J) = 0.0</td>
<td>HBU00640</td>
</tr>
<tr>
<td>59</td>
<td>NRRN1U(I,J) = 0.0</td>
<td>HBU00650</td>
</tr>
<tr>
<td>60</td>
<td>NRRNL(I,J) = 0.0</td>
<td>HBU00660</td>
</tr>
<tr>
<td>61</td>
<td>NRRB1U(I,J) = 0.0</td>
<td>HBU00670</td>
</tr>
<tr>
<td>62</td>
<td>NRRB1L(I,J) = 0.0</td>
<td>HBU00680</td>
</tr>
<tr>
<td>63</td>
<td>NRRNOU(I,J) = 0.0</td>
<td>HBU00690</td>
</tr>
<tr>
<td>64</td>
<td>NRRNOL(I,J) = 0.0</td>
<td>HBU00700</td>
</tr>
<tr>
<td>65</td>
<td>NRRB0U(I,J) = 0.0</td>
<td>HBU00710</td>
</tr>
<tr>
<td>66</td>
<td>NRRB0L(I,J) = 0.0</td>
<td>HBU00720</td>
</tr>
<tr>
<td>67</td>
<td>NPB(I,J) = 0.0</td>
<td>HBU00730</td>
</tr>
<tr>
<td>68</td>
<td>NPBBL(I,J) = 0.0</td>
<td>HBU00740</td>
</tr>
<tr>
<td>69</td>
<td>NPB(I,J) = 0.0</td>
<td>HBU00750</td>
</tr>
<tr>
<td>70</td>
<td>NPA(I,J) = 0.0</td>
<td>HBU00760</td>
</tr>
<tr>
<td>71</td>
<td>NNO(I,J) = 0.0</td>
<td>HBU00770</td>
</tr>
<tr>
<td>72</td>
<td>NRRHO(I,J) = 0.0</td>
<td>HBU00780</td>
</tr>
<tr>
<td>73</td>
<td>NPOT(I,J) = 0.0</td>
<td>HBU00790</td>
</tr>
<tr>
<td>74</td>
<td>NPBUT(I,J) = 0.0</td>
<td>HBU00800</td>
</tr>
<tr>
<td>75</td>
<td>NPBBLT(I,J) = 0.0</td>
<td>HBU00810</td>
</tr>
<tr>
<td>76</td>
<td>OPQ(I,J) = 0.0</td>
<td>HBU00820</td>
</tr>
<tr>
<td>77</td>
<td>OPM(I,J) = 0.0</td>
<td>HBU00830</td>
</tr>
<tr>
<td>78</td>
<td>OPBU(I,J) = 0.0</td>
<td>HBU00840</td>
</tr>
<tr>
<td>79</td>
<td>OPBL(I,J) = 0.0</td>
<td>HBU00850</td>
</tr>
<tr>
<td>80</td>
<td>ORN1U(I,J) = 0.0</td>
<td>HBU00860</td>
</tr>
<tr>
<td>81</td>
<td>ORN1L(I,J) = 0.0</td>
<td>HBU00870</td>
</tr>
</tbody>
</table>
82   ORB1U(I,J) = 0.0  
83   ORB1L(I,J) = 0.0  
84   ORNOU(I,J) = 0.0  
85   ORNOL(I,J) = 0.0  
86   ORB0U(I,J) = 0.0  
87   ORB0L(I,J) = 0.0  
88   OPAN(I,J) = 0.0   
89   OPABU(I,J) = 0.0  
90   OPABL(I,J) = 0.0  
91   OPA1(I,J) = 0.0   
92   OQA1(I,J) = 0.0   
93   ORHOU(I,J) = 0.0  
94   ORHOL(I,J) = 0.0   
95   ORHO(I,J) = 0.0   
96   OPOT(I,J) = 0.0   
97   OPBUT(I,J) = 0.0  
98   OPBUTL(I,J) = 0.0 
99   PH(J) = 0.0       
100  PWH(J) = 0.0      
101  CONTINUE
102  10 CONTINUE
    C                              
    C *****************************************************
    C INITIALIZATION OF STARTING VALUES AT T = 0-
    C *****************************************************
    C                              
103  DO 16 K = 1,N
104    HQ(1,1) = R
105    HRHO(1,1) = R
106    HPOT(1,K) = 1.0
107    NPOT(1,K) = 1.0
108    OPOT(1,K) = 1.0
109    HPAT(1,K) = 1.0
110    NPAT(1,K) = 1.0
111    OPA1(K) = 1.0
112    HPAT1(K) = 1.0
113    NPAT1(K) = 1.0
114    OPA1(K) = 1.0
115  16 CONTINUE
    C *****************************************************
    C CALCULATION OF THE PROBABILITIES OF ACCESSING THE HOT AND
    C NON-HOT OUTPUT LINKS OF A HOT SWITCH AT ANY STAGE
    C *****************************************************
    C                              
116  T1 = 0.0
117  T2 = 0.0
118  DO 18 K = 1,N
119    T1 = (((2**(K-1))/(2**K)) - 1.0) * Q1 + Q
120    T2 = (((2**(K-1))/(2**K)) - 1.0) * Q1 + Q
121    PH(K) = T1 / T2
122    PWH(K) = 1.0 - PH(K)
123    WRITE (6,*), PH(K), PWH(K)
124  18 CONTINUE
    C *****************************************************
    C MAIN ROUTINE
    C *****************************************************
C HOTSW : SUBROUTINE FOR HOT SWITCHES AT ANY STAGE
C NONHOT : SUBROUTINE FOR NONHOT SWITCHES OF TYPE 1 AT ANY STAGE
C OTHER : SUBROUTINE FOR NONHOT SWITCHES OF TYPE 2 AT ANY STAGE
C ************************************************************************************
HBU01260
C THE PROGRAM PROCEEDS FROM THE LAST STAGE TO THE FIRST STAGE,
C CALCULATING ALL THE PROBABILITIES IN EACH STAGE (K) EXCEPT THE
C Q(.) AND RHQ(.) WHICH ARE CALCULATED WHILE EVALUATING THE
C PROBABILITIES FOR THE PREVIOUS STAGE (K-1).
C ************************************************************************************
HBU01260
C HBU01400
DO 30 1 = 2, 7
HBU01410
DO 25 K = N, 1, -1
HBU01420
127 CALL HOTSW(HPO, HPN, HPBH, HPBN, HRN1H, HRN1N, HRB1H, HRB1N, HRNOH, HBU01430
HRNOB, HRNOH, HPAN, HPBN, HPABN, HPA, HN, HN0H, HPOT, HBU01440
HPBH1, HPBN1, HPAB1, NPAB1, NPAB2, HPA1, HPA2, PH, HBU01450
PMH, R, 1, K, N) HBU01460
128 IF(K.GT.1) THEN HBU01470
129 CALL NONHOT(HPO, HPN, HPBH, HPBN, HRN1U, HRN1L, HRB1U, HRB1L, HRNOU, HBU01480
HRNOL, HRNOU, HRBOL, NPAB1, NPAB2, NPAB3, NPAB4, NQ, NRHO, HBU01490
NPOT, NPUB1, NPUBL, OPAN, OPBH, OPBA1, ORBH1, ORBH2, ORN1U, HBU01500
ORNL, ORNH1, ORN1L, ORBH1, ORBH2, ORN1U, ORNH1, ORNL, HBU01510
OPBH1, OPBH2, OPBL1, 1, K) HBU01520
130 ENDIF HBU01530
131 IF(K.EQ.N) THEN HBU01540
132 CALL OTHER(HPO, HPN, HPBH, HPBN, HRN1U, HRN1L, HRB1U, HRB1L, HRNOU, HBU01550
HRNOL, HRNOU, HRBOL, NPAB1, OPBH, OPAN, OPBH1, OPBH2, OPBL1, 1, K) HBU01560
133 ENDIF HBU01570
134 25 CONTINUE HBU01580
C ************************************************************************************
HBU01260
C HBW(1) CONTAINS THE DATARATE ON THE HOT OUTPUT OF THE
C HOT SWITCH AT THE FINAL STAGE WHILE HBW(I) CONTAINS THE
C DATARATE ON THE NON-HOT OUTPUT OF THE HOT SWITCH AT THE
C FINAL STAGE. HBW(I) CONTAINS THE DATARATE ON AN OUTPUT OF
C TYPE 1 SWITCH AT THE FINAL STAGE WHILE OBW(I) CONTAINS THE
C DATARATE ON AN OUTPUT OF TYPE 2 SWITCH AT THE FINAL STAGE.
C ************************************************************************************
HBU01260
C HBW(1) = 2.0*((HPW(1,3)*HRW1H(1,3)) + (HPW(1,3)*HRW1H(1,3)))
HBU01260
C HBW(1) = 2.0*((HPW(1,3)*HRW1N(1,3)) + (HPW(1,3)*HRW1N(1,3)))
HBU01260
C HBW(I) = 2.0*((HPW(1,3)*HRW1U(1,3)) + (NPB(1,3)*HRW1U(1,3)))
HBU01260
C OBW(I) = 2.0*((OPW(1,3)*ORW1U(1,3)) + (OPB(1,3)*ORW1U(1,3)))
HBU01260
139 30 CONTINUE HBU01590
140 DO 250 1 = 1, T
HBU01620
141 WRITE(3, 190) (HP(1, L), L = 1, N) HBU01630
142 WRITE(3, 191) (HP(1, L), L = 1, N) HBU01640
143 WRITE(3, 192) (HPBH1(1, L), L = 1, N) HBU01650
144 WRITE(3, 193) (HPBN(1, L), L = 1, N) HBU01660
145 WRITE(3, 194) (HRW1H(1, L), L = 1, N) HBU01670
146 WRITE(3, 195) (HRW1N(1, L), L = 1, N) HBU01680
147 WRITE(3, 196) (HRW1U(1, L), L = 1, N) HBU01690
148 WRITE(3, 197) (HRW1V(1, L), L = 1, N) HBU01700
149 WRITE(3, 198) (HRBH1(1, L), L = 1, N) HBU01710
150 WRITE(3, 199) (HRBH2(1, L), L = 1, N) HBU01720

FILE: HBUF2       LISTING A1       KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS, DHAHRAN
WRITE(3,200) (HRBON(I,L), L = 1,N)  
WRITE(3,201) (HRBON(I,L), L = 1,N)  
WRITE(3,202) (HPOT(I,L), L = 1,N)  
WRITE(3,203) (HPBNT(I,L), L = 1,N)  
WRITE(3,204) (HPBNT(I,L), L = 1,N)  
WRITE(3,205) (HPAM(I,L), L = 1,N)  
WRITE(3,206) (HPABH(I,L), L = 1,N)  
WRITE(3,207) (HPABH(I,L), L = 1,N)  
WRITE(3,208) (HPAM(I,L), L = 1,N)  
WRITE(3,209) (HQ(I,L), L = 1,N)  
WRITE(3,210) (HRHO(I,L), L = 1,N)  
WRITE(3,*)  
FORMAT (3X, 'PO=',3(2X, F6.4))  
FORMAT (3X, 'PM=',3(2X, F6.4))  
FORMAT (2X, 'PB=',3(2X, F6.4))  
FORMAT (2X, 'PBL=',3(2X, F6.4))  
FORMAT (1X, 'RHUI=',3(2X, F6.4))  
FORMAT (1X, 'RHI=',3(2X, F6.4))  
FORMAT (1X, 'RHUM=',3(2X, F6.4))  
FORMAT (1X, 'RHL=',3(2X, F6.4))  
FORMAT (1X, 'RBUI=',3(2X, F6.4))  
FORMAT (1X, 'RB=',3(2X, F6.4))  
FORMAT (1X, 'RP=',3(2X, F6.4))  
FORMAT (1X, 'PBL=',3(2X, F6.4))  
FORMAT (1X, 'PA=',3(2X, F6.4))  
FORMAT (1X, 'Q=',3(2X, F6.4))  
FORMAT (1X, 'RHOU=',3(2X, F6.4))  
CONTINUE  
WRITE (4,35) (HBWN(T) + HBNM(T) + (2.0 * NBW(T)) +  
(4.0 * OBW(T)))  
FORMAT (35,2X,F6.4)  
FORMAT (35, 'HBWN=',100(2X,F6.4))  
FORMAT (35, 'HBWN=',100(2X,F6.4))  
FORMAT (35, 'NBW=',100(2X,F6.4))  
FORMAT (35, 'OBW=',100(2X,F6.4))  
CONTINUE  
WRITE (4,*))  
STOP  
END  
**************************************************************************  
SUBROUTINE HOTSM(HPO, HPN, HPBH, HPBN, HRN1H, HHR11, HRB11, HHR1N, HRHO, HPO,  
1 HRMON, HRBOH, HRBON, HPA, HPBN, HPABH, HPABN, HPA, HRHO, HPO,  
1 HPBH, HPBN, HPA, HPABH, HPABN, HPA, HRHO, NPABL, HQ, NPA, PH,  
1 PHR, R, I, K, N)  
C  
C
REAL HP(100,5),HPN(100,5),HPBH(100,5),HRNH(100,5),
HRBH(100,5)

194 REAL HPBN(100,5),HRBN(100,5),HRBH(100,5),HRBN(100,5)
195 REAL HRBN(100,5),HRBN(100,5),HRBH(100,5),HPAB(100,5)
196 REAL HPBN(100,5),HP(100,5),HRBH(100,5),HRBN(100,5),
197 HPOT(100,5)
198 REAL HPBN(100,5),HPBN(100,5),HRBH(100,5),HRBN(100,5),
199 REAL PH(5),PNH(5),R,HRBH(100,5)
200 REAL TEMPC,TEMP5,TEMP6,TEMPC,TEMP8,TEMP9,TEMP2,TEMP3,TEMP4,TEMP5,TEMP6,
201 REAL TEMPC1,TEMPC2,TEMPC3,TEMP4,TEMP5,TEMP6,TEMP7,TEMP8,
202 INTEGER I,K

C
203 TEMP2 = 0.0
204 TEMP3 = 0.0
205 TEMP4 = 0.0
206 TEMP5 = 0.0
207 TEMP6 = 0.0
208 TEMP7 = 0.0
209 TEMP8 = 0.0
210 TEMP9 = 0.0
211 TEMP10 = 0.0
212 TEMP11 = 0.0
213 TEMP12 = 0.0
214 TEMP13 = 0.0
215 TEMP14 = 0.0
216 TEMP15 = 0.0
217 TEMP16 = 0.0
218 TEMP17 = 0.0
219 TEMP18 = 0.0
220 TEMP19 = 0.0

C
C CALCULATION OF STATE PROBABILITIES
C
221 HP(100,K) = HPOT(1-1,K) * (1.0 - HQ(1-1,K))
222 HPN(100,K) = HPOT(1-1,K) * HPN(100,K)
223 HPBH(100,K) = HPBN(1-1,K)
224 HPBN(100,K) = HPBN(1-1,K)

C
C CALCULATION OF THE TRANSITION PROBABILITIES. CARE IS TAKEN
C REGARDING THE CONDITIONS OF THE NETWORK IN THE INITIAL
C CYCLES: FOR EXAMPLE, THE TRANSITION PROBABILITY WHEN THE
C PACKET IS IN A BLOCKED STATE COMES INTO EFFECT ONLY AFTER
C CYCLE NUMBER 3. WE HAVE DIFFERENT FORMULAE FOR THE FINAL
C STAGE. ALL TEMPS ARE TEMPORARY VARIABLES.
C
225 IF (K.EQ.N) THEN
226 IF (I.GT.3) THEN
227 HRNH(1,K) = PH(K) - (0.5 * (PH(K)**2) * HPN(1,K)) - (0.5 *
1 PH(K) * HPBH(1,K))
228 HRBN(1,K) = PNH(K) - (0.5 * (PNH(K)**2) * HPN(1,K)) - (0.5 *
1 PNH(K) * HPBN(1,K))
HROH1(K) = PH(K) - HROH1(K)
HROK(K) = PH(K) - HROK1(K)
IF (I.GT.4) THEN
HROH1(K) = 1.0 - (0.5 * PH(K) * HPW(1,K))
HROH1(K) = 1.0 - (0.5 * PHH(K) * HPW(1,K))
ELSE IF (K.EQ.2) AND (K.EQ.1) THEN
HROH1(K) = (PH(K) * HPW(1,K)) - (0.5 * (PH(K) ** 2) * HPW(1,K))
HROH2(K) = (PH(K) * HPW(1,K)) - (0.5 * (PHH(K) ** 2) * HPW(1,K))
I = (PHH(K) * HPO(1,K))
1 = (PH(H) * HPO(1,K))
HROH1(K) = PH(K) - HROH1(K)
HROH1(K) = PH(K) - HROH1(K)
GO TO 45
ENDIF
END IF
C
TEMP4 = PH(K) * (((1.0 - (0.5 * PH(K))) * HPW(1,K)) + HPO(1,K) + 1 * HPW1(K) + HPW(1,K))
C
TEMP15 = (1.0 - (0.5 * HROH1(-1,K+1))) * PH(K+1)
C
TEMP16 = (1.0 - (0.5 * HROH1(-1,K+1))) * PHH(K+1)
C
TEMP5 = (HPW1(K+1) * HROH1(-1,K+1) * 0.5) + (HPAB1(1,K) * 1 * TEMP15 + HPABB(1,K+1) * TEMP16)
C
IF (K.EQ.1) THEN
TEMPA = 1.0 - HPA(1,K+2) + (PH(K+1) ** 2)
TEMPB = 1.0 - NPA(1,K+2) + (PMH(K+1) ** 2)
ELSE
TEMPA = PH(K+1) ** 2
TEMPB = PMH(K+1) ** 2
ENDIF
C
TEMP16 = HPAH1(K+1) * (1.0 - (PHH(K) * HHHO(1-1,K+1))) * (HPA1(K+1) + (HPW1(K+1) + HPW(1,K)) / HPW(1,K))
C
TEMP17 = HPAH1(K+1) * (1.0 - (PHH(K) * HHHO(1-1,K+1))) * (HPA1(K+1) + (HPW1(K+1) + HPW(1,K))/ HPW(1,K))
C
TEMP15 = HPB1(K+1) * 0.5 * HHHO(1-1,K+1)
C
TEMP18 = TEMP15 + TEMP16 + TEMP17
C
HRN1H1(K) = TEMP4 + (0.5 * PHH(K) * HPBB1(K) * TEMP18)
C
HRN1H1(K) = PH(K) - HRN1H1(K)
C
TEMP6 = PMH(K) * (((1.0 - (0.5 * PMH(K))) * HPW(1,K)) + HPO(1,K) + 1 * HPW1(K) + HPB1(K+1))
C
TEMP7 = (HPB1(K+1) * HHHO(1-1,K+1) * 0.5) + (HPADD(1,K+1) + (HPADD(1,K+1) + 1 * HHHO(1-1,K+1) / 2.0))
C
HRN1H1(K) = TEMP6 + (0.5 * PHH(K) * HPBB1(K) * TEMP7)
C
HRN1H1(K) = PMH(K) - HRN1H1(K)
C
IF ((1.EQ.3) AND (K.EQ.2)) GO TO 45
C
TEMP8 = (HPB1(K+1) * (1.0 - (0.5 * PHH(K))) + HPO(1,K)
C
TEMP9 = (0.5 * HPBB1(K) + HPW(1,K))
C
ENDIF
C
ENDIF
C TEMP10 = TEMP16 + TEMP17
267 HRB1H(I,K) = (TEMP10*TEMP8) + (TEMP10*TEMP9)
268 HRB0H(I,K) = 1.0 - HRB1H(I,K)
C TEMPP1 = (HPN(I,K) * (1.0 - (0.5*HPN(K)))) + HPO(I,K)
270 TEMPP2 = (0.5 * HPBN(I,K)) + HPBN(I,K)
271 TEMPP3 = ((NPABU(I,K+1) + NPABL(I,K+1)) * (1.0 - (0.5 * HPBN(I,K+1))))
1 272 HRHO(1-I,K+1)) / 2.0)
C 273 HRBN(I,K) = (TEMP1*TEMP7) + (TEMP12*TEMP3)
274 HRBN(I,K) = 1.0 - HRBN(I,K)
275 45 CONTINUE
276 ENDIF
C 
C ******************************************************************************
C CALCULATION OF INTERMEDIATE STATE PROBABILITIES AND THE
C ACCEPTANCE PROBABILITIES. IF IT IS NOT THE FINAL STAGE, THEN
C THE RHO ON THE INPUT LINK OF THE SWITCH AT THE SUCCEEDING
C STAGE IS CALCULATED. THE OFFERED LOAD TO THE SUCCEEDING STAGE
C IS ALSO CALCULATED. IF IT IS THE FIRST STAGE, SINCE THE LOAD
C IS KNOWN, ONLY THE EFFECTIVE LOAD ( RHO ) IS CALCULATED.
C ******************************************************************************
C 
C 277 TEMP2 = (HPBN(I,K)*HRBN1H(I,K)) + (HPBN(I,K)*HRBN1H(I,K))
278 TEMP3 = HPN(I,K) * (HRN1H(I,K) + HRN1H(I,K))
279 HPOT(I,K) = HPO(I,K) + TEMP2 + TEMP3
280 HPBTH(I,K) = (HPN(I,K)*HRN0H(I,K)) + (HPBN(I,K)*HRB0H(I,K))
281 HPBTH(I,K) = (HPN(I,K)*HRN0H(I,K)) + (HPBN(I,K)*HRB0H(I,K))
282 HPA(I,K) = HPOT(I,K)
283 HPAN(I,K) = HRN1H(I,K) + HRN1H(I,K)
284 HPABH(I,K) = HRBN1H(I,K)
285 HPABM(I,K) = HRBN1H(I,K)
C 
C 286 IF (K.NE.N) THEN
287 HRHO(I,K+1) = 2.0 * ((HPN(I,K) * HRN1H(I,K)) + (HPBN(I,K) *
1 HRN1H(I,K))
288 HRHO(I,K+1) = 2.0 * ((HPN(I,K) * HRN1H(I,K)) + (HPBN(I,K) *
1 HRBN1H(I,K)))
289 HQ(I,K+1) = HRHO(I,K+1) / HPA(I,K+1)
290 HQ(I,K+1) = HRHO(I,K+1) / NPA(I,K+1)
291 ENDIF
C 
C 292 IF (K.EQ.1) THEN
293 HQ(I,K) = R
294 HRHO(I,K) = HPA(I,K) * HQ(I,K)
295 ENDIF
C 
C 296 RETURN
297 END
C ******************************************************************************
C SUBROUTINE NONHOT(NPO,NPN,NPBU,NPBL,NRN1U,NRN1L,HRB1U,HRB1L,
0 1 NRN1U,NRN1L,HRB0U,HRB0L,NPAH,NPABU,NPABL,NPA,NRHO,
1 NPOT,NPAN,DPAN,ORA,B1,OPBL,OPAN,ORH0,ORH0L,QQ,OPA,
1 ORHO,1,K,N)
C 
C HBU03390
C HBU03400
C HBU03410
C HBU03420
C HBU03430
C HBU03440
C HBU03450
C HBU03460
C HBU03470
C HBU03480
C HBU03490
C HBU03500
C HBU03510
C HBU03520
C HBU03530
C HBU03540
C HBU03550
C HBU03560
C HBU03570
C HBU03580
C HBU03590
C HBU03600
C HBU03610
C HBU03620
C HBU03630
C HBU03640
C HBU03650
C HBU03660
C HBU03670
C HBU03680
C HBU03690
C HBU03700
C HBU03710
C HBU03720
C HBU03730
C HBU03740
C HBU03750
C HBU03760
C HBU03770
C HBU03790
C HBU03800
C HBU03810
C HBU03820
C HBU03830
C HBU03840
C HBU03850
REAL NPO(100,5), NPM(100,5), NPBU(100,5), NPBL(100,5),
INNW(100,5), HBU03860
REAL NRN1L(100,5), NRBU(100,5), NRBL(100,5), NRN0U(100,5),
HBU03860
REAL NRNOL(100,5), NRB0U(100,5), NRB1L(100,5), NPAN(100,5),
HBU03890
INPABL(100,5) HBU03900
1 REAL NPABU(100,5), NPA(100,5), NQ(100,5), NRHO(100,5)
HBU03910
REAL NPOT(100,5), NPBLT(100,5), INPBT(100,5) HBU03920
REAL ORPA(100,5), ORPH(100,5), ORPHU(100,5), ORPAH(100,5),
HBU03930
REAL ORHOL(100,5), ORQ(100,5), ORPA(100,5), ORHO(100,5)
HBU03940
REAL T1, T2, T3, T4, T5, T6, T7, T8, T9, T10 HBU03950
307 INTEGER 1, K HBU03960
308 T1 = 0.0 HBU03970
309 T2 = 0.0 HBU03980
310 T3 = 0.0 HBU03990
311 T4 = 0.0 HBU04000
312 T5 = 0.0 HBU04010
313 T6 = 0.0 HBU04020
314 T7 = 0.0 HBU04030
315 T8 = 0.0 HBU04040
316 T9 = 0.0 HBU04050
317 T10 = 0.0 HBU04060
318 HBU04070
C ---------- CALCULATION OF STATE PROBABILITIES ----------
C ----------------------------------------------------------
319 HBU04080
320 NPO(1,K) = NPOT(I-1,K) * (1.0 - NQ(I-1,K)) HBU04090
321 NPM(1,K) = NPOT(I-1,K) * NQ(I-1,K) HBU04100
322 NPBU(1,K) = NPBUT(I-1,K) HBU04110
323 NPBL(1,K) = NPBLT(I-1,K) HBU04120
C ---------- CALCULATION OF TRANSITION PROBABILITIES ----------
C ----------------------------------------------------------
324 HBU04130
IF (K.EQ.N) THEN
325 IF (I.GT.3) THEN
326 NRN1U(1,K) = 0.5 - (0.125 * NPM(1,K)) - (0.25 * NPBU(1,K)) HBU04140
327 NRM0U(1,K) = 0.5 - NRM1L(1,K) HBU04150
328 IF (I.GT.4) THEN
329 NRB1L(1,K) = 1.0 - (0.25 * NPM(1,K)) HBU04160
330 NRB0U(1,K) = 1.0 - (0.25 * NPM(1,K)) HBU04170
331 NRB0U(1,K) = 1.0 - NRB1L(1,K) HBU04180
332 NRBOL(1,K) = NRB0U(1,K) HBU04190
333 ELSE
334 ENDIF
335 ENDIF
336 IF (I.GT.2) THEN
337 T1 = 0.375 * NPM(1,K) * OPA(I,K+1) HBU04190
338 T2 = 0.5 * OPA(I,K+1) + NPO(I,K) HBU04200
339 T3 = (ORHOU(I-1,K+1) + ORHOL(I-1,K+1)) / 4.0 HBU04210
340 T4 = (OPAN(I,K+1) + T3) + ((OPABU(I,K+1) * OPABL(I,K+1)) *
1 ((1.0 - T3)/2.0))
341 T5 = 0.5 * OPA(1,K+1) * NPBL(1,K)  
342 T6 = 0.5 * OPA(1,K+1) * NPBU(1,K)  
343 NRNU1(1,K) = T1 + T2 + T5 + (0.25 * NPBU(1,K) * T4)  
344 NRNL1(1,K) = T1 + T2 + T6 + (0.25 * NPBL(1,K) * T4) 
345 NRNOU(1,K) = 0.5 - NRNU1(1,K)  
346 NRNOL(1,K) = 0.5 - NRNL1(1,K)  
347 IF (((1.EQ.3).AND.(K.EQ.2)) .OR. (1)) GO TO 60  
348 T7 = (((1.0-T3)/2.0) * (OPABU(1,K+1) + OPABL(1,K+1))) 
349 T8 = T4 + (0.75 * NPBM(1,K)) + NPO(1,K)  
350 NRBU1(1,K) = T8 + (T7 * ((0.5*NPBU(1,K) + NPBL(1,K))))  
351 NRBL1(1,K) = T8 + (T7 * ((0.5*NPBL(1,K) + NPBU(1,K))))  
352 NRBOU(1,K) = 1.0 - NRBU1(1,K)  
353 NRBOL(1,K) = 1.0 - NRBL1(1,K)  
354 60 CONTINUE  
355 ENDIF  
356 ENDIF  
357 C****************************************************************************** 
358 C CALCULATION OF INTERMEDIATE STATE PROBABILITIES AND THE 
359 C ACCEPTANCE PROBABILITIES. 
360 C******************************************************************************  
361 T9 = NPBM(1,K) * (NRNU1(1,K) + NRNL1(1,K))  
362 T10 = (NPBU(1,K)*NRBU1(1,K)) + (NPBL(1,K)*NRBL1(1,K))  
363 NPOT(1,K) = NPO(1,K) + T9 + T10  
364 NPBU(1,K) = (NPBM(1,K)*NRNOU(1,K)) + (NPBU(1,K)*NRBOU(1,K))  
365 NPBL(1,K) = (NPBM(1,K)*NRNOL(1,K)) + (NPBL(1,K)*NRBOL(1,K))  
366 IF (K.NE.N) THEN  
367 ORHOU(1,K+1) = 2.0 * ((NPBM(1,K) * NRNU1(1,K)) + 
368 1 (NPBU(1,K) * NRBU1(1,K)))/NPOT(1,K)  
369 ORHOL(1,K+1) = 2.0 * ((NPBM(1,K) * NRNL1(1,K)) + 
370 1 (NPBL(1,K) * NRBL1(1,K)))/NPOT(1,K)  
371 ENDIF 
372 C******************************************************************************  
373 RETURN  
374 END 
375 C******************************************************************************  
376 C SUBROUTINE OTHER(OPO,OPM,OPBU,OPBL,ORN1U,ORN1L,ORB1U,ORB1L, 
377 1 ORHOU,ORHOL,ORB1U,ORB1L,OPAN,OPABU,OPABL,OPA,OQ,ORHOU, 
378 1 ORHOL,OPOT,OPBU,OPBLT,1,K) 
379 C******************************************************************************  
380 C REAL OPO(100,5),OPM(100,5)  
381 C REAL OPBU(100,5),OPBL(100,5),ORN1U(100,5),ORN1L(100,5), 
382 C ORB1U(100,5), 
383 C C REAL ORN1L(100,5),ORB1L(100,5),ORN1(100,5),ORB1(100,5), 
384 C ORB1(100,5) 
385 C******************************************************************************
REAL OPAM(100,5), OPABU(100,5), OPBUT(100,5), ORHOU(100,5)
REAL OPA(100,5), OQA(100,5), ORHOL(100,5),
TOP(100,5)
REAL OPBLT(100,5), TEMP19, TEMP20
INTEGER I, K

TEMP19 = 0.0
TEMP20 = 0.0

C
C ***********************************************************************
C CALCULATION OF STATE PROBABILITIES
C ***********************************************************************
C
384     OP0(I,K) = OP0(I-1,K) * (1.0 - OQ(I-1,K))
385     OPH(I,K) = OP0(I-1,K) * OQ(I-1,K)
386     OPBUI(K) = OPBUT(I-1,K)
387     OPBLI(K) = OPBLT(I-1,K)
C
C ***********************************************************************
C CALCULATION OF TRANSITION PROBABILITIES
C ***********************************************************************
C
388     IF (1.GT.3) THEN
389         ORHIU(I,K) = 0.5 - (0.125 * OPH(I,K)) - (0.25 * OPBUI(K))
390         ORHL1(I,K) = 0.5 - (0.125 * OPH(I,K)) - (0.25 * OPBLI(K))
391         ORHOU(I,K) = 0.5 - ORHIU(I,K)
392         ORHOL(I,K) = 0.5 - ORHL1(I,K)
393     IF (1.GT.3) THEN
C
394         ORBIU(I,K) = 1.0 - (0.25 * (1.0 - OPH(I,K)))
395         ORBIL1(I,K) = 1.0 - (0.25 * (1.0 - OPH(I,K)))
396         ORB1U(I,K) = 1.0 - (0.25 * OPH(I,K))
397         ORB1L(I,K) = 1.0 - (0.25 * OPH(I,K))
398         ORB0U(I,K) = ORBIU(I,K)
399         ORB0L(I,K) = ORB0U(I,K)
        ENDIF
        ENDIF
C
C ***********************************************************************
C CALCULATION OF INTERMEDIATE STATE PROBABILITIES AND THE
C ACCEPTANCE PROBABILITIES.
C ***********************************************************************
C
400     TEMP19 = (OPBU(I,K)*ORB1U(I,K)) + (OPBL(I,K)*ORB1L(I,K))
401     TEMP20 = OPM(I,K) * (ORHIU(I,K) + ORHL1(I,K))
402     OP0(I,K) = OPM(I,K) + TEMP19 + TEMP20
403     OPBUT(I,K) = (OPM(I,K)*ORHOU(I,K)) + (OPBU(I,K)*ORB0U(I,K))
404     OPBLT(I,K) = (OPM(I,K)*ORHOL(I,K)) + (OPBL(I,K)*ORB0L(I,K))
405     OPA(I,K) = OP0(I,K)
406     OPAM(I,K) = ORHIU(I,K) + ORHL1(I,K)
407     OPABU(I,K) = ORBIU(I,K)
408     OPABL(I,K) = ORB1L(I,K)
C
C ***********************************************************************
C
409     RETURN
410     END
C ***************************************************************
C PROGRAM TO COMPUTE THE BANDWIDTH USING MODEL 2. PREFIX H
C IDENTIFIES THE HOT SWITCH WHILE PREFIX N IDENTIFIES SWITCH
C OF TYPE 1 and 0 OF TYPE 2. THE OTHER POINTS OUTLINED AS
C COMMENTS IN MODEL 3 HOLD HERE ALSO.
C ***************************************************************
C
1 REAL HP(15,5), HPN(15,5), HPB(15,5), HRR1(15,5), HRR1N(15,5)
2 REAL HRB1(15,5), HRB1N(15,5), HRR0(15,5), HRBO(15,5), HPAN(15,5)
3 REAL HPAB(15,5), HPA(15,5), HQ0(0:15,5), HRRH0(0:15,5), HPPOT(0:15,5)
4 REAL HPBT(0:15,5), NHPO(15,5), NHPN(15,5), NHPB(15,5), NHRR1(15,5)
5 REAL HHAB(15,5), HHAN(15,5), HHBO(15,5), HHAN(15,5)
6 REAL NHAB(15,5), NHPA(15,5), HQO(0:15,5), HHRH0(0:15,5)
7 REAL NHPPOT(0:15,5), NHPPBT(0:15,5), OP0(15,5), OPO(15,5), OPB(15,5)
8 REAL ORN(15,5), ORB1(15,5), ORNO(15,5), ORBO(15,5), OPAN(15,5)
9 REAL OPAB(15,5), OPA(15,5), OQ0(0:15,5), ORRO(0:15,5), OPOT(0:15,5)
10 REAL OPBT(0:15,5), PH(5), PHK(5), Q, R, Q1, T1, T2
11 REAL H, NH1(15,5), NHM1(15,5), NHMM1(15,5), NHBM(15,5)
12 INTEGER I, J, K, N, T
C
13 T = 15
14 N = 3
15 DO 300 R = 0.1, 1.01, 0.1
16 Q = 0.3
17 Q1 = (1.0 - Q)/(2**N - 1)
18 EQ = Q * R
19 EQ1 = Q1 * R
C
20 DO 10 I = 1, T
21 DO 9 J = 1, N
22 HP(1, J) = 0.0
23 HPM(1, J) = 0.0
24 HPB(1, J) = 0.0
25 HRR1(1, J) = 0.0
26 HRR1N(1, J) = 0.0
27 HRB1(1, J) = 0.0
28 HRB1N(1, J) = 0.0
29 HRR0(1, J) = 0.0
30 HRBO(1, J) = 0.0
31 HPA(1, J) = 0.0
32 HPAB(1, J) = 0.0
33 HPA(1, J) = 0.0
34 HQ(1, J) = 0.0
35 HRRH0(1, J) = 0.0
36 HPPOT(1, J) = 0.0
37 HPBT(1, J) = 0.0
38 NHPO(1, J) = 0.0
39 NHPPN(1, J) = 0.0
40 NHPB(1, J) = 0.0
41 NHRR1(1, J) = 0.0
42 NHRR1N(1, J) = 0.0
FILE: HBUF1        LISTING A1      KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS, DHAHRAH

43    MHRNO(1,J) = 0.0
44    MHRBO(1,J) = 0.0
45    NHPAM(1,J) = 0.0
46    NHPAB(1,J) = 0.0
47    NHPA(1,J) = 0.0
1
48    NMQ(1,J) = 0.0
49    NHRHO(1,J) = 0.0
50    NHPOT(1,J) = 0.0
51    NHPBT(1,J) = 0.0
52    OPO(1,J) = 0.0
53    OPM(1,J) = 0.0
54    OPB(1,J) = 0.0
55    ORN1(1,J) = 0.0
56    ORB1(1,J) = 0.0
57    ORNO(1,J) = 0.0
58    ORBO(1,J) = 0.0
59    OPAM(1,J) = 0.0
60    OPAB(1,J) = 0.0
61    OPA(1,J) = 0.0
62    OQ(1,J) = 0.0
63    ORHO(1,J) = 0.0
64    OPOT(1,J) = 0.0
65    OPBT(1,J) = 0.0
66    HRHA(1,J) = 0.0
67    HBRH(1,J) = 0.0
68    NHEN(1,J) = 0.0
69    NHBM(1,J) = 0.0
70    PHK(J) = 0.0
71    PWHK(J) = 0.0
72   9    CONTINUE
73  10    CONTINUE
C
C **************************************************************
C C INITIALIZATION OF APPROPRIATE VALUES
C
74   DO 15 K = 1, N
75      HPOT(0,K) = 1.0
76      HQ(0,K) = 0.0
77      HPBT(0,K) = 0.0
78      NRHO(0,K) = 0.0
79      NHPOT(0,K) = 1.0
80      NHQ(0,K) = 0.0
81      NHPBT(0,K) = 0.0
82      NHRHO(0,K) = 0.0
83      OPOT(0,K) = 1.0
84      OQ(0,K) = 0.0
85      OPBT(0,K) = 0.0
86      ORHO(0,K) = 0.0
87  15    CONTINUE
88   BW = 0.0
C
C **************************************************************
C C PROBABILITY OF ACCESSING A HOT AND A NON-HOT OUTPUT LINK OF
C A HOT SWITCH FOUND OUT
C
DO 18 K = 1, N
T1 = (((2**(K-1)) - 1.0) * Q1) + Q
T2 = (((2**(K-1)) - 1.0) * Q1) + Q
PHK(K) = T1 / T2
PHNK(K) = 1.0 - PHK(K)
WRITE (6,*) PHK(K), PHNK(K)
18 CONTINUE
C
C ****************************************************************************
C MAIN ROUTINE
C
DO 30 I = 1, T
DO 25 K = N, 1, -1
1 CALL HOATSW(HPO, HPN, HPB, HRN1, HRN1N, HRB1, HRB1N, HRNO, HRBO, HPAN,
HPAB, HPH, HO, HRO, HROT, HPS, HPSI, HPSB, HPSB1, HPSB2, HPSB3, HPSB4,
* PHNK, R, I, K, N)
2 IF(K .LT. 1) THEN
3 CALL MOMHOT(HPO, HPN, HPB, HRN1M, HRN1, HRB1M, HRB1, HRNO, HRBO, HPAM,
HPAB, HPH, HRO, HROT, HPS, HPSI, HPSB, HPSB1, HPSB2, HPSB3, HPSB4,
* R, I, K, N)
4 ENDIF
5 IF(K .EQ. N) THEN
6 CALL OTHER(OP0, OPN, OPB, ORN1, ORN1D, ORN2, ORN2D, ORN3, ORN3D,
OPB, OPA, OQ, ORH0, OPOT, OPBT, R, I, K, N)
7 ENDIF
8 HBWN(1,3) = 2.0 * ((HPN(1,3) * HRN1(1,3)) + (HPB(1,3)
9 * HRB1(1,3)))
10 HBWNH(1,3) = 2.0 * ((HPN(1,3) * HRN1N(1,3)) + (HPB(1,3)
11 * HRB1N(1,3)))
12 NHBN(1,3) = 2.0 * ((HPS(1,3) * HRN1(1,3)) + (HPSB(1,3)
13 * HRB1(1,3)))
14 NHBNH(1,3) = 2.0 * ((HPS(1,3) * HRN1N(1,3)) + (HPSB(1,3)
15 * HRB1N(1,3)))
16 BW = HBWN(T,3) + HBWNH(T,3) + (2.0 * NHBN(T,3))
17 + (4.0*NHBNH(T,3))
18 DO 40 K = 1, N
19 WRITE (3,35) (HRNO(1,K), I = 1, T)
20 WRITE (3,37) (HHRNO(1,K), I = 1, T)
21 WRITE (3,38) (ORNO(1,K), I = 1, T)
22 WRITE (3,35) (HBWN(1,I), I = 1, T)
23 WRITE (3,35) (HBWNH(1,I), I = 1, T)
24 WRITE (3,37) (NHBN(1,I), I = 1, T)
25 WRITE (3,38) (NHBNH(1,I), I = 1, T)
26 WRITE (3,38) (NHBN(1,I), I = 1, T)
27 WRITE (3,35) BW
28 FORMAT (2X,F6.4)
35 FORMAT (2X,15(2X,F6.4))
37 FORMAT (2X,15(2X,F6.4))
38 FORMAT (2X,15(2X,F6.4))
39 CONTINUE
300 CONTINUE
30 WRITE (3,*) '
31 STOP
32 END
SUBROUTINE HOTSN(HPO, HPM, HPB, HRN1, HRN1N, HRB1, HRB1N, HRO, HRBO, HPAN,
               HPAB, HPA, HQ, HPAO, HPOT, HPBT, NHPAN, NHPAB, NHRNO, NHRBO, NHQ, NHPA, PHK,
               PHK,R, I, K, N)

REAL HPO(15,5), HPM(15,5), HPB(15,5), HRN1(15,5), HRN1N(15,5)
REAL HRB1(15,5), HRB1N(15,5), HRO(15,5), HRBO(15,5), HPAN(15,5)
REAL HPAB(15,5), HPA(15,5), HQ(0:15,5), HRNO(0:15,5), HPOT(0:15,5)
REAL HPBT(0:15,5), NHPAN(15,5), NHPAB(15,5), NHQ(0:15,5)
REAL PHK(5), PHK(R), HRBO(0:15,5)
INTEGER I, K

TEMP1 = 0.0
TEMP2 = 0.0
TEMP3 = 0.0
TEMP4 = 0.0
TEMP5 = 0.0
TEMP6 = 0.0
TEMP7 = 0.0
TEMP8 = 0.0
TEMP9 = 0.0

HPO(I,K) = HPO(I-1,K) * (1.0 - HQ(I-1,K))
HPM(I,K) = HPOT(I-1,K) * HQ(I-1,K)
HPB(I,K) = HPBT(I-1,K)
TEMP1 = HPM(I,K) + HPB(I,K)
TEMP2 = PHK(K) * (1.0 - (PHK(K) * 0.5 * TEMP1))
TEMP3 = PHK(K) * (1.0 - (PHK(K) * 0.5 * TEMP1))

IF (I.NE.1) THEN
  IF (K.EQ.M) THEN
    HRN1(I,K) = TEMP2
    HRN1N(I,K) = TEMP3
    HRNO(I,K) = 1.0 - HRN1(I,K) - HRN1N(I,K)
  ELSE
    HRB1(I,K) = TEMP2
    HRB1N(I,K) = TEMP3
    HRBO(I,K) = 1.0 - HRB1(I,K) - HRB1N(I,K)
  ENDIF
ENDIF
ELSE
  IF (I.EQ.2) THEN
    HRB1(I,K) = 0.0
    HRB1N(I,K) = 0.0
  ELSE
    HRN1(I,K) = TEMP2
    HRN1N(I,K) = TEMP3
    HRNO(I,K) = 1.0 - HRN1(I,K) - HRN1N(I,K)
  ENDIF
FILE: HBUF1  LISTING A1  KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS, DHAHRAN

162 ELSE
163 TEMP4 = (HPAN(1,K+1) * 0.5*HRHO(1-1,K+1)) + (HPAB(1,K+1) *
1 (1.0 - 0.5*HRHO(1-1,K+1))
164 TEMP5 = (NHFPN(1,K+1) * 0.5*HRNO(1-1,K+1)) + (NHAB(1,K+1) *
1 (1.0 - 0.5*HRNO(1-1,K+1))
165 HRMN(1,K) = TEMP2 * HPAB(1,K+1)
166 HRBN(1,K) = TEMP3 * NPAB(1,K+1)
167 HRNO(1,K) = 1.0 - (HRMN(1,K) + HRBN(1,K))
168 IF ((1.EQ.3) .AND. (K.EQ.2)) GO TO 55
169 IF (HRNO(1,K) < 0.5) THEN
170 IF (HRNO(1,K) < 0.5) THEN
171 TEMP6 = PHK(K) * (1.0 - 0.5 * PHK(K)) * HPB(1,K) *
172 NHFPN(1,K+1) * 0.5*HRNO(1-1,K+1)
173 HRBN(1,K) = (TEMP2 * TEMP4) - TEMP6
174 CONTINUE
175 ENDIF
176 ENDIF
177 ENDIF
178 ENDIF
179 ENDIF
180 HPOT(1,K) = HPOT(1,K) + TEMP8 + TEMP9
181 HPB(1,K) = (HPN(1,K) * HRNO(1,K)) + (HPB(1,K) * HRNO(1,K))
182 HPA(1,K) = HPOT(1,K)
183 HPAN(1,K) = HRBN(1,K) + HRBN(1,K)
184 HPAB(1,K) = HRBN(1,K) + HRBN(1,K)
185 IF (K.NE.1) THEN
186 HRNO(1,K+1) = 2.0 * ((HPN(1,K) * HRBN(1,K)) + (HPB(1,K) *
1 HB1(1,K)))
187 HRNO(1,K+1) = 2.0 * ((HPN(1,K) * HRBN(1,K)) + (HPB(1,K) *
1 HB1(1,K)))
188 HQ(1,K+1) = HRNO(1,K+1) / HPA(1,K+1)
189 HQ(1,K+1) = NRHRHO(1,K+1) / NHFPN(1,K+1)
190 ENDIF
191 IF (K.EQ.1) THEN
192 HPA(1,K) = R
193 HQ(1,K) = HPA(1,K) * HQ(1,K)
194 ENDIF
195 RETURN
196 END
C ******************************************************************************
C SUBROUTINE FOR A SWITCH OF TYPE 1
C
0 197 SUBROUTINE NONHOT(NHPO, NHPN, NHPB, NRHR1, NRHRNO, NRHRBO, NHFPN,
1 1 KHPAB, NHPA, NHQ, HRHRHO, NHPNBT, HPOT, OPA, ORHO, OQ, OPA,
1 * R, 1, K, N)
C
198 REAL NHPO(15,5), NHPN(15,5), NHPB(15,5), NRHR1(15,5)
199 REAL NRHR1(15,5), NRHRNO(15,5), NRHRBO(15,5), NHFPN(15,5)
200 REAL NHFPN(15,5), NHPA(15,5), NHQ(0:15,5), HRHRHO(0:15,5)
REAL NHPOT(0:15,5),NHPBT(0:15,5),OPAN(15,5),OPAB(15,5)
REAL ORHO(0:15,5),OQ(0:15,5),OPA(15,5),R
REAL TEMPI2,TEMP13,TEMP14,TEMP15,TEMP16,TEMP17
INTEGER I,K

C
TEMP13 = 0.0
TEMP14 = 0.0
TEMP15 = 0.0
TEMP16 = 0.0
TEMP17 = 0.0
TEMP12 = 0.0

C
NHPOT(I,K) = NHPOT(I-1,K) * (1.0 - ORHO(I-1,K))
NHPM(I,K) = NHPOT(I-1,K) * ORHO(I-1,K)
NHPB(I,K) = NHPBT(I-1,K)
TEMP12 = NHPM(I,K) + NHPB(I,K)
TEMP13 = 0.5 - (0.125 * TEMP12)

C
IF (.NE.1) THEN
IF (K.EQ.I) THEN
IF (I.GT.3) THEN
NHRN1(I,K) = TEMP13
NHRNO(I,K) = 1.0 - (2.0 * NHRN1(I,K))
IF (I.GT.4) THEN
NHRB1(I,K) = TEMP13
NHRBO(I,K) = 1.0 - (2.0 * NHRB1(I,K))
ENDIF
ENDIF
ELSE
IF (.EQ.2) THEN
NHRN1(I,K) = 0.0
NHRB1(I,K) = 0.0
ELSE
NHRN1(I,K) = OPA(I,K+1) * TEMP13
NHRNO(I,K) = 1.0 - (2.0 * NHRN1(I,K))
IF ((I.EQ.3).AND.(K.EQ.2)) GO TO 65
TEMP14 = (OPAN(I,K+1) * 0.5*ORHO(I-1,K+1)) + (OPAB(I,K+1) *
1 (1.0 - 0.5*ORHO(I-1,K+1)))
TEMP15 = 0.375 * NHPB(I,K) * OPAN(I,K+1) * 0.5*ORHO(I-1,K+1)
NHRB1(I,K) = (TEMP13 * TEMP14) - TEMP15
NHRBO(I,K) = 1.0 - (2.0 * NHRB1(I,K))
CONTINUE
65 ENDIF
ENDIF
ENDIF

C
TEMP16 = 2.0 * NHPM(I,K) * NHRN1(I,K)
TEMP17 = 2.0 * NHPB(I,K) * NHRB1(I,K)
NHPOT(I,K) = NHPOT(I,K) + TEMP16 + TEMP17
NHPBT(I,K) = (NHPM(I,K) * NHRNO(I,K)) + (NHPB(I,K) * NHRBO(I,K))
NHPA(I,K) = NHPOT(I,K)
NHPAN(I,K) = 2.0 * NHRN1(I,K)
NHPAB(I,K) = 2.0 * NHRB1(I,K)
IF (K.NE.I) THEN
ORHO(I,K+1) = TEMP16 + TEMP17
ENDIF
251     OQ(I,K+1) = ORHO(I,K+1) / OPA(I,K+1)
252     ENDIF
253     RETURN
254     END
255     ************************************************************
256     SUBROUTINE OTHER(OPO,OPM,OPB,ORNI,ORB1,ORNO,ORBO,OPAN,
257       1   OPAB,OPA,ORQ,ORHO,OPOT,OPBT,R,I,K,N)
258     REAL OPO(15,5),OPM(15,5),OPB(15,5)
259     REAL ORNI(15,5),ORB1(15,5),ORNO(15,5),ORBO(15,5),OPAN(15,5)
260     REAL OPAB(15,5),OPA(15,5),OQ(I:15,5),ORHO(0:15,5),OPOT(0:15,5)
261     REAL OPBT(0:15,5),R
262     REAL TEMP18,TEMP19,TEMP20
263     INTEGER I,K
264     TEMP18 = 0.0
265     TEMP19 = 0.0
266     TEMP20 = 0.0
267     OPO(I,K) = OPOT(I-1,K) * (1.0 - OQ(I-1,K))
268     OPM(I,K) = OPOT(I-1,K) * OQ(I-1,K)
269     OPB(I,K) = OPBT(I-1,K)
270     TEMP18 = OPM(I,K) + OPB(I,K)
271     IF (I.NE.1) THEN
272     IF (I.GT.3) THEN
273       ORNI(I,K) = 0.5 - (0.125 * TEMP18)
274       ORNO(I,K) = 1.0 - (2.0 * ORNI(I,K))
275     IF (I.GT.4) THEN
276       ORB1(I,K) = ORNI(I,K)
277       ORB1(I,K) = ORNO(I,K)
278     ENDIF
279     ENDIF
280     TEMP19 = 2.0 * OPB(I,K) * ORB1(I,K)
281     TEMP20 = 2.0 * OPM(I,K) * ORN1(I,K)
282     OPOT(I,K) = OPO(I,K) + TEMP19 + TEMP20
283     OPBT(I,K) = (OPM(I,K) * ORNO(I,K)) + (OPB(I,K) * ORBO(I,K))
284     OPB(I,K) = OPOT(I,K)
285     OPAB(I,K) = 2.0 * ORN1(I,K)
286     OPAN(I,K) = 2.0 * ORB1(I,K)
287     RETURN
288     END

Compilation time (seconds): 0.50 Execution time (seconds): 0.79
Size of object code: 17350 Number of extensions: 0
Size of local data area(s): 6129 Number of warnings: 5
Size of global data area: 14680 Number of errors: 0
Object/Dynamic bytes free: 799168 Statements Executed: 63155