Universidad Politécnica de Valencia.
Departamento de Informática de Sistemas y Computadores.

EFFICIENT MECHANISMS TO PROVIDE FAULT TOLERANCE IN INTERCONNECTION NETWORKS FOR PC CLUSTERS

PhD thesis by:
José Miguel Montañana Aliaga.

Advisors:
Dr. José Flich Cardo,
Dr. Antonio Robles Martínez.

Valencia, Spain, 2008

This thesis was written using LaTeX. Figures were plotted using XFIG and GNUPLOT.

Prólogo

El presente documento se ha elaborado bajo la dirección de los profesores D. José Flich Cardo y D. Antonio Robles Martínez, con la finalidad de obtener el título de Doctor.

En la primera parte de la presente memoria se presenta el estado del arte en las redes de interconexión, haciendo énfasis en las técnicas y métodos de tolerancia a fallos y su análisis.

En la segunda parte se presentan y evalúan las diferentes técnicas que se proponen para proveer a las redes de interconexión para clusters de aceptables márgenes de tolerancia a fallos. Dicha actividad investigadora se ha desarrollado en el Grupo de Arquitecturas Paralelas (GAP) del Departamento de Informática de Sistemas y Computadoras (DISCA) de la Universidad Politécnica de Valencia (UPV).

En la última parte, se concluye con un resumen de las contribuciones y de las conclusiones. También se presentan las publicaciones relacionadas con las contribuciones de esta tesis, junto con las direcciones futuras de investigación.

Preface

The current document has been elaborated with the aim to obtain the PhD degree. This work has been done under the advising and guidance of professors José Flich and Antonio Robles.

The first part of the document introduces the state-of-art in the interconnection networks field, specially focusing on fault tolerance issues and their analysis.

In the second part all the research work is presented and evaluated. This research activity has been developed within the Parallel Architectures Group (GAP) in the department of computer engineering (DISCA) at the Technical University of Valencia (UPV).

In the last part, the contributions and conclusions are summarized. Related publications of the contributions are also presented together with future directions in the research.

This page intentionally left blank

Acknowledgements

I would like to thank for the support and guidance of my advisors, professors José Flich and Antonio Robles, without their knowledge and research experience this work would not be possible.

Thanks to professors Timothy Mark Pinkston and Olav Lysne for sharing their knowledge on reconfiguration protocol.

I want to thank for the support of the people in the research group. Also to my family and friends for their support.

Additionally to the powerful clusters that had run so many simulations.

This thesis is dedicated to my family and friends, in gratitude for moral support and help with everyday problems.

This page intentionally left blank

Abstract

Currently, clusters of PCs are considered a cost-effective alternative to large parallel computers. In these systems thousands of components (processors and/or hard disks) are connected through high-performance interconnection networks. Among the high-performance network technologies currently available to build clusters, InfiniBand (IBA) has emerged as a new standard interconnection technology suitable for clusters. Indeed, has been adopted by many of the most powerful systems currently built (top500 list).

As the number of nodes increases in these systems, the interconnection network grows accordingly. Along with the increase in components the probability of faults increases dramatically, and thus, fault tolerance in the system, in general, and in the interconnection network, in particular, becomes a necessity.

Unfortunately, most of the fault-tolerant routing strategies proposed for massively parallel computers cannot be applied because routing and virtual channel transitions are deterministic in IBA, which prevent packets from avoiding the faults. Therefore, a new and effective fault-tolerant strategy is needed.

Thus, this thesis focuses on providing mechanisms for provinding adequate levels of fault tolerance to the routing in PC clusters, specially tailored to IBA networks. We propose and evaluate in this thesis several mechanisms suitable for interconnection networks for clusters.

The first mechanism to provide fault tolerance in IBA (referred to as Transition Fault Tolerant Routing; TFTR) consists of using several disjoint paths between every source-destination pair of nodes and selecting the appropriate path at the source end node by using the APM mechanism provided by IBA. It consists of migrating on the fly the paths affected by the failure to alternative fault-free paths. However, to this end, an efficient routing algorithm able to provide enough disjoint paths, while still guaranteeing deadlock freedom, is required. We refer to an efficient routing algorithm as the one that minimizes the required set of resources and is computed in a time-efficient manner. We address this issue/approach, in a second effort, by proposing an scalable fault-tolerance methodology (referred to as SPFTR) for tori in IBA networks.

As a second contribution of this thesis, we propose a simple and effective fault-tolerant routing methodology (referred to as Reachability Based Fault Tolerant Rout-

ing; RFTR), which can be applied to any topology. RFTR builds new alternative paths by joining subpaths extracted from the set of already computed paths, thus being time-efficient.

In the last contribution, we focus on providing fault tolerance based on dynamic reconfiguration. We propose a simple and fast method for dynamic network reconfiguration, referred to as Epoch Based Reconfiguration (EBR). EBR guarantees a fast and deadlock-free reconfiguration, but instead of avoiding deadlocks our mechanism is based on regressive deadlock recoveries. EBR works in an asynchronous manner, does not require additional resources and can be applied on any topology.

Most of the proposals made in this thesis are suitable (with no hardware modification) to be implemented on commercial network technologies (mainly IBA networks) currently used in clusters of PCs, and are able to tolerate dynamically a reasonable number of faults as long as the network remains physically connected.

Resumen

Actualmente, los clusters de PC son un alternativa rentable a los computadores paralelos. En estos sistemas, miles de componentes (procesadores y/o discos duros) se conectan a través de redes de interconexión de altas prestaciones. Entre las tecnologías de red actualmente disponibles para construir clusters, InfiniBand (IBA) ha emergido como un nuevo estándar de interconexión para clusters. De hecho, ha sido adoptado por muchos de los sistemas más potentes construidos actualmente (lista top500).

A medida que el número de nodos aumenta en estos sistemas, la red de interconexión también crece. Junto con el aumento del número de componentes la probabilidad de averías aumenta dramáticamente y, así, la tolerancia a fallos en el sistema en general, y de la red de interconexión en particular, se convierte en una necesidad.

Desafortunadamente, la mayor parte de las estrategias de encaminamiento tolerantes a fallos propuestas para los computadores masivamente paralelos no pueden ser aplicadas porque el encaminamiento y las transiciones de canal virtual son deterministas en IBA, lo que impide que los paquetes eviten los fallos. Por lo tanto, son necesarias nuevas estrategias para tolerar fallos.

Por ello, esta tesis se centra en proporcionar mecanismos para proporcionar los niveles adecuados de tolerancia a fallos al encaminamiento en los clusters de PC, y en particular a las redes IBA. En esta tesis proponemos y evaluamos varios mecanismos adecuados para las redes de interconexión para clusters.

El primer mecanismo para proporcionar tolerancia a fallos en IBA (al que nos referimos como encaminamiento tolerante a fallos basado en transiciones; TFTR) consiste en usar varias rutas disjuntas entre cada par de nodos origen-destino y seleccionar la ruta apropiada en el nodo fuente usando el mecanismo APM proporcionado por IBA. Consiste en migrar las rutas afectadas por el fallo a las rutas alternativas sin fallos. Sin embargo, con este fin, es necesario un algoritmo eficiente de encaminamiento capaz de proporcionar suficientes rutas disjuntas, que además garantice la ausencia de situaciones de bloqueo. Definimos un algoritmo de encaminamiento como eficiente, aquel que reduce al mínimo las necesidades de recursos requeridos del sistema y se obtiene de una manera eficiente en tiempo de cálculo. Para ello, proponemos una metodología escalable tolerante a fallos (denominada SPFTR) para toros en IBA.

Como segunda contribución de esta tesis, proponemos una metodología tolerante a fallos simple y eficaz (denominada como encaminamiento tolerante a fallos basado en alcanzabilidad; RFTR), que se puede aplicar a cualquier topología. RFTR compone las rutas alternativas uniendo fragmentos de rutas ya existentes, de un modo rápido y eficiente.

Como última contribución, nos centramos en proporcionar tolerancia a fallos mediante reconfiguración dinámica. Proponemos un método simple y rápido de reconfiguración dinámica de red, denominado reconfiguración basada épocas (EBR). EBR garantiza una reconfiguración rápida y la ausencia de bloqueos, pero en vez de evitar las situaciones de bloqueo nuestro mecanismo se basa en técnicas de recuperación para evitar los bloqueos. EBR funciona de una manera asincrónica, no requiere recursos adicionales y se puede aplicar en cualquier topología.

La mayor parte de las propuestas hechas en esta tesis son válidas (sin ninguna modificación del hardware) para ser utilizadas en tecnologías de red comerciales (principalmente en redes IBA) usadas actualmente en clusters de PC, y pueden tolerar dinámicamente un número razonable de fallos mientras la red se mantenga conectada físicamente.

Resum

Actualment, els clústers de PC es consideren un alternativa rendible als computadors massivament paral·lels. En aquests sistemes, milers de components (processadors i/o discs durs) estan connectats mitjançant xarxes d'interconnexió d'altes prestacions. Entre les tecnologies de xarxa d'altes prestacions actualment disponibles per a construir clústers, InfiniBand (IBA) ha emergit com un nou estàndard d'interconnexió adient per als clústers. De fet, ha sigut adoptat per molts dels sistemes més potents construïts actualment (llista top500).

A mesura que el nombre de nodes augmenta en estos sistemes, la xarxa d'interconnexió també creix. Junt amb l'augment del nombre de components la probabilitat d'avaries augmenta dramàticament i, així, la tolerància a fallades en el sistema en general, i de la xarxa d'interconnexió en particular, esdevé una necessitat.

Dissortadament, la major part de les estratègies d'encaminament tolerants a fallades proposades per a les computadores massivament paral·leles no poden ser aplicades perquè l'encaminament i les transicions de canal virtual en IBA són deterministes, cosa que impedix que els paquets defugen de les fallades. Per tant, és necessària una estratègia nova i eficaç per a tolerar fallades.

Per això, aquesta tesi se centra a proporcionar mecanismes per proporcionar els nivells adequats de tolerància a fallades al encaminament en clústers de PC, especialment confeccionats per a les xarxes IBA. En aquesta tesi proposem i avaluem diversos mecanismes adients a les xarxes d'interconnexió per a clústers.

El primer mecanisme per a proporcionar tolerància a fallades en IBA (al qual ens referim com encaminament tolerant a fallades basat en transicions, TFTR) consistix a usar de diverses rutes disjuntes entre cada parell de nodes origen-destí, i seleccionarne l'apropiada en el node font fent servir el mecanisme APM proporcionat per IBA. Consistix a migrar dinàmicament de les rutes afectades per la fallada a les rutes alternatives lliures de fallades. No obstant això, amb aquest fi, cal un algoritme eficient de encaminament capaç de proporcionar suficients rutes disjuntes, que a més garantisca l'absència de situacions de bloqueig. Definim un algoritme d'encaminament com eficient el que reduïx al mínim els recursos del sistema requerits i que es calcula d'una manera eficient en temps. Nosaltres tractem esta qüestió, en un segon esforç, proposant una metodologia d'encaminament tolerant a fallades i escalable (denominada SPFTR) per a xarxes IBA amb topologia de torus.

Com a segona contribució d'esta tesi, proposem una metodologia tolerant a fallades simple i eficaç (denominada encaminament tolerant a fallades basat en alcanzabilitat, RFTR), que es pot aplicar a qualsevol topologia. RFTR compon les rutes alternatives unint fragments de rutes ja existents, d'una manera eficient en temps.

Com ultima contribució, ens centrem en proporcionar tolerància a fallades basada en reconfiguració dinàmica. Proposem un mètode simple i ràpid de reconfiguració dinàmica de xarxa, denominat reconfiguració basada en èpoques (EBR). EBR garanteix una reconfiguració ràpida i l'absència de bloquejos però, en comptes d'evitar les situacions de bloqueig, el nostre mecanisme es basa en recuperacions regressives d'aquestes situacions de bloqueig. EBR funciona d'una manera asíncrona, no requerix recursos addicionals i es pot aplicar en qualsevol topologia.

La major part de les propostes fetes en esta tesi són vàlides (sense cap modificació del maquinari) per a ser utilitzades en les tecnologies de xarxa comercials (principalment en xarxes IBA) usades actualment en clústers de PC, i poden tolerar dinàmicament un nombre raonable de fallades mentres la xarxa es mantinga connectada físicament.

Contents

1 Introduction **1**

1.1 Motivation and Objectives 6
1.2 Contributions 7
1.3 Organization 8

2 Interconnection Networks for HPC Clusters **9**

2.1 Network Design for Clusters 10
2.1.1 Data Units 11
2.1.2 Switching 12
2.1.3 Virtual Channels 13
2.1.4 Flow Control 14
2.1.5 Arbitration 16
2.1.6 Topology 17
2.1.7 Routing Algorithm 20
2.1.7.1 Taxonomy 20
2.1.7.2 The Deadlock Problem 22
2.1.7.3 The Livelock Problem 24
2.1.7.4 Routing Algorithms for Clusters 25
2.1.8 Network Reconfiguration 27
2.1.8.1 Static and Dynamic Reconfiguration 28
2.1.8.2 Existing Proposals on Dynamic Reconfiguration . 28
2.2 InfiniBand Networks 29
2.2.1 InfiniBand Architecture 31
2.2.2 Virtual Channels and Flow Control 33
2.2.3 Routing in IBA 33
2.2.4 Subnet Manager 36

3 Fault Tolerance for Interconnection Networks **37**
3.1 Basic Concepts . . . 37
3.2 Fault Models . . . 40
3.3 Fault-Tolerance Mechanisms . . . 41
3.3.1 Fault-Tolerant Routing Algorithms . . . 42
3.3.1.1 Hardware-Based Fault Tolerant Routing Algorithms 42
3.3.1.2 Software-Based Fault Tolerant Routing Algorithms 46
3.4 Applying Fault Tolerant Mechanisms to PC Clusters . . . 47
3.4.1 Fault Tolerance in IBA . . . 47
3.4.2 Contributions of the Thesis . . . 48

4 Fault-Tolerant Routing Methodologies Based on Disjoint Paths **51**
4.1 Preliminaries . . . 51
4.2 TFTR Methodology . . . 55
4.2.1 TFTR Stages . . . 56
4.2.2 Enhancements to the TFTR Methodology . . . 63
4.2.3 Computational Cost . . . 66
4.3 SPFTR Methodology . . . 67
4.3.1 Motivation . . . 67
4.3.2 SPFTR stages . . . 69
4.3.3 Routing Algorithm . . . 70
4.3.4 Route Patterns and Network Regions . . . 71
4.3.5 SLtoVL Table Initialization . . . 73
4.3.6 Extending the Methodology . . . 78
4.3.7 Optimizing the methodology: A-SPFTR . . . 81

5 Fault-Tolerant Routing Methodology Based on Reachability **83**
5.1 Motivation . . . 83
5.1.1 RFTR Preliminaries . . . 84
5.2 RFTR Description . . . 86
5.2.1 Direct and Indirect Reachability . . . 86
5.2.2 RFTR Methodology . . . 87
5.3 RFTR Complexity . . . 91
5.4 RFTR on InfiniBand . . . 92

6 Fault-Tolerance Methodology Based on Reconfiguration **95**
6.1 Motivation . . . 95
6.2 Basic EBR Description . . . 97

6.3 Triggering the EBR Mechanism . . . 98
6.4 Routing Path Computation . . . 99
6.5 Path Distribution . . . 100
6.6 Deadlock Recovery Process . . . 101
6.7 Overlapping Multiple Reconfiguration Processes . . . 104

7 Evaluation Model **107**
7.1 Introduction to Simulation Modeling . . . 107
7.1.1 Simulation Tool . . . 108
7.1.2 Advantages and Disadvantages . . . 109
7.2 Network Model . . . 110
7.2.1 Topologies . . . 110
7.2.2 End to End Flow Control . . . 111
7.2.3 Traffic Patterns . . . 111
7.2.4 Simulation Parameters . . . 112
7.3 Modelling and Analyzing Faults . . . 113
7.4 Compared Reconfiguration Mechanisms . . . 114
7.5 Collecting Results . . . 116
7.5.1 Performance Metrics . . . 118

8 Evaluation **121**
8.1 TFTR Methodology . . . 121
8.1.1 Number of Paths . . . 122
8.1.2 Singular Cases . . . 122
8.1.3 Resources Required . . . 124
8.1.4 Quality of Routing Paths . . . 125
8.1.5 TFTR with Additional Virtual Channels . . . 126
8.1.6 Network Throughput . . . 126
8.1.7 Computational Cost . . . 128
8.1.8 Summary . . . 128
8.2 SPFTR and A-SPFTR Methodologies . . . 129
8.2.1 Singular Cases . . . 129
8.2.2 Quality of Routing Paths . . . 130
8.2.3 Network Throughput . . . 131
8.2.4 Computational Cost and Resources Needed . . . 132
8.2.5 Summary . . . 132
8.3 RFTR Methodology . . . 133
8.3.1 Singular Cases . . . 134

8.3.2 Quality of Routing Paths . . . 134
8.3.3 Resources Required . . . 136
8.3.4 Computational Cost . . . 136
8.3.5 Network Throughput . . . 137
8.3.6 Summary . . . 142
8.4 EBR Methodology . . . 143
8.4.1 Traffic Patterns and Topologies . . . 143
8.4.2 Dropped Packets . . . 146
8.4.3 Average Packet Latency . . . 146
8.4.4 Network Throughput . . . 147
8.4.5 Scalability . . . 148
8.4.6 Summary . . . 150

9 Conclusions **151**
9.1 Conclusions . . . 151
9.2 Contributions . . . 152
9.3 Publications . . . 156
9.4 Future Work . . . 157

A Route Patterns for SPFTR **159**

B Acronyms **167**

Bibliography **169**

Index **183**

List of Figures

1.1 Examples of parallel architectures: (a) Multiprocessor system with Uniform Memory Access (UMA), (b) Multiprocessor system with Non-Uniform Memory Access (NUMA), and (c) Multicomputer system. . . . 3
1.2 Chip and performance evolution: (a) Growth of transistor counts for Intel processors (dots) and Moore's Law, (b) performance measured in MIPS for Intel processors. . . . 5
1.3 Architectures used by supercomputers on the *top500* list. . . . 5

2.1 Different data units in an interconnection network. . . . 11
2.2 Switching mechanisms: (a) *store & forward* and (b) *virtual cut-through*. 12
2.3 (a) Example of blocked messages in *wormhole*. (b) Virtual channels. 13
2.4 Examples of flow control mechanisms: (a) *credit-based*, (b) *on/off*, and (c) *ack/nack*. . . . 15
2.5 Internal model of a switch. . . . 17
2.6 Examples of direct topologies: (a) 2D mesh, (b) 2D torus, and (c) hypercube (n–cube). . . . 18
2.7 Examples of indirect topologies: (a) 4×4 crossbar, (b) irregular network, and (c) bidirectional multistage network. . . . 19
2.8 Taxonomy of the routing algorithms. . . . 21
2.9 Different routing mechanisms: (a) source routing and (b) distributed routing based on tables. . . . 22
2.10 Example of a deadlock situation: (a) a deadlock situation, and (b) example of a cycle in the CDG. . . . 23
2.11 Allowed turns in XY routing algorithm. . . . 25
2.12 Link labels for the *up*/down** (BFS) routing algorithm. . . . 26
2.13 Global reconfiguration stages. . . . 28
2.14 Interconnection technologies used by supercomputers on the *top500* list. . . . 30
2.15 Example of an IBA subnet. . . . 32

2.16 Examples of IBA tables (a) forwarding table and (b) SLtoVL table. . 34
2.17 Example of routing tables with different LMC values. 35
2.18 Mapping conflict example. 35

3.1 MTBF values of some supercomputers and MTBF curves obtained for different values of reliability. 38
3.2 Taxonomy of fault tolerant routing algorithms. 42
3.3 Example of block faults: (a) rectangular block faults, and (b) a solid block fault (nonconvex). 43
3.4 Example of disjoint paths. 49
3.5 Example of composing an alternative path from two already existing routing paths in the presence of faults. 49

4.1 Examples of disjoint paths: (a) Sharing one link, (b) Sharing one intermediate switch, (c) Disjoint paths, (d) Multiple NICs at end nodes. 53
4.2 Examples of the degree of disjoint paths: (a) sharing one link and one switch, and (b) sharing two links and two switches. 54
4.3 Example of disjoint paths. (a) Four disjoint paths in a 5×5 torus and (b) the set of routing restrictions enforced by *up*/down** on the same 5×5 torus (R means root switch). 55
4.4 TFTR Stages. 56
4.5 Example of up*/down* routing restrictions in a 4×4 torus. 57
4.6 Example of a cycle and how it can be avoided when using virtual networks: (a,c) example of a cycle, (b,d) example of virtual network transition. 58
4.7 Example of a set of four disjoint paths in a 2D torus. 59
4.8 Examples of different ways to solve a mapping conflict. (a) Example of a mapping conflict, (b) mapping conflict solved by changing the VL, (c) by using an alternative paths, (d) by using a new SL, and (e) by discarding one path. 61
4.9 Example of multiple transitions between virtual networks. 62
4.10 Detailed example for a given source destination pair when applying the TFTR methodology (in a 2D mesh). 64
4.11 Example of disjoint paths. 65
4.12 Link ordering in order to speed up the computing process of the disjoint paths: (a) for 2D tori/meshes and (b) for 3D tori/meshes. . . . 66
4.13 (a,b) Examples of two sets of the minimal disjoint paths in a 3D torus. 66
4.14 Stages of the enhanced TFTR version. 67
4.15 Steps followed by SPFTR. 69

4.16 Routing restrictions of (a) *up*/down** routing algorithm and (b) the ad-hoc underlying routing algorithm used for SPFTR. 71
4.17 Example of a path using SL=0. 72
4.18 Regions in a 6×6 torus network when using SPFTR. 73
4.19 An example of route patterns for a source-destination pair of nodes placed at the same region #0. 74
4.20 Underlying routing algorithm applied: (a) routing restrictions, (b) SLtoVL table initialization for switch 14. N/D stands for Not Defined. X stands for any value. 75
4.21 Initial route patterns used for initial definition of SLtoVL tables. . . 76
4.22 Example of a route pattern. 77
4.23 Example of pattern applied in a (a) 6×6 and (b) 7×7 torus network. 77
4.24 Regions in 2D torus when using SPFTR. 78
4.25 2D planes in a 3D torus. 79
4.26 Switches in the same position in the *XZ* planes in a $6 \times 6 \times 6$ torus are included in the same position in a *XY* plane. 80
4.27 Pair of links shared between every pair of planes in a 3D torus. . . . 80
4.28 (a) Frontal view of routing restrictions in the XY plane of a 3D torus. (b) SPFTR regions in a 3D torus. 81
4.29 Regions in 2D torus when using A-SPFTR in (a) 7×7 torus network, and (b) in a 8×8 torus network. 82

5.1 Paths in a 3×3 mesh applying DOR. 85
5.2 Paths in a 3×3 mesh. 87
5.3 Applying RFTR for one link failure in a 3×3 mesh (part 1). 89
5.4 Applying RFTR for one link failure in a 3×3 mesh (part 2). 90
5.5 Stages followed by RFTR on IBA. 92

6.1 Cycle in a dynamic reconfiguration not introducing a deadlock. . . . 98
6.2 Stages of the EBR mechanism. 99
6.3 Sequence of routing table updates. The numbers indicate the sequence and the distance to the failed component. 100
6.4 Example of a switch status: (a) before and (b) after updating the routing info. 101
6.5 Example of routing and scheduling during the reconfiguration process at switch C (a) before being updated (*old* and (b) after being updated (*new*). 103
6.6 Logic implementing the EBR deadlock recovery mechanism, (a) simple circuit, (b) extended version. 104

7.1 Structure of the Discrete-Event Simulation Model. 110
7.2 Reconfiguration schemes evaluated and events that distinguish them. 116
7.3 Determining transient state duration in the simulations. 117
7.4 Plotting latency at (a) reception time or at (b) injection time. 118

8.1 Percentage of singular cases in 2D tori that could not be handled by the methods. 123
8.2 Network throughput degradation when using TFTR for different torus networks. 127
8.3 Computational cost of TFTR for computing the set of disjoint paths on different 2D torus networks. 128
8.4 Singular cases for link failures for (a) *up*/down** and (b) SPFTR in 2D and in 3D tori (100,000 combinations evaluated at each point when the number of fault combinations is higher than 100,000). . . 130
8.5 Singular cases for switch failures for SPFTR in 2D tori (100,000 combinations evaluated at each point when the number of fault combinations is higher than 100,000). 130
8.6 Average path length in (a) 2D torus and (b) in 3D torus. 131
8.7 Degradation of performance with faults for SPFTR and A-SPFTR. . 131
8.8 Comparison between TFTR and SPFTR of the required number of SLs in 2D tori. 132
8.9 Comparison of computational cost of TFTR and SPFTR in a Xeon 3.06 GHz (a) linear scale and (b) logarithmic scale. 133
8.10 Average path length when using RFTR and SR. 135
8.11 Average computational time (a,b) for RFTR and for different number of failures in 2D and 3D tori, (c,d) for *up*/down** and RFTR in 2D and 3D tori with one failed link. 137
8.12 Events of fault detection, computation and distribution of the new routing tables evaluated. 138
8.13 Network throughput degradation. 139
8.14 (a,b) Accepted traffic when using RFTR and Simple Reconfiguration algorithms, respectively. (c) Accumulated number of lost bytes using RFTR and Simple Reconfiguration algorithms. 140
8.15 Average routing information sent after the first failure. 141
8.16 Average computational time for TFTR,SPFTR, *up*/down** and RFTR on 2D tori. Figure also shows the correlated exponential function for the computational cost of each mechanism. 142
8.17 Performance and reference traffic injection rates for the 6×6 torus network. 144

8.18 Reconfiguration time for 6×6 torus network and uniform traffic pattern. 145
8.19 Sequence in the distribution of routing tables (may affect the reconfiguration time). 145
8.20 Dropped and discarded packets for 6×6 torus network and uniform traffic pattern. 146
8.21 Traffic lost when using (a) Synthetic traffic patterns and (b) and traces. 147
8.22 Average latency on 10x10 torus network during the reconfiguration and the transient time after it, each plot has a close up which shows the latency during the reconfiguration. 148
8.23 Delivered traffic for 10×10 torus network and uniform traffic pattern. 149
8.24 Performance results for different tori and uniform traffic pattern, (a) Reconfiguration time and (b) Dropped and discarded packets. 149

9.1 Schema for selecting the most suitable fault tolerance mechanism depending on the number of faults and the time required for tolerating them. 155
9.2 Schema for selecting fault tolerance methodology, depending on MTBF and MTTR. 155

A.1 Example of route pattern applied. 160
A.2 Complete set of sequences of movements for obtaining all the patterns for 2D direct networks. 161
A.3 Complete set of route patterns for 2D direct networks. Each patter contains 4 sequences of movement for composing 4 disjoint paths. . 162

This page intentionally left blank

List of Tables

1.1 Comparison of Shared and Distributed Memory Architectures. . . . 4

2.1 Lowest measured interconnect latency (smaller number is better), and bandwidth (larger number is better) in interconnection networks. . . 10

3.1 MTBF values on Large-Scale HPC Systems. 39

4.1 SLtoVL table for switch Y and SL=0. X stands for any port. 72
4.2 SLtoVL tables for every region in 2D torus network for SPFTR method. Values imposed by routing restrictions in each region are written in bold face. X stands for any value. 77
4.3 SLtoVL tables for every region in 2D torus network for A-SPFTR method. Values imposed by routing restrictions in each region are written in bold face. X stands for any value. 82

5.1 Example of DRT and IRT tables. 87

6.1 Actions performed by EBR (when no credits are available) based on the packet (E_{PCK}) and the switch (E_{RTD} and E_{SCHD}) epoch bits. (No overlapping is supported). 102
6.2 Actions performed by EBR (when no credits are available) based on the packet (E_{PCK}) and the switch (E_{RTD} and E_{SCHD}) epoch bits. Overlapping is supported. 105

8.1 Evaluated scenarios for TFTR. 122
8.2 Percentage of end node pairs with n disjoint paths, when using an unbounded number of SLs. 123
8.3 Percentage of pairs of end nodes with n disjoint paths when using 2 VLs and a bounded number of SLs. 125
8.4 Percentage of paths using a particular SL with a bounded number of SLs and 2 VLs. 125

8.5 Routing path metrics for TFTR and *up*/down**. 126
8.6 Routing path metrics for TFTR when using 4VLs and an unbounded number of SLs. 127
8.7 Evaluated scenarios for SPFTR and A-SPFTR. 129
8.8 Evaluated scenarios for RFTR. 134
8.9 Fault tolerance achieved and resources required by RFTR, TFTR, and SPFTR. For SPFTR the minimum network size for 3D torus topology is $4 \times 4 \times 4$. 135
8.10 Memory requirements. 136
8.11 Evaluated scenarios for EBR. 143
8.12 Summary of the characteristics of the different dynamic reconfiguration mechanisms. 150

9.1 Summary of the advantages and drawbacks of the proposals. 154

A.1 Nomenclature used to define the sequences of movements. 159

Chapter 1

Introduction

Nowadays, the remarkable cost/performance ratio of personal computers[1] (PCs) has allowed them to become a common element in many environments where intensive computation is required. Many of the most powerful machines nowadays are built from commercial-of-the-shelf (COTS) systems as can be observed when examining the *top500* supercomputer list [39].

At the same time, new services and applications require higher computational capacities, dealing also with higher amounts of information and hosting an increasing number of users connected to Internet. Indeed, the increasing number of users (through Internet), as well as the use of sophisticated applications is causing a high computational load on servers where the response time becomes a critical factor.

As a side effect, the limitation of available bandwidth and the increasing necessities in the transmission of information is leading to the use of compression and/or encoding techniques, in many cases, in real time, thus increasing the requirements of higher computational power in such systems. Moreover the data centers of large companies and institutions manage large volumes of information. These companies use computation intensive techniques such as *data warehousing* [90] and *data mining* [80]. These applications often require exchanging information in the order of terabytes.

The necessity to attend such an increase of users, as well as computational power and storage capacity, leads to the need of building scalable and flexible systems that facilitate their expansion in an incremental way and guarantee their long term growth without degrading performance.

Although the performance of processors has increased steadily in the last years, there is a physical limit. To break this limit, the logical solution for providing higher computational power is to coordinate multiple processors in a system to concurrently perform the computation tasks, dividing the tasks into subtasks, each one being

[1] With the term personal computers we also include workstations.

solved in a different processor. These systems are widely known as parallel computers.

During the last decades different parallel computer architectures have emerged. Basically, there are two fundamental types of parallel computers, either a single computer with multiple internal processors sharing all the memory (single memory address space), known as a Shared Memory Multiprocessor, or a set of independent computers interconnected through a network. the second case the memory is distributed. These systems are known as multicomputers. Next, we briefly describe each system type:

- Shared memory multiprocessor

  A conventional computer consists of a processor and a memory readily accessible by the processor. The shared memory multiprocessor is a natural extension where multiple processors are connected to multiple memory modules through a certain interconnection network and support a single address space throughout the system. This means that any processor can access any memory location without the need of copying data from one memory to another. We can classify multiprocessors in two classes depending on how the memory is shared:

  - Multiprocessors with a centralized memory or multiprocessor with uniform memory access (UMA), also known as Symmetric multiprocessor (SMP). In this type of systems the access time is uniform for every memory module from any processor. Figure 1.1.a shows a multiprocessor system with a centralized memory with Uniform Memory Access. Examples of commercial UMA machines are Sun Fire 15000 system with 106 UltraSparc III processors [36], SGI Challenge [33], and IBM POWER3 [31].
  - Multiprocessors with Distributed Shared Memory (DSM), also known as multiprocessors with Non-Uniform Memory Access (NUMA). The memory is shared but distributed among the processors, therefore the access time to memory is non-uniform (smaller to the local memory, and larger to non-local memories). To reduce the effects of non-uniform memory access, caches can be used. These systems with cache are referred to as Cache Coherent Non-Uniform Memory Access (CC-NUMA). Figure 1.1.b shows a NUMA multiprocessor system. Examples are BBN Butterfly [60], Cray T3D [38], and SGI Origin 2000 [100], which use NUMA, noncoherent cache NUMA, and CC-NUMA, respectively.

- Non-shared memory multiprocessors or multicomputers

  A multicomputer consists of independent computers (processor and local memory) connected via an interconnection network as shown in Figure 1.1.c. Each processor is able to address only its local memory space. Inter-processor communication is achieved by sending explicit messages from each computer to another using a message-passing library such as MPI (Message-Passing Interface) [13]. In such a setup, each computer has its own memory address space. In order to access a certain value from a different computer, it has to be copied by sending a message to the desired computer. Message-passing multicomputers physically scale easier than shared memory multiprocessors because it is easier to extend the system by adding more computers. Moreover, these systems can support shared-memory applications by providing a certain software layer, such as *TreadMarks* [92].

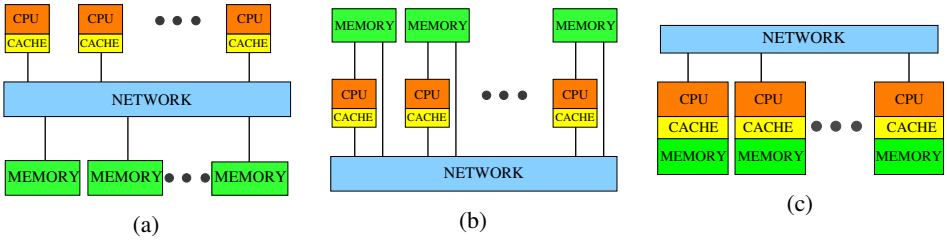


Figure 1.1: Examples of parallel architectures: (a) Multiprocessor system with Uniform Memory Access (UMA), (b) Multiprocessor system with Non-Uniform Memory Access (NUMA), and (c) Multicomputer system.

These systems (multicomputers and multiprocessors) solve challenging problems of high computational cost like the modeling of protein folders, modeling of the global climatic change during long periods of time, the evolution of galaxies, the atomic structure of materials, and so on.

Although these systems are very powerful, they present a series of disadvantages, being the most important their high costs (due to their reduced volume of sales). Additionally, although they are helpful when running highly parallel applications, a high number of applications exist that are not easily parallelizable or even do not require their parallelization. Thus, only a reduced set of applications take advantage of these systems. In addition to this, they present a high cost per node, as they are made up of expensive components. Another drawback is the need for highly qualified personnel for their maintenance. Table 1.1 shows a comparison between the different architectures in terms of scalability, software availability, communication, and drawbacks. Also, some examples are shown.

	Architecture			
	CC-UMA	CC-NUMA	NUMA	PC Clusters
Examples	SMPs Sun Vexx DEC/Compaq SGI Challenge IBM POWER3	SGI Origin Sequent HP Exemplar DEC/Compaq IBM POWER4 (MCM)	Cray T3E	Mare Nostrum
Communications	MPI Threads OpenMP shmem	MPI Threads OpenMP shmem	MPI	MPI
Scalability	to 10s of processors	to 100s of processors	to 1,000s of processors	to 10,000s of processors
Drawbacks	Memory-CPU bandwidth	Memory-CPU bandwidth Non-uniform access times	System administration Programming is hard to develop and maintain	Memory is distributed
Software availability	many 1,000s ISVs	many 1,000s ISVs	100s ISVs	many 1,000s ISVs

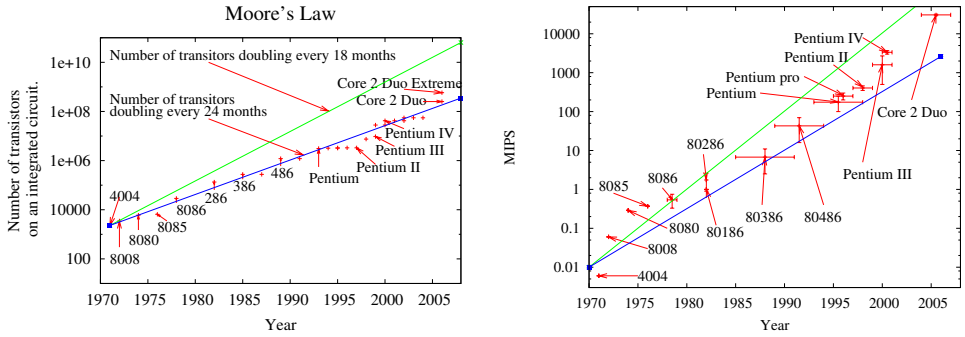
ISV: independent software vendor.

Table 1.1: Comparison of Shared and Distributed Memory Architectures.

As previously stated, an attractive solution to build high performance systems (like supercomputers or web servers) is the use of clusters of PCs. We refer to a cluster as a group of PCs (or workstations) interconnected among them. The cost/performance ratio of the PC has improved dramatically in the last years and nowadays hundreds or even thousands of commercial-off-the-shelf (COTS) processors can be glued to challenge Massive Parallel Systems (MPPs).

To support this trend it is well known that chips duplicate their number of transistors every 18 - 24 months. A similar trend is observed in the performance achieved by commercial multiprocesors. Figure 1.2.a and Figure 1.2.b show the number of transistors and MIPS (million of instructions per second), respectively, for different Intel processors at their introduction year in the market. Also, in both figures, upper line and bottom line represent a trend that duplicates every 18 and 24 months, respectively. Additionally, PCs decrease their cost/performance ratio by 80% every year, while the one for supercomputers is decreased only between 20% and 30% [45]. All this together has shifted the use of proprietary parallel computers to a small and specific market.

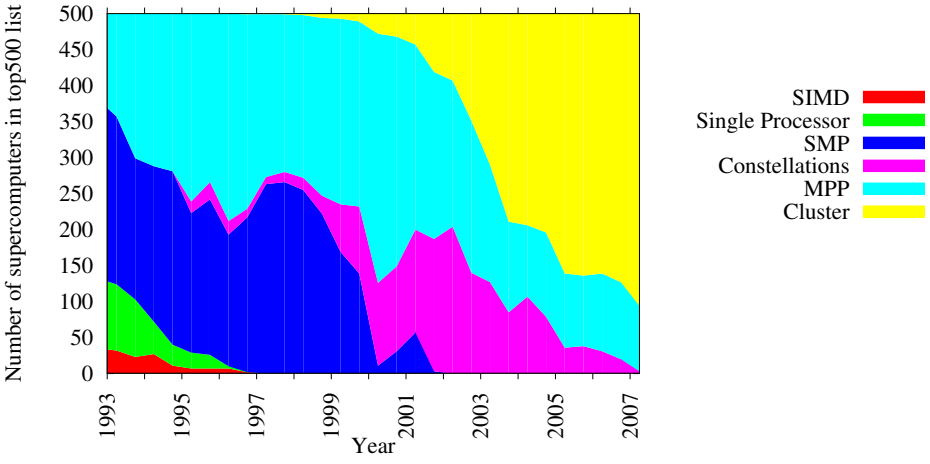
Following this trend, more and more cluster-based systems are being included in the *top500* list of supercomputers [39]. Figure 1.3 shows the number of supercomputers in the *top500* list classified by architecture. It is clear that in the last years cluster-based systems had become the most used ones in the *top500* list. As an example, the Abe system [16] with 2,400 quad-core (9,600 cores) Intel Xeon 64 processors at a frequency of 2.3 GHz is in the eighth position, the Mare Nostrum system [8] with



(a) (b)

Figure 1.2: Chip and performance evolution: (a) Growth of transistor counts for Intel processors (dots) and Moore's Law, (b) performance measured in MIPS for Intel processors.

10,240 processors and a computation capacity of 94.21 teraflops is in the ninth position, and currently is the most powerful supercomputer in Europe. Additionally, another domain where PC clusters are dominating is the Internet. Examples of Internet servers based on clusters of PCs are AOL [18], Google [11], Amazon [7] or Yahoo [41].



(a)

Figure 1.3: Architectures used by supercomputers on the *top500* list.

One of the key components that has enabled PC clusters as a platform for High-Performance Computing (HPC) is the emergence of high-performance interconnec-

tion networks like Quadrics [123], Myrinet [49], and InfiniBand [89]. Examples of these networks are the MareNostrum with 10240 PPC processors using Myrinet, the SGI Altix ICE-8200 with 14336 Xeon quad core using InfiniBand, and the Tera-10 with 9968 Itanium2 processors at 1.6 GHz using Quadrics. These systems are on the 3^{rd}, the 13^{th}, and the 19^{th} position on the *top500*, respectively. These commercial network technologies easily scale to a number of nodes of hundreds and thousands. Additionally, one key benefit of these systems is that upgrading the system is quite easy as more memory, processors or nodes can be added to the current system. In addition, these systems can be easily repaired and maintained, simply by replacing the damaged component by another with the same or higher performance. This leads to an efficient maintenance model.

In addition to the offered bandwidth and reduced latencies of such networks, their interconnection flexibility also promotes their use for building HPC clusters for small- and medium-sized companies, due to the low cost of PCs. Typically, these companies connect all their distributed computers using a high performance network. Although regular topologies should be preferred for performance reasons, the interconnection network must be designed, in particular its routing scheme, to support irregular topologies. Indeed, due to space restrictions, the network topology[2] may become completely irregular. In addition to this the occurrence of faults in the network components (links, switches) may convert a regular topology into an irregular one.

1.1 Motivation and Objectives

In many of the cluster-based systems, providing fault tolerance is a requirement and is becoming a necessity. These systems use very large number of components (processors, switches, and links). Each individual component can fail, and thus the probability of failure of the entire system increases with the number of components. Although switches and links are robust, they work close to their technological limit, and therefore they are prone to faults. Increasing clock frequency leads to higher power dissipation, and a higher heating could lead to premature faults. So, fault-tolerant mechanisms in cluster-based systems are becoming a key issue.

Most of the fault-tolerant routing strategies proposed in the literature for massively parallel computers are not suitable for clusters (see Chapter 6 in [77] for a description of some of the most outstanding approaches). This is because they often require certain hardware support that, unlike the proprietary network technologies commonly used in parallel computers, is not provided by the current commercial interconnect technologies [49, 89]. Additionally, these routing strategies have

[2]Network topology is the physical connection scheme of end nodes and switches. We classify them in Section 2.1.6.

been normally designed for specific regular network topologies, like meshes and tori. However, the switch interconnection pattern in clusters can even be irregular. Furthermore, they cannot be applied to systems with deterministic routing algorithms (this kind of routing is the most used in cluster networks), which prevents packets from circumventing the faulty components found along their paths. Also, some of the routing strategies need to perform virtual channel transitions when the packet is blocked due to a fault. However, in PC clusters virtual channels either are not supported by the interconnect technology or cannot be selected at routing time.

Therefore, the main objective of this thesis is to provide fault tolerant solutions for interconnection networks used to build PC clusters. To reach this overall objective we need to perform several smaller steps. Thus, the decomposed objectives are:

- Analysis of previous existing works related with fault tolerance on interconnection networks.
- Analysis of existing interconnection networks for PC clusters, mainly InfiniBand.
- Development of different fault tolerance mechanisms suitable for PC cluster interconnection networks. These solutions need to provide the maximum fault tolerance degree while reducing the traffic lost due to failures, maintaining both the required number of resources (e.g., virtual channels or service levels) and the elapsed time to tolerate the failure to low values.

1.2 Contributions

As commented above, the fault tolerant routing techniques proposed in the literature for parallel computers (e.g., MPPs) are not applicable to networks with deterministic routing, as it is the case of Myrinet and InfiniBand, which are commonly used as commercial network technologies.

More specifically, in this thesis we develop new fault tolerant techniques for network technologies whose switches support deterministic routing based on routing tables. In particular, and without loss of generality, we focus on the case of InfiniBand networks. As concrete contributions of the thesis, we propose:

- The design of a set of simple fault-tolerant routing methodologies based on the selection of paths at the source end nodes. With these methodologies a set of alternative disjoint paths is provided for each pair of end nodes, in such a way that failures are avoided by using alternative paths.
- The design of a fault-tolerant strategy based on routing in stages. The main goal of this strategy consists of computing alternative paths in a time-efficient

manner. The idea consists of composing new paths taking fragments of existing paths, where the first fragment starts at the source and the last fragment ends at the destination.

- The design of a fault-tolerant strategy based on applying dynamic reconfiguration of the entire network. In particular, in the presence of failures, the routing algorithm applied is changed partially or completely.

All these strategies are described and evaluated in this thesis. The aim of the techniques proposed is to provide an efficient way of dealing with failures in networks for clusters, specially in systems with an InfiniBand network.

1.3 Organization

This dissertation is organized as follows. First, we introduce the background on interconnection networks for HPC clusters in Chapter 2. Then, in Chapter 3, we introduce the concepts and definitions of fault tolerance in networks, together with an overview of the state of the art in the area.

In Chapters 4, 5, and 6, we present the contributions of this dissertation, describing the proposed mechanisms based on disjoint paths, routing in stages and reconfiguration, respectively.

In Chapter 7 the simulation model and the evaluation methodology are described. In Chapter 8 the contributions are evaluated and analyzed. Finally, in Chapter 9 the contributions and conclusions are summarized. Related publications of the contributions are also presented together with future directions in the research.

Chapter 2

Interconnection Networks for HPC Clusters

In Chapter 1, clusters were presented as a cost-effective alternative to parallel computers. These systems not only require powerful processors, but also a very efficient interconnection network offering a high bandwidth and a low latency, able to cope with the communication needs imposed by the high processing and data transfer rates reached by current processors. Additionally, clusters have the necessity for interconnecting an increasing number of storage devices to cope with the access bandwidth required by applications that manage huge amounts of information.

Thus, the global performance of the system can be affected by the performance of the interconnect. The bandwidth and latency are the main performance metrics of an interconnection network. Bandwidth[1] measures how much data a network can deliver at a time, and latency refers to the time required by an interconnect to deliver a data packet.

Traditional (and old fashioned) interconnection networks like Ethernet [1] and FDDI [2] are not a good solution to build an HPC cluster, as these networks exhibit high transmission latencies which oscillate between 0.1 ms and 1 ms, and reduced bandwidth between 1 and 100 Mb/s. An additional limitation of these networks is the fragmentation of the bandwidth between all the processors of the system. Therefore, these networks do not allow increasing the number of processors in the system without affecting its global performance. In fact, for a medium number of end nodes, the interconnection network becomes the bottleneck of the system. Thus, in those systems applications have to wait long time for the requested piece of information,

[1] We refer to the term bandwidth as the maximum rate at which information can be transferred, where information includes packet header and payload. The bandwidth is traditionally measured as bits per second although bytes per second is sometimes used. The term bandwidth is also used to measure the speed of network links.

stopping doing any useful task until the requested information is received.

Moreover, the interprocess communication (IPC) characteristics have a significant effect on application performance. As a clear example, in the Earth Simulator [10], the internode MPI commutation latency is 8.6 μs [93] whereas the processor cycle time is 2 ns. This 4,300x factor significantly limits how parallel computation can be exploited.

With the new Ethernet technologies (Fast Ethernet [3] and Gigabit Ethernet [20]) the offered bandwidth has been increased remarkably. Moreover, these networks are switch-based, which allows concurrent communication (higher bandwidth). However, transmission latencies are still high (tens or hundreds of microseconds).

Currently, new switch-based interconnection network technologies are available with capacities varying from 1 Gb/s to several Gb/s, offering at the same time low latencies. Examples of these networks are Myrinet [49], Quadrics [123], and InfiniBand [89]. Table 2.1 summarizes performance numbers of different network technologies and implementations extracted from [9, 30, 40, 32, 43, 79, 146].

Technology	Measured latency	Measured bandwidth
PathScale InfiniPath(InfiniBand)	1.31 μs	954MBps
Cray RapidArray	1.63 μs	2 GB/s per link
Quadrics	4.89 μs	$\sim$875-910 MBps
Myrinet	19.00 μs	240 MBps
Myrinet(XP2)	5.7μs	$\sim$495 MBps
Gigabit Ethernet	42.23 μs	$\sim$125 MBps
Fast Ethernet	603.15 μs	$\sim$12 MBps

Table 2.1: Lowest measured interconnect latency (smaller number is better), and bandwidth (larger number is better) in interconnection networks.

Thus, the interconnection network must be efficient enough in terms of bandwidth and latency to be able to support the traffic features required by current applications. In what follows, we provide an overview of the required background on interconnection networks for HPC clusters.

2.1 Network Design for Clusters

The interconnection network in a cluster allows the communication between end nodes, which are connected to the network by Network Interface Cards (NIC). The main components in a network for clusters are switches and links. Switches interconnect end nodes and other switches by point-to-point links. Switches are in charge of transmitting data through the network.

The interconnection network design is influenced mainly by three design choices: switching technique, network topology, and routing strategy. The switching mechanism (described in detail in Section 2.1.2) determines the advance of data, and defines how and when network resources are assigned for such a task. Network topology is the physical interconnection pattern of the end nodes and switches. We classify them in Section 2.1.6. Finally, the routing algorithm determines the path to follow by packets when going from one end node to another in the network. We describe them in more detail in Section 2.1.7. Additionally, there are many other design choices like flow control, virtual channels, and arbitration, which are covered in the following sections.

2.1.1 Data Units

Before describing more complex structures, we first focus on the different types of data units we might find in a network. Figure 2.1 shows the different data units.

The data to be transmitted is arranged into different units. The message is the largest unit. It is referred to as the data to be transmitted between two end nodes. Messages are often packetized into smaller data units, referred to as packets, depending on the switching mechanism.

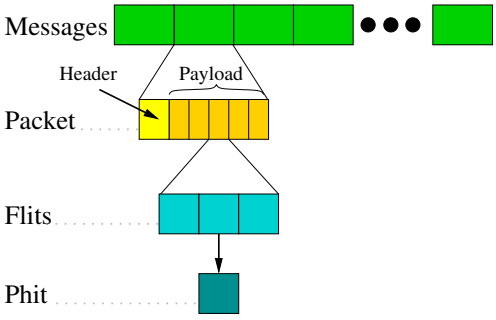


Figure 2.1: Different data units in an interconnection network.

The structure of a packet is composed of a header and a payload (and sometimes even a tail). The header contains routing and control information, which is used to route the packet from its source to its destination. The payload contains data to be delivered to the destination.

Packets are divided into flits (flow control units). Flits are the minimum data that can be controlled over a link by the flow control mechanism. To transmit a single flit multiple link cycles may be used. The piece of data transmitted in a single cycle over the link is referred to as *phit* (physical unit).

2.1.2 Switching

The switching technique establishes how network resources are assigned to messages or packets, and what to do when the message (or packet) is blocked. It includes the connection between the input and the output ports of a switch and how the transfer of information takes place. There are three main switching techniques in current networks: *store & forward* [62], *virtual cut-through* [94], and *wormhole* [65].

In *store & forward* (SAF) (Figure 2.2.a) a packet arriving to a switch is stored in a buffer associated to the input port, and only when it is completely received is routed and forwarded. The packet is forwarded to the next switch or destination node when it has enough buffer space to be completely stored at the next node or switch. This mechanism has the disadvantage that the packet latency is proportional to the distance between the source and destination. An additional problem is the limitation of the packet size. The maximum packet size is limited by the size of the buffer. ATM (Asynchronous Transfer Mode) networks [108] use this switching mechanism.

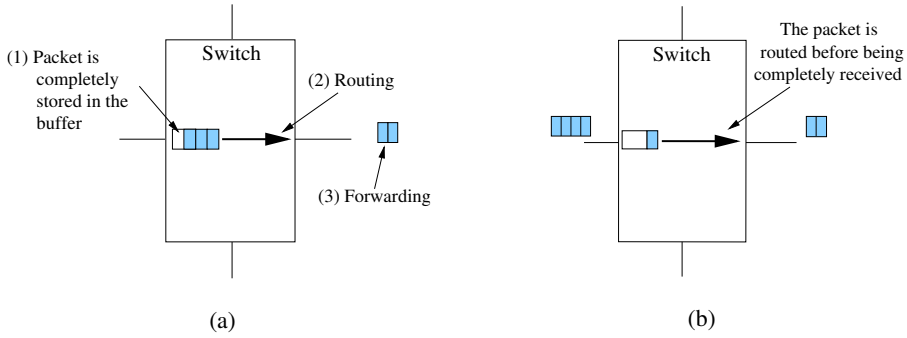


Figure 2.2: Switching mechanisms: (a) *store & forward* and (b) *virtual cut-through*.

In *virtual cut-through* (VCT) switching a packet is relayed towards the next switch as soon as its head is decoded, provided the selected output channel is free, without waiting for the complete reception of the packet (Figure 2.2.b). When compared with SAF, the latency experienced by a packet is drastically reduced. However, if the selected output channel is busy, the packet is completely stored in the input buffer at the current switch until the output becomes available. Therefore, *virtual cut-through* switching behaves like *store & forward* when the output channel is busy (i.e., as long as the traffic load becomes high). With this switching mechanism, in absence of contention[2], the latency of the packet is largely insensitive to the distance. As an example, IBA switches use the *virtual cut-through* switching mechanism.

[2]Contention occurs when two or more packets request the same output channel. In this situation, only one packet is granted the requested channel, whereas the rest of packets are blocked until the output channel becomes free.

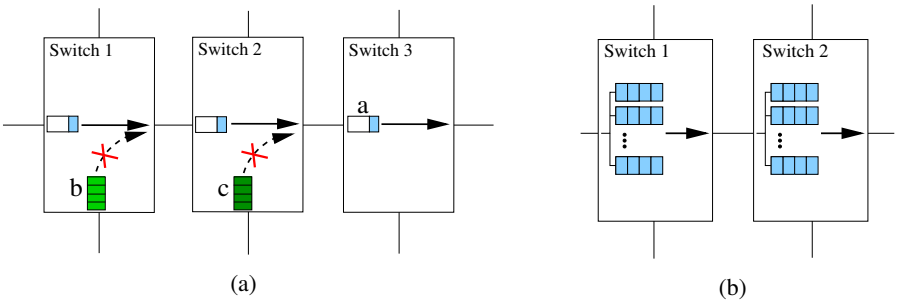


Figure 2.3: (a) Example of blocked messages in *wormhole*. (b) Virtual channels.

In *wormhole* (WH) switching, the message (not the packet) is forwarded immediately towards the next switch before the entire message is received, as in *virtual cut-through*. However, unlike *virtual cut-through*, message size is not limited by buffer size. Only space for a few flits is needed at each buffer. Therefore, when the required output channel is busy, the message remains stored in place throughout the buffers that it is occupying at different intermediate switches across the network. One major advantage of *wormhole* switching is that switches can be implemented with little storage space (buffers). Thus, *wormhole* allows to design fast, simple and low cost switches. However, the network throughput achieved by *wormhole* is, often, low. *Wormhole* switching causes a higher *contention* degree in the network, due to the fact that messages, when blocked, keep the buffers and channels being traversed. As a consequence, all the messages that request any of the resources occupied by the blocked message are also blocked, which block in turn other messages, and thus more and more messages become blocked. All this entails to a low utilization of the channels, which causes a low throughput in the network. Figure 2.3.a is an example, in which message a becomes blocked, occupying the channels of the three visited switches. These channels cannot be used by messages b and c until the blocked message continues its way towards its destination and releases the resources. Additionally, while being blocked, messages b and c will block also other messages. Myrinet is an example of network using *wormhole* switching.

2.1.3 Virtual Channels

In order to alleviate the problem of low throughput (due to the contention) achieved by the *wormhole* switching mechanism, virtual channels were proposed [59]. Instead of associating only one buffer to each physical channel, multiple buffers are associated (see Figure 2.3.b), in such a way the storage space associated to the physical channel is decoupled from the channel bandwidth. With the use of virtual channels,

the physical channel bandwidth is shared by the messages instead of being associated exclusively to only one message. If a message blocks (due to the fact that the header stored in the reserved buffer cannot advance) the channel will be used by other messages, thus avoiding wasting the channel bandwidth.

Indeed, although only one physical channel exists, the information can be transmitted as if there were several channels. We denominate to these channels as virtual channels, and each of them has associated a dedicated set of buffers.

The use of virtual channels provides an important increase in the throughput of the network, at the expense of increasing the control logic associated to each physical channel. In particular, flow control should be provided at virtual channel level. The disadvantage of this mechanism is that the available bandwidth for each message is reduced, because it is distributed between the messages sharing the physical channel. Thus, the messages will advance at a lower speed. In addition, if at least one of the channels used by the message is multiplexed, then the totality of the message advances at a lower speed. Virtual channels can be used as a complementary mechanism together with wormhole and virtual cut-through.

2.1.4 Flow Control

Packets can be transmitted through a channel when it is available. However, buffer space is required to temporarily store the packet. In some cases there is no more available buffer space, and a flow control mechanism is required to stop the transmission (other options not commonly used in high performance networks is to drop or reroute the packet).

A *flow control* mechanism determines when packets can be forwarded, because it is possible that channel bandwidth (or buffer capacity) can not be allocated. Commonly, flow control is tightly coupled to the switching technique. Flow control requires to know the availability of buffers for storing the next flit (*wormhole*) or the next packet (*store & forward* and *cut-through*).

There are three flow control mechanisms commonly used for managing the availability of buffers: *credit-based*, *on/off*, and *ack/nack* mechanisms. These mechanisms should be designed with the aim of maximizing the effective channel bandwidth.

With *credit-based* flow control, the upstream end of a channel keeps a count of credits, which corresponds to the number of free flit buffers[3]at the downstream end of the channel. Then, each time a flit is forwarded, a downstream buffer is consumed and the appropriate counter is decremented. If the counter reaches zero, the corresponding downstream buffer may be full and no further flits can be forwarded. Once the downstream end of the channel forwards a flit and frees the associated buffer, it

[3]In store & forward and cut-through switching number of credits corresponds to the number of packets that can be mapped.

sends back a credit to the upstream end of the channel, causing the associated buffer counter to be incremented (see Figure 2.4.a).

A potential drawback of *credit-based* flow control is a one-to-one correspondence between flits and credits. For each flit sent downstream, a corresponding credit is eventually sent back upstream. This requires a significant amount of upstream signaling, especially for small flits, and can represent a large overhead.

Stop/go flow control, also known as *on/off*, can greatly reduce the amount of upstream signaling in certain cases. With this method, the downstream state is a single control bit that represents whether the upstream node is permitted to send (*on*) or not (*off*). The round-trip time (RTT) is defined as the time that elapses between the control signal is sent upstream until traffic is stopped or resumed (this time includes the travel time of the control signal and the required time by the injected traffic to reach the end node). *Stop/go* flow control generates much less control traffic than *credit-based* flow control because a signal is sent back upstream only when it is necessary to change this state. In particular, it is sent when the number of free buffers crosses a certain threshold. This threshold corresponds to the required buffer size to store the traffic during the round-trip delay (Figure 2.4.b. shows an example of *on/off* flow control, where the stop and go thresholds are referred to as *high watermark* and *low watermark*, respectively). Thus, the required buffer is at least two times the traffic which can be transmitted during the round-trip delay.

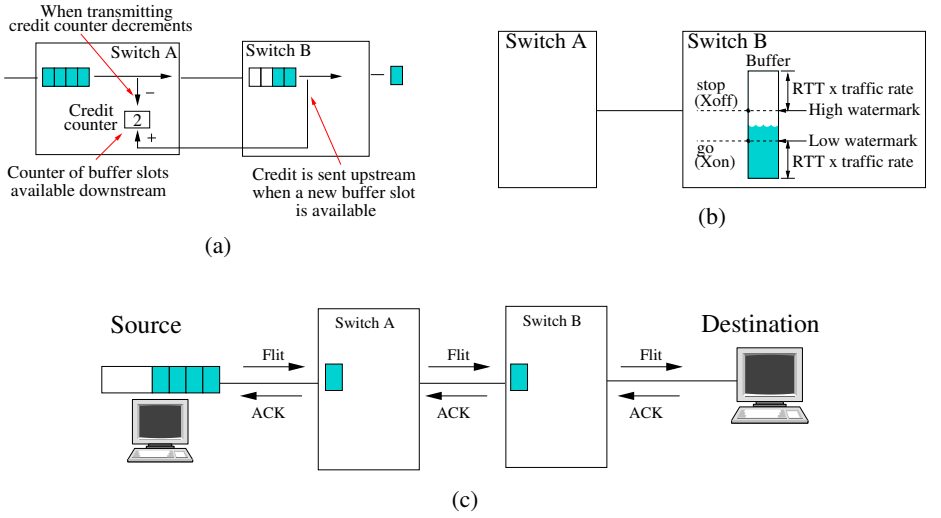


Figure 2.4: Examples of flow control mechanisms: (a) *credit-based*, (b) *on/off*, and (c) *ack/nack*.

Both *credit-based* and *on/off* flow control methods must consider the round-trip delay t_{rt} to avoid buffer under utilization. Also, notice that both mechanisms back-pressure the sender to avoid dropping packets.

With *ack/nack* flow control, each switch/end node sends flits whenever they become available. If the downstream node has a buffer available when the flit arrives, then the flit is accepted and an acknowledge (*ack*) is sent to the upstream node, otherwise the flit is dropped and a negative acknowledgment (*nack*) is sent. The upstream node, then, keeps each flit until it receives an *ack*.

Ack/nack flow control reduces buffer needs. Unfortunately there is no net gain because buffers are needed also at the transmitter side waiting for an acknowledgment, making ack/nack flow control less efficient than credit-based flow control in terms of buffer usage. It is also inefficient in its use of bandwidth because flits must be dropped when no buffer is available. *Ack/nack* flow control is also referred to as *optimistic* flow control, since it should be used when the chances of having available buffers is high.

As a summary, *credit-based* flow control is typically used in systems with small number of buffers, and *on/off* flow control is used in most systems that have large number of flit buffers. However, *ack/nack* flow control is rarely used because of its buffer and bandwidth inefficiency.

2.1.5 Arbitration

Switches as shown in Figure 2.5, have multiple ports with associated buffers and channels. In this context, it is possible that several incoming messages (from different input ports) request the same output port or resource at the same time. In order to describe which of the requesters wins access to the requested resource, an arbitration mechanism is required. The arbiter assigns the requested resource to one requester at a time, and the other ones keep waiting. We consider the arbitration time as the time required by the arbiter to determine the assignment of a requested resource. Fast arbitration algorithms are crucial to keep low latencies through the switch.

A key property of an arbiter is fairness. The arbiter must provide the same chance to any request in order to be fair. Different arbiters have been proposed: examples are [67]: *fixed priority*, *oblivious*, and *round robin*.

The *fixed priority* arbiter, serves always requests in the same order. Thus, the order defines the priority assigned to each of them. The *oblivious* arbiter, however, is based on selecting randomly a request each time. Both mechanisms do not provide fairness, since eventually multiple incoming requests from the same input port will (or may have) higher priorities than a request waiting in another input port. In order to guarantee fairness the *round-robin* mechanism can be used. It is based on assigning the lowest priority to the last served port on the next arbitration cycle.

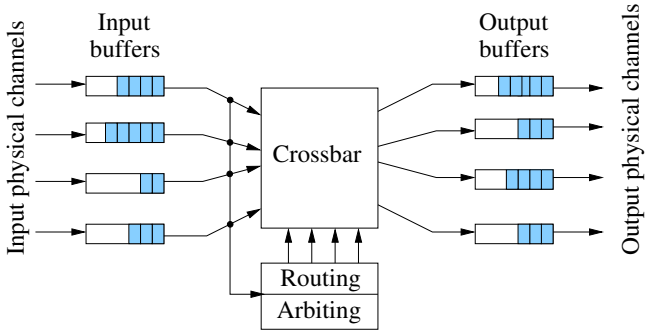


Figure 2.5: Internal model of a switch.

2.1.6 Topology

Topology is defined as the arrangement of switches and end nodes, and the connection pattern through links. The topological distance between two end nodes is defined as the minimum number of links that must be traversed to reach one of them from the other.

Clusters of PCs are very flexible in the way network resources can be connected. Indeed, the topology can be defined completely by the user. Thus, although regular topologies should be preferred for performance reasons, irregular topologies are also possible. In what follows, we provide a classification of different topologies.

A. *Shared medium networks*: These networks, also known as buses, share the communication medium, which has a limited bandwidth. Therefore, the number of devices that can be connected is limited (scalability problems). Buses become a bottleneck as long as the transfer rate and/or the number of connected devices increases. Additionally, an arbitration mechanism is required to access the bus.

   In these networks the implementation of broadcast is simple and efficient. Local area networks such as Ethernet, Fast Ethernet or FDDI and the internal buses of multiprocessor systems are examples of shared medium networks.

B. *Direct networks*: Every end node in the system includes a switch/router, which handles message communication among end nodes. For this reason direct networks are also known as *router-based networks* [77]. The topology is defined, thus, by connecting end nodes among them accordingly to a certain interconnection pattern.

   Most of the implemented systems have an orthogonal topology. In these topologies, nodes can be allocated into a n–dimensional space, with k nodes along

each dimension. The position of a node is defined by its coordinates into the space, that is, a n–tuple $(x_{n-1}, \ldots, x_1, x_0)$, so that $0 \leq x_i < k$. In particular, the most popular topology is the k–ary n–cube topology, where k is the number of nodes interconnected in each of the n dimensions. Every end node has at least one link crossing each dimension. The distance between every pair of end nodes is the sum of the offsets in all the dimensions.

There are two topologies, tori and hypercubes, on which every end node has two links in each dimension. In particular, hypercubes are a special case of k–ary n–cube in which k=2. Figures 2.6.b and 2.6.c show a 2D torus and a binary hypercube, respectively. Examples of real systems are: the *KSR first-level ring* [131] which uses a 1D torus network, the *Intel/CMU iWarp* [24] and *Alpha 21364* [119] which use a 2D torus network, the *Cray/SGI T3D* [139], *Cray T3E systems* [140] and *BlueGene/L* [44] which use a 3D torus network, and the *Intel iPSC* [81] and the *nCUBE* [118] which use an hypercube network.

In the case of tori, the links used to connect nodes at the boundaries of each dimension are referred to as wraparound links. Meshes are similar to tori, but without the wraparound links. Unlike k–ary n–cube, in a mesh topology there are some nodes with a different number of ports (neighbours), in particular, those placed at the borders of the network. Figure 2.6.a shows a 2D mesh. Real systems which use this topology are the Intel Paragon [78] and the MIT J-Machine [122], using a 2D and 3D mesh, respectively.

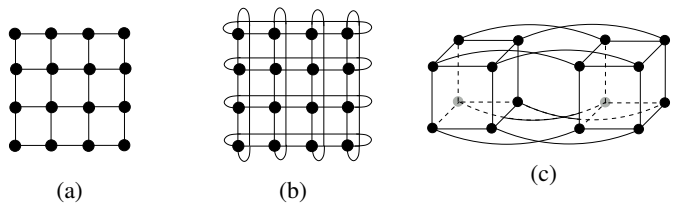


Figure 2.6: Examples of direct topologies: (a) 2D mesh, (b) 2D torus, and (c) hypercube (n–cube).

C. *Indirect networks*: In these networks, the end nodes are interconnected through switches, and the connection to the switch is made by a Network Interface Card (NIC). Each switch port can be connected to an end node or to a switch, thus each switch can be connected to several end nodes and/or switches. The most common indirect topologies are crossbars, multistage networks, and irregular networks.

- A crossbar allows a connection to be established between any input-output pair, whenever the output is unassigned (is not being used by a

previous connection). As complexity grows quadratically with the number of ports, this topology does not scale with system size. Figure 2.7.a shows an example.

- To overcome the scalability problem of a crossbar multistage interconnection networks (MINs) are used. Switches are organized in stages and end nodes are connected only to switches on the first stage (case of unidirectional MINs or fat-trees)[4]. In unidirectional MINs, the distance between any pair of nodes is the number of stages plus two (external links connecting the end nodes). Figure 2.7.b shows an example of a multistage network (fat-tree).
- Finally, in an irregular network, the connection between switches and end nodes does not follow any regular pattern. Figure 2.7.c shows an irregular topology connecting 11 end nodes.

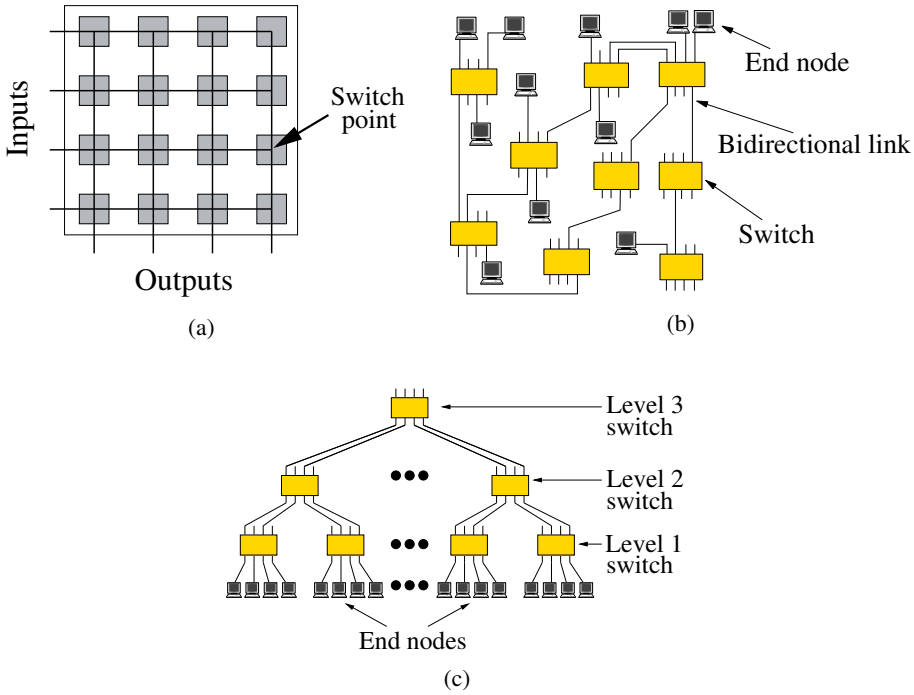


Figure 2.7: Examples of indirect topologies: (a) 4×4 crossbar, (b) irregular network, and (c) bidirectional multistage network.

[4]Networks are referred to as bidirectional MINs, when they are organized in stages, and end nodes are connected to the first and last stages.

As can be deduced, the differences between direct and indirect networks are very subtle. The functionality of a router (in a direct network) and a switch (in an indirect network) is the same. A network can be seen as a set of point to point links that interconnect switches with a certain interconnection pattern, in such a way that a switch can be connected to one end node, to several end nodes or to none at all. A direct network is equivalent to an indirect network in which every switch is connected to a single end node. A crossbar network is a network with a single switch connecting all the end nodes. A multi-stage network corresponds to the case in which the switches are organized in several stages, so that switches belonging to intermediate stages are not connected with any end node. This unified view allows us to apply the same strategies to route messages through the network. Some common topological metrics of direct and indirect networks are:

- Diameter: Maximum distance between any pair of end nodes.
- Degree: Number of channels of each switch or router.
- Regularity: All the routers/switches have the same degree.
- Symmetry: The network looks like the same from any router/switch point of view.
- Connectivity: The minimum number of switches/links necessary to disconnect the network.

2.1.7 Routing Algorithm

Once defined the topology, we need to route packets within the network. The routing algorithm determines the path taken across the network by each packet from its source to its destination. A path is an ordered set of channels, which starts at its source and ends at its destination. The length of a path is the total number of channels traversed along it.

Many routing algorithms have been proposed in the last decades for different interconnection network domains. In this section, we provide a taxonomy of routing algorithms, and describe later some algorithms suitable for PC clusters.

2.1.7.1 Taxonomy

Figure 2.8 shows a taxonomy of the routing algorithms. This taxonomy has been obtained from [77]. Routing algorithms can be classified according to the number of destinations. So, packets can be sent to either only one destination (*unicast* routing) or multiple destinations (*multicast* routing). In particular, when using unicast routing

the path followed by messages/packets can be defined in different ways. The first one, referred to as *centralized* routing, consists of all paths being computed at a centralized controller. A second way, referred to as *source* routing, consists of each source end node computing the entire path that each of its packets must follow. Typically, all the information about the ports to be traversed is included in the packet header, thus, switches only have to forward the packet according to this information. Each switch reads the appropriate element at the packet header and uses it to select the output channel. Notice that the packet has the path fixed on its header from the source and it cannot be changed. Figure 2.9.a illustrates the source routing mechanism. This mechanism has the main advantage of switches being simpler and the routing delay being small. The main disadvantage is that the size of the header depends on the distance to the destination. This size can be considerably high with respect to the number of bytes to transmit (specially when short messages are sent).

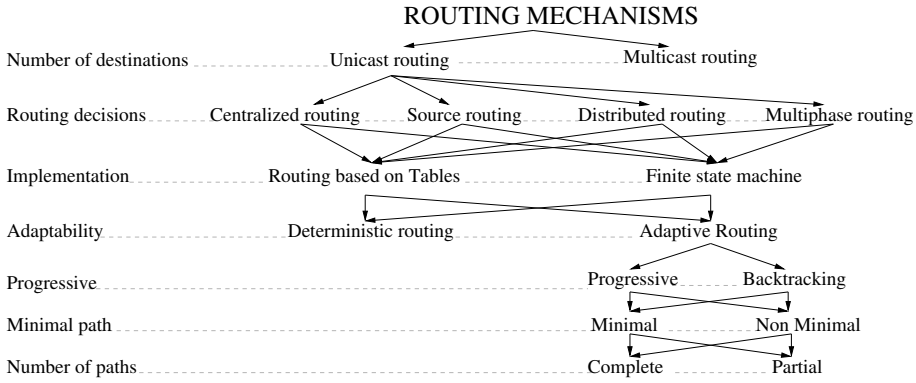


Figure 2.8: Taxonomy of the routing algorithms.

In *distributed* routing, however, paths are defined dynamically while packets travel through the network. The information of the path is distributed in the network, so each switch knows the output channel to be used based on the destination of the packet (in some cases also considering the input channel as well). Greater flexibility is provided as switches are able to decide the output channel, probably based on the status of the output ports. Also, the size of the packet header is small and constant.

Myrinet is an example of a network using source routing. Figure 2.9.a shows a packet being routed in a switch using source routing. IBA uses deterministic distributed routing based on tables (also uses source routing for control packets). Figure 2.9.b illustrates the distributed routing mechanism based on tables, assuming deterministic routing. In the figure the unique output port is selected as a function of the destination ID indicated by the packet header and the input port.

Hybrid schemes are also possible. In [77] hybrid schemes are referred to as *multiphase* routing. In *multiphase* routing, the source node computes some destination

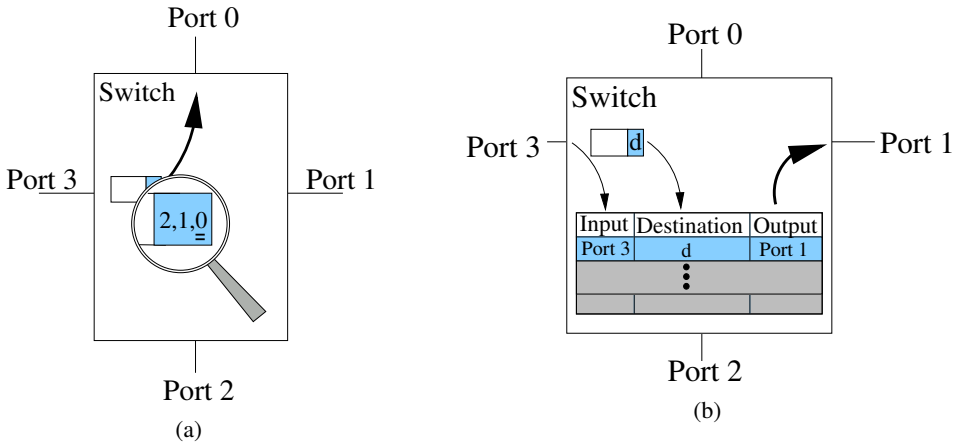


Figure 2.9: Different routing mechanisms: (a) source routing and (b) distributed routing based on tables.

nodes. The path between them is established in a distributed manner.

Routing algorithms can be implemented in different ways. The most common ones consist of either looking at a routing table (table look up) or using a finite-state machine.

Additionally, routing can be classified as *deterministic* or *adaptive*. When using *deterministic routing* the path is unique and cannot be changed, whereas *adaptive routing* allows modifying the path depending on different criteria (e.g., the network traffic status).

Depending on whether the message can backtrack or not, *adaptive* routing can be further classified as *progressive* or *backtracking*. In both cases the adaptive routing is referred to as *minimal* routing when the path can only use channels that get the packet closer to its destination (it is also referred to as *profitable* routing). Otherwise, it is referred to as *non minimal* routing (also known as *misrouting*). An adaptive routing algorithm is referred to as *fully adaptive* when it can use any possible routing option, taking into account that these options are only minimal. Alternatively, it can be referred to as *partial adaptive* when it is not allowed to take all the possible routing options.

2.1.7.2 The Deadlock Problem

Deadlock occurs when some packets cannot advance toward their destination because the buffers/channels requested are not released by the packets using them. A dependence is defined between the resource held by a packet and the resource requested

by the same packet. Packets involved in a deadlock (cycle of resource dependences) keep blocked forever, waiting for the transmission of another packet also in the deadlock cycle. This situation arises because network resources are finite, in particular buffer capacity. As a deadlock blocks the network, it is extremely important to avoid or remove them when designing a routing algorithm.

For example, Figure 2.10.a shows a deadlock situation. In the figure, the involved switches are labeled A, B, C and D. The involved buffers are full, and the requested output link by the message placed at the head of each buffer is represented in the figure by a curved arrow. Figure shows that these dependencies form a cycle which can never be broken as none of the messages can advance.

The probability to reach a deadlock situation increases with network traffic and decreases with the amount of buffer storage. In both *store-and-forward* and *virtual-cut-through* switching mechanisms, deadlock may occur only as a result of cyclic dependencies across adjacent channels. This contrasts with *wormhole* switching, where a packet may block holding multiple channels along its path. Therefore *wormhole* is prone to deadlocks.

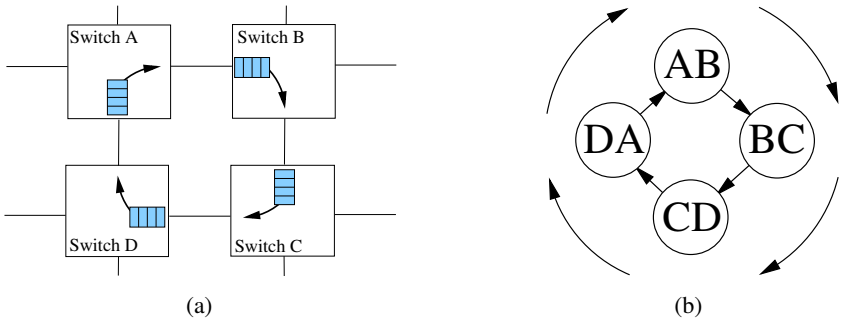


Figure 2.10: Example of a deadlock situation: (a) a deadlock situation, and (b) example of a cycle in the CDG.

There are two different ways to deal with deadlocks, the deadlock avoidance and deadlock recovery.

First, we define the concepts of dependences between channels and the channel dependency graph, because the deadlock avoidance techniques are based on them.

A dependence from channel i to channel j is defined when there is a routing option which will route a packet holding the channel i into the channel j. Using this definition, the channel dependency graph (CDG) is generated considering all the channel dependencies. The graph is a directed graph where the vertexes are the channels and the edges represent the dependences between them. The CDG can be used to detect the existence of cyclic dependences between channels [66].

Figure 2.10.b shows the corresponding CDG when considering the channels involved in the deadlock situation in Figure 2.10.a. This second figure presents the dependences between channels. Notice that there is a cycle in the CDG.

The deadlock avoidance theorem when using deterministic routing of messages was proposed by Dally and Seitz in [64]. They demonstrated that deterministic routing of messages in the network is deadlock-free *if and only if* the channel dependency graph does not contain cycles. This can be achieved by enforcing some routing restrictions. Basically, a deterministic routing algorithm defines routing restrictions between pairs of channels. A routing restriction between two channels forbids the use of the second channel after using the first one. In some cases, such as k–ary n–cube topologies, virtual channels must be introduced to break cycles in the CDG. In particular, in tori physical channels must be splitted into two virtual channels to avoid deadlock under deterministic routing.

Notice, that Dally and Seitz theorem is a necessary and sufficient condition to guarantee deadlock freedom when applied to deterministic routing. However, when applied to adaptive routing, the absence of cycles in the CDG represents only a sufficient condition. That relaxed condition allowed to design great quantity of adaptive routing algorithms. In most cases, at the expense of introducing large number of virtual channels to break cyclic channel dependences in the CDG [63, 103].

Later, Duato proposed in [74] a deadlock-avoidance theory for adaptive routing. According to this theory a routing function is deadlock-free *if and only if* there exists a routing subfunction that is connected and has no cycles in its extended channel dependency graph. That is, it is possible to guarantee that an adaptive routing algorithm is deadlock-free despite allowing cyclic channel dependencies into the CDG, provided the existence of a routing subfunction that has no cycles in its extended channel dependency graph. The channels used by the aforementioned routing subfunction are referred to as escape channels. Removing the necessity of breaking cycles into the CDG allowed to design deadlock-free adaptive routing algorithms by using the minimum number of network resources (virtual channels).

The alternative way to deal with the deadlock problem consists of allowing cycles in their channel dependency graph, and using a deadlock recovery mechanism when a deadlock situation arises. For instance, true *fully adaptive* routing [47, 46], which allows any path available in the topology with no restrictions, allows cycles and requires a deadlock recovery technique for guaranteeing deadlock freedom.

2.1.7.3 The Livelock Problem

Livelock occurs when packets continue to move through the network, but never reach their destination.

A simple solution to the livelock problem consists of limiting the maximum number of misroutings of the packet, or by giving higher priority to older packets in the

network. This can be implemented by labeling each packet. Such label can be used for storing a misrouting counter, or to store the packet age.

2.1.7.4 Routing Algorithms for Clusters

Many routing algorithms have been proposed in the literature, although most of them are not applicable to clusters. This is because either most routing algorithms require certain hardware support or resources not available in current interconnects for clusters, or they are defined for some particular topologies whereas clusters adopt any topology.

Dimension Order Routing (DOR) is one of the most simple and popular routing algorithms used in MPPs. DOR is a deterministic routing algorithm as it always supplies the same path between every pair of nodes. It requires, indeed, regular topologies like hypercubes, meshes, and tori. DOR routes all the packets by crossing dimensions always in the same order, reducing to zero the offset in one dimension before routing into the next dimension. Therefore, this deterministic routing algorithm avoids deadlock by prohibiting turns between dimensions out of the defined order. However, when DOR is applied to torus topologies cycles can still be formed along the wraparound links. To remove cycles in the CDG each physical channel must be split into two virtual channels [64] or an alternative mechanism, like bubble flow control [127], must be used. When applied to 2D meshes or tori DOR is also referred to as *XY routing algorithm* (first dimension X, then dimension Y). The four turns allowed by the XY routing algorithm are shown in Figure 2.11.

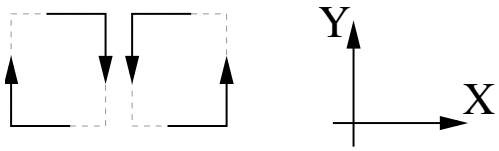


Figure 2.11: Allowed turns in XY routing algorithm.

Despite the fact that DOR can be applied to PC clusters, this routing scheme cannot be considered the most suitable for clusters. This is because DOR requires the use of regular topologies, whereas the topology in PC clusters may be irregular[5] due to either topological constraints of the user capability to define the network layout. As the topology can be modified by the user, routing schemes valid for any topology are preferable. They are known as generic routing or topology-agnostic routing algorithms.

[5]Notice also that a regular topology becomes an irregular one in the presence of faults.

The most popular topology-agnostic routing algorithm is *up*/down** (UD) [138]. This algorithm can be used in networks with either source routing or distributed routing, and can be applied to regular and irregular networks as well. Indeed, it has been used in commercial networks, including Myrinet [49], Gigabit Ethernet [20] switches (using the Spanning-Tree Protocol (STP)), and IBA (the OpenSM [27] software in InfiniBand uses an *up*/down** tree).

UD is based on an assignment of direction labels (*up* or *down*) to each link. To this end, the UD algorithm computes a spanning tree, selecting a certain node as root. Depending on the way the tree is computed we can have a BFS (Breadth-First-Spanning) tree [138] or a DFS (Depth-First-Spanning) tree [134]. The *up* direction is assigned to the channels going upwards the tree (towards the root) and the down direction to the channels going downwards. If both switches are at the same level, the direction is assigned based on the identifiers of the switches. For example, *up* direction is assigned to the channel that connects to a switch with a smaller identifier. The algorithm uses the following rule to avoid cycles: packets cannot use channels in the *up* direction after they have used channels in the *down* direction. Figure 2.12 shows the allocation of channels in an irregular network according to the *up*/down** routing algorithm based on a BFS Spanning tree. the main difference between usual UD algorithms based on BFS and DFS trees is that the DFS tree imposes less routing restrictions, allowing more packets follow minimal paths.

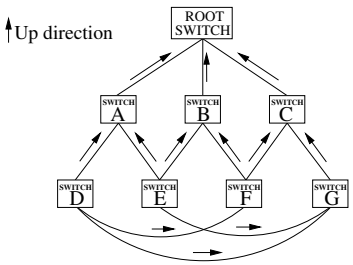


Figure 2.12: Link labels for the *up*/down** (BFS) routing algorithm.

Another routing algorithm we may consider for PC clusters is *smart-routing* [57]. It computes all possible paths between all source-destination pairs, building at the same time the CDG. Then, it looks for the existence of cycles in the CDG. By using an iterative process all cycles are broken (this process tries to balance the traffic over the channels). This process finalizes when the CDG has no cycles. Although *smart-routing* distributes better the traffic than other routing algorithms, it has the disadvantage of its high computational cost, since it uses a linear algorithm for balancing the traffic while removing cycles. *Smart-routing* can be used in networks with source routing and distributed routing.

Other deterministic routing algorithms are the *FleXible routing scheme* (FX) [133] and the *left-up-first turn* routing algorithm (*LTURN*) [96]. FX introduces unidirectional routing restrictions to break cycles. *LTURN*, however, is based on a logical BFS spanning tree and a left-right directed graph to provide deadlock freedom and to distribute the traffic over the network.

These routing algorithms (*UD, DFS, LTURN, FX*, and *SMART* routing) do not require the use of virtual channels. There are other deterministic routing algorithms, however, which require the use of virtual channels. Examples of them are *Multiple Roots* (*MRoots*) [107], the *Layered Shortest Path* (*LASH*) [142], the *Descending Layers* routing (DL) [97], and the *Transition Oriented Routing* (*TOR*) [135].

MRoots provides a solution to the congestion experienced when using *up*/down**. To achieve this, *MRoots* distributes traffic over multiple different *up*/down** trees each one on a different virtual channel. *LASH* routing algorithm provides deterministic shortest path routing by dividing the physical network into a set of virtual networks using separate virtual channels. Paths are mapped on a virtual channel only if the CDG of the corresponding virtual network is acyclic. To achieve minimal paths an unbounded number of virtual channels may be required.

Both routing algorithms (*MUD* and *LASH*), use multiple virtual channels and traffic being routed in one virtual channel can not migrate to another virtual channel. *TOR* and *DL* routing algorithms, however, allow virtual channel transitions (virtual channels are used in increasing order in order to guarantee deadlock freedom). *TOR* uses *up*/down** as the underlying routing algorithm in order to decide when traffic migrates between virtual channels. Similarly, *DL* uses *up*/down** or *LTURN* as the underlying routing.

On the other hand, adaptive routing algorithms can not be applied to clusters. For instance, *adaptive trail* [130] and *minimal adaptive* [125] routing algorithms cannot be used in networks with source routing (for example Myrinet and ASI [6]) neither in networks with deterministic distributed routing (like IBA), as they require switches to choose among different alternatives at routing time. Such a hardware support is not provided by current commercial interconnect technologies.

2.1.8 Network Reconfiguration

Once described the main characteristics of interconnection networks for clusters, for the sake of completeness, and due to the fact that this thesis is related to fault tolerance issues, we describe in this section the reconfiguration process of the network and the main existing reconfiguration proposals. By reconfiguration we mean the entire process change of the network (possibly due to a failure).

Network reconfiguration consists of three steps (See Figure 2.13). The first step is intended to detect network devices and their connections (i.e., topology), the second step consists of computing the new routing info by applying a routing algorithm,

whereas the last step distributes the new routing info among the network devices to update routing tables. Notice that a reconfiguration process finishes when the new routing algorithm has been completely assimilated (it is being used in all network components).

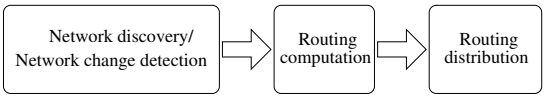


Figure 2.13: Global reconfiguration stages.

2.1.8.1 Static and Dynamic Reconfiguration

Reconfiguration techniques can be either static or dynamic. Static reconfiguration techniques require completely stopping the traffic in the network before changing any routing table, so the network is emptied [51, 106]. The routing algorithm used after the reconfiguration process is different. This implies that all the paths for each source-destination pair need to be computed. Static reconfiguration largely impacts on packet latency, due to network down-time, i.e., halting packet injection, which may cause a strong performance degradation during the reconfiguration process. This prevents static reconfiguration techniques from being used in systems with high performance requirements. For instance, systems with QoS or real time requirements.

On the other hand, in a dynamic reconfiguration the transition from one routing function to another is performed while the functional parts of the network are fully operational, i.e., no network down-time and not halting packet injection. This typically leads, when compared with static reconfiguration, to a reduction in the number of packets that miss their QoS deadline. The problem in this approach resides in the fact that, in general, two different and individually deadlock-free routing functions may be prone to deadlocks if they coexist in the network. This means that, in a dynamic reconfiguration, there will be a transition phase between the old and new routing functions where reconfiguration-induced deadlocks may occur. Another drawback when using dynamic reconfiguration is that it usually requires extra resources.

2.1.8.2 Existing Proposals on Dynamic Reconfiguration

Unlike static techniques, dynamic reconfiguration techniques [51] do not require completely stopping the traffic in the network. However, some packets must be removed from the network and re-injected later, which could cause a strong degradation in performance during the reconfiguration time. In the last decade several dynamic reconfiguration mechanisms have been proposed. Next we describe some of them.

The Immunet mechanism [129] tries to minimize the impact of the reconfiguration process on the performance of the system, at the expense of providing a specific hardware support, which prevents it from being used on current commercial interconnects.

In [52], a *Partial Progressive Reconfiguration* (PPR) technique is proposed, allowing arbitrary networks to migrate between two instantiations of *up*/down** routing. The effect of load and network size on PPR performance is evaluated in [53]. Another approach is the *NetRec* scheme [120] which requires every switch to maintain information about switches some number of hops away. Yet another approach is the *Double Scheme* [124] (DS), which uses two sets of virtual channels in the network which act as two disjoint virtual network layers during the reconfiguration. The basic idea is first to drain one virtual network layer and reconfigure it while the other is fully up and running, then to drain and reconfigure the other virtual network layer while the first is up and running. A methodology for deriving new reconfiguration processes for any given pair of old and new routing function is given in [106]. An orthogonal approach which may be applicable on top of all of the above techniques is described in [104], where, for *up*/down** routing, only parts of the network (i.e., the "skyline") need to be reconfigured on a network change. Solid theoretical support on which dynamic reconfiguration design methodologies and techniques are proved deadlock-free can be found in [76].

In [48] a reconfiguration mechanism is proposed for InfiniBand. The mechanism computes in a fast manner suboptimal routing tables and distributes them. The reason for this is to minimize the number of dropped packets through the failed link. Later, more balanced paths are computed and distributed. In order to avoid deadlocks (only in the first distribution of paths) some packet transitions are disabled in the network, thus possibly dropping packets (even if packets do not cause deadlocks).

Recently, a new mechanism was proposed in [106], which is referred to as *Simple Reconfiguration* (SR). In SR a token is issued to separate packets routed with the old routing function from packets routed with the new routing function. Tokens advance through an output port in a switch once there are no more old packets passing through the output port (based on input and output dependencies generated from the old routing function). By doing this, there are no cycles in the network since there will be no old packets behind new ones.

2.2 InfiniBand Networks

InfiniBand is a clear example of a successful interconnection network. Although it was created to involve many environments it finally found its niche in data centers and HPC systems, which require high bandwidth and low latency (see Table 2.1). The InfiniBand Trade Association (IBTA) is comprised of leading enterprise IT vendors

including Agilent [17], Dell [19], Hewlett-Packard [15], IBM [21], SilverStorm [28], Intel [23], Mellanox [25], Network Appliance [26], Oracle [5], Sun [22], Topspin [14] and Voltaire [29].

The appearance of IBA (during 2001) was due to the necessity to eliminate the bottleneck that at those days represented the access of processing nodes to I/O devices in the realm of PC clusters. In this sense, its main contribution consists of replacing the bus by point to point connections between processors and I/O devices through switches.

InfiniBand does not solve any problem that has not been solved previously by other network technologies. The novelty lies in the fact that, until then, previous technologies had been directed to proprietary systems (for example the case of the Compaq's Himalaya [12] or the IBM's Parallel Sysplex [121]), which prevented a reduction in costs and their adoption in the market. InfiniBand, however, emerged as an standard, easing its commercialization, at the time that a reduction in cost progressively allowed its adoption to new and wider sectors. In addition, InfiniBand covers in a homogeneous form the access to peripheral devices and the inter-processor communication (IPC), whereas most of the proprietary systems only cover some of these aspects.

Figure 2.14 shows the number of supercomputers in the *top500* list as a function of the interconnection technology. The figure shows that InfiniBand during the last years is becoming popular, reaching 24% in 2007.

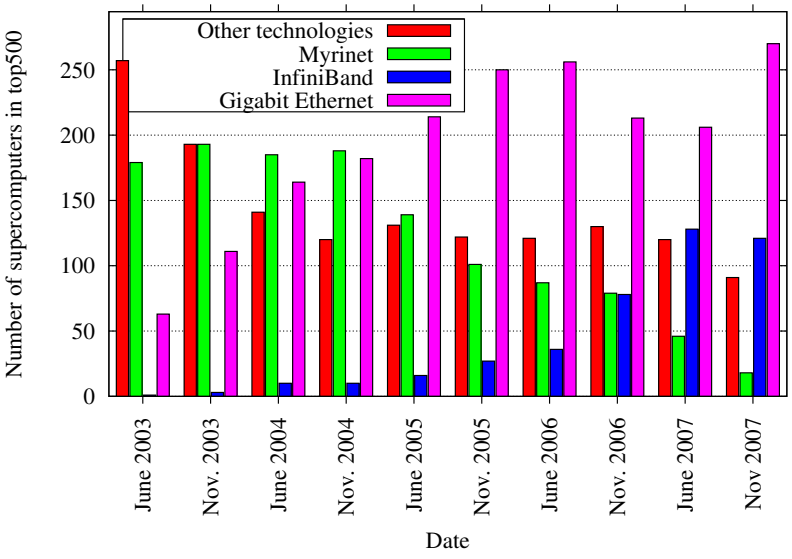


Figure 2.14: Interconnection technologies used by supercomputers on the *top500* list.

Some of the major advantages offered by the InfiniBand network architecture include the following:

- Throughput of 2.5 Gb/s, 10 Gb/s or 30 Gb/s is achievable, depending on the link width implementation, for simple data rate (SDR). Currently, double or quadruple data rate is being explored.
- The processor is not involved in message passing. Rather, each network message transfer is handled by hardware DMA transfer engines within the network interface (named channel adapter).
- InfiniBand protocol includes message transfer commands that permit direct memory-to-memory message transfers between the local memories of two channel adapters.
- The majority of the protocol layers can be implemented in silicon, thereby minimizing the burden placed on software and on the processor.

2.2.1 InfiniBand Architecture

InfiniBand defines a System Area Network (SAN) in which a set of processing nodes and I/O units are connected to the network fabric through point to point links using Channel Adapters (CA). The network is divided into hierarchized subnets. The connection between subnets is done through routers. Each subnet is formed by a set of switches connected to each other by bidirectional point-to-point links forming a connection pattern defined by the user. Thus, the resulting topology may be irregular.

The specification defines a subnet as a set of ports and associated links with a common Subnet ID and managed by a common Subnet Manager (SM). Each subnet exchanges packets solely by traversing switches (not routers). During network configuration, the SM assigns to each of these ports a unique Local ID address (referred to as the LID address) as well as a common Subnet ID (also referred to as the Subnet Prefix) that identifies the subnet that a port resides in. Figure 2.15 shows an example of an IBA subnet. It shows different components as processing nodes, I/O units, links, switches, and a router.

The end node (processing node or I/O device in IBA specs) acts either as the initiator or the ultimate recipient of a packet. The specification defines an end node as any node that contains a CA (either Host Channel Adapter or Target Channel Adapter depending on the type of the end node, processing node or I/O device, respectively), and is allowed to establish connections and generate messages. The processing nodes include memory modules and one or more CPUs, allowing communication between processors and between them and I/O devices. I/O devices can be of different types, from a simple terminal to a RAID disc system.

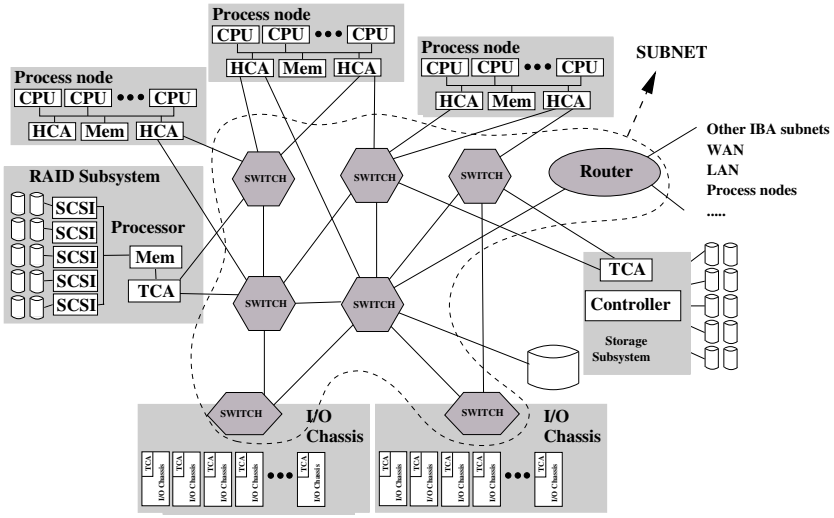


Figure 2.15: Example of an IBA subnet.

The inter-processor communication is handled by the CA, thus releasing the CPU of the load from the I/O communication. This allows multiple communications at the same time without the associated overhead of typical communication protocols.

A node can have multiple CAs, and each CA can have multiple ports. Thus, multiple ports and multiple paths can be used from the source end node. Each CA port has a unique address assigned during the configuration process. It is possible to use this multiplicity of ports and paths to provide fault tolerance, as well as to increase the bandwidth.

Packets are used to send requests or responses from one CA to another. One packet's data payload field can contain a maximum of 4096 bytes of data. A CA sending a message that is larger than a single packet's data payload field[6] must be, therefore, splitted into a series of two or more packets. Each packet contains a data payload field, one or two packet routing headers, and CRCs, among other elements.

Links provide bidirectional and high-speed connection between two ports. At a minimum, it is implemented using one high-speed serial transmission link in each direction capable of transmitting at 2.5 Gb/s, yielding 250 MB/s throughput[7]. Optionally, a link may be implemented with four or twelve transmission lines in each direction, yielding 1 GB/s or 3 GB/s throughput respectively. These values are for a single data rate (SDR) implementation.

[6]Message transfer can be whatever from zero to 2 GB in size.

[7]Note that each 8-bit character is converted to a 10 bit character before transmission (2 Gb/s).

2.2.2 Virtual Channels and Flow Control

Each input port at each switch can implement up to 16 virtual lanes (VL0 -VL15) and 15 service levels. The selection of virtual channels in IBA is based on the use of service levels (SLs). Virtual lanes are devoted mainly for QoS purposes. Nothing prevents them, however, for being used for other purposes like deadlock avoidance and performance improvement. Anyway, its use must be limited for such purposes in order to not compromise the desired QoS levels of the network. The specs define the last virtual lane (e.g., VL15) as a control virtual lane for routing control packets. All the virtual channels except the control channel (VL15) are flow-controlled. It means that the receiver logic for VL15 is permitted to silently drop packets when the receive buffer does not have enough room for the packet. In such a case, control packets are end-to-end flow controlled, thus they are handled by a timeout and retry mechanism.

A particular data VL buffer is, however, only permitted to transmit a packet if the corresponding VL reception buffer on the other end of the link has sufficient buffer space available for storing the entire packet. Each data VL receiver sends credits (a special flow control packet) indicating its available buffer space to its respective transmission buffer on the other end of the link. A credit corresponds to a 64-byte data chunk.

2.2.3 Routing in IBA

The IBA architecture uses distributed routing based on tables (forwarding tables) placed at switches. This routing is deterministic because the forwarding tables keep a single output port for each destination identifier or *DLID*. The destination *DLID* is indicated in the packet header, which cannot be changed. This prevents packets from using alternative paths when the corresponding output port is not available. Therefore, routing in InfiniBand is like source-based routing with the routing information distributed along the path.

The switch's forwarding table is set up by the configuration software (see Section 2.2.4) at startup time. Once selected the output port, the virtual lane to be used is defined by the SLtoVL mapping table, located also on every switch. The new VL to use at the next switch is computed by indexing the table with the input port, the SL identifier placed at the packet header and the output port provided by the forwarding table (see Figure 2.16). The SL label placed at the packet header cannot be changed by switches. However, the packet header contains the VL identifier that is changed on every hop. The InfiniBand specification does not establish how to fill none of the mentioned tables.

Forwarding tables support the *virtual addressing* mechanism, which allows to use more than one identifier to address the same destination node. IBA allows a range of addresses to be assigned to each destination by using a control mask (LMC).

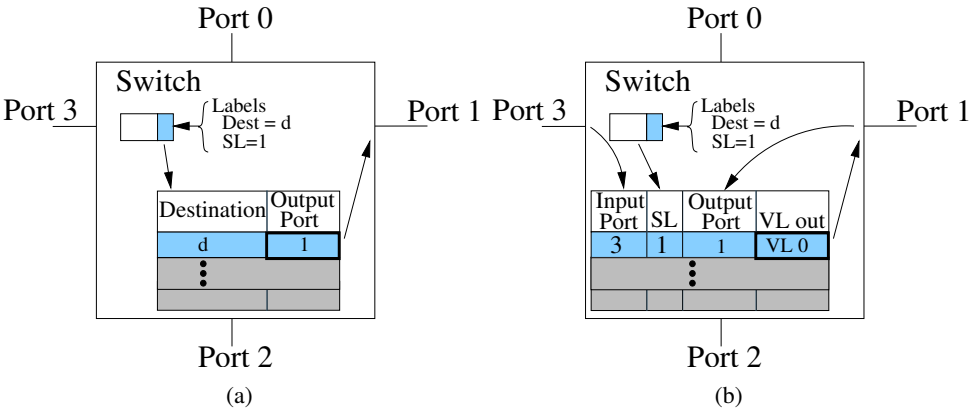


Figure 2.16: Examples of IBA tables (a) forwarding table and (b) SLtoVL table.

The LMC defines the number of less significative bits to be ignored from the DLID when the packet is validated at the destination. In this way each destination port has assigned a consecutive address range with 2^{LMC} values. The value of LMC must be defined large enough for providing the required address range. In particular, 2^{LMC} should be greater or equal to R, where R is the maximum number of routing options between any source-destination pair. Notice that these identifier address bits are not ignored by switches, therefore they consider them as different addresses. In this way, it is possible to provide different routing paths to arrive to the same destination port by using different addresses, or DLIDs.

When a source end node sends a packet to a destination port, the DLID destination label on the packet header is composed with the local port identifier (LID) and the LMC bits. The LID is a different identifier assigned to each device port in the subnet by the SM. If we wish two paths for each destination, then LMC must be set equal to 1. (e.g., if the destination LID is 3 (0011), the address range of DLIDs is 6–7(00110 and 00111)).These two addresses are considered different by all switches, since they will have a dedicated entry in the routing table. Therefore, because both DLIDs are different (though addressing the same physical destination) it is possible to use a different path for each DLID. See Figure 2.17 for an example.

Notice that the maximum number of virtual addresses is limited, mainly because the memory size at switches is limited. Notice also that virtual addressing can be used with any routing algorithm. The maximum value of LMC is 7, thus IBA allows up to 7 bits of the DLID to be used as virtual address (masked at destination). Therefore, up to 128 virtual addresses can be used per destination port.

The fact of fixing a path with a unique SL and the use of several virtual lanes may lead to a mapping conflict [137]. It occurs when two packets labeled with the same

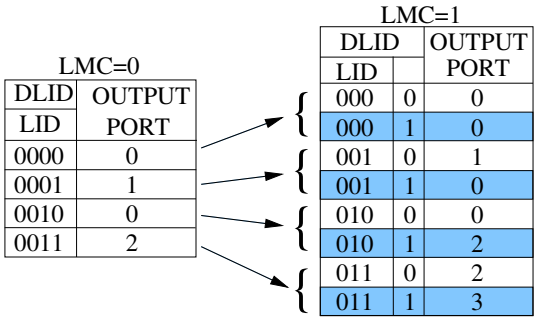


Figure 2.17: Example of routing tables with different LMC values.

SL enter a switch through the same input port, and they need to be routed through the same output port but along different VLs. The problem is that the SLtoVL mapping table does not consider the input VL in order to determine the output VL. Figure 2.18 shows an example of a mapping conflict situation. At switch V a mapping conflict arises as it is not possible to distinguish both paths because they are labeled with the same SL. It has to be noted that this problem arises only when there are paths that use different VLs. For example, path B initially uses VL0 and from switch Q it uses VL1.

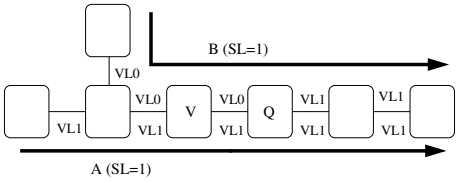


Figure 2.18: Mapping conflict example.

A mapping conflict can be solved only by using different service levels (SLs) for each path causing the mapping conflict. However, this often leads to an excessive number of SLs (notice that the total number is limited to 16). Another solution is to use an alternative path that does not cause a mapping conflict. However, obtaining such alternative path strongly depends on the flexibility provided by the applied routing algorithm, the available network resources (VLs), and the strategy applied to obtain SLtoVL mapping tables. As we will see through this thesis, the mapping conflict issue will influence the applicability of different routing and fault-tolerant mechanisms to IBA networks.

2.2.4 Subnet Manager

In InfiniBand, each subnet is managed in a centralized way by the *Subnet Manager* (SM), which can be located in any device. The SM is the entity that discovers all of the devices in the subnet at startup time to obtain the topology and configures them updating the routing tables. Also, it includes other tasks like the activation of the devices of the subnet, and performs a periodic sweep of the subnet to detect any change in the subnet's topology.

To cope with these tasks, the SM communicates with the *subnet management agents* (SMA), allocated in every network device, through control messages using source routing through the control virtual channel (VL15).

Chapter 3

Fault Tolerance for Interconnection Networks

In this chapter we first introduce some basic concepts on fault tolerance, focusing on PC clusters. Later, we describe the different fault models and fault-tolerant techniques proposed in the literature. Finally, we focus on the mechanisms provided in IBA networks to achieve fault tolerance.

3.1 Basic Concepts

The most relevant terms in fault tolerance are reliability, availability and dependability [126]. In what follows, we provide their basic definitions.

Reliability refers to a system's ability to operate continuously without failure. Reliability, in its simplest form, is defined by the exponential distribution (Lusser's equation), which assumes random failures:

$$R(t) = e^{-\lambda t} \tag{3.1}$$

where λ is the failure rate expressed as the percentage of failures per time (usually expressed as failures per 1,000 hours or as failures per hour). The Mean Time Between Failures ($MTBF$) is derived from equation 3.1, as the average time a system will run between failures. The MTBF is usually expressed in hours and it is defined as:

$$MTBF = \int_0^\infty R(t)dt = \int_0^\infty e^{-\lambda t}dt = \frac{1}{\lambda} \tag{3.2}$$

It can be approximated to:

$$MTBF = \frac{t}{1 - R(t)} \tag{3.3}$$

MTBF can also be defined when considering a system composed of N identical elements:

$$MTBF = \frac{t}{1 - R(t)^N} \tag{3.4}$$

Figure 3.1 shows MTBF curves for different number of nodes and different values of R when applying equation 3.4, which allows us to illustrate the MTBF of some of the most outstanding supercomputers listed in Table 3.1. Values were obtained from [37, 4, 70, 117, 141, 102]. Notice that MTBF is measured in hours, but complex computation tasks on supercomputers often are measured in days. For instance, the complete simulation of the 50 models of the Hafnium Gate Material takes approximately 250 days on the BlueGene/L [34]. Currently, most supercomputers are cluster based. Therefore, in clusters fault tolerance mechanisms are one of the most important aspects.

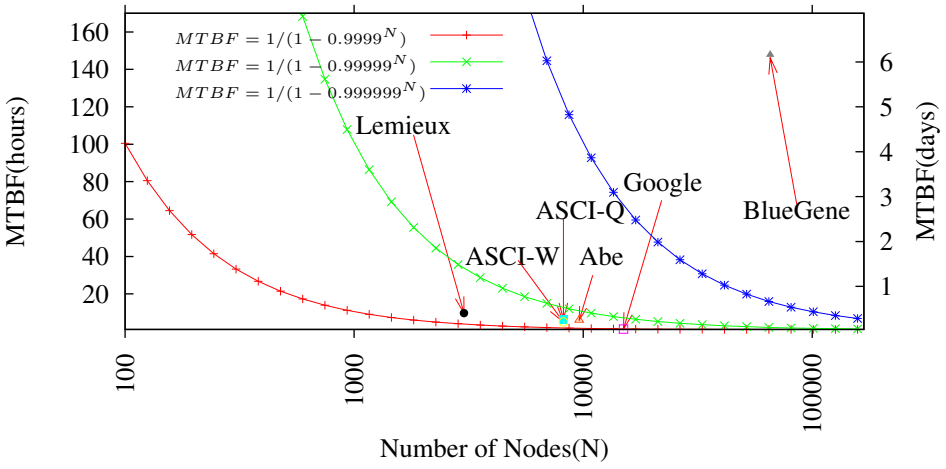


Figure 3.1: MTBF values of some supercomputers and MTBF curves obtained for different values of reliability.

Machine	# CPU	Medium Time Between Failures
ASCI Q (LANL)	8,192	MTBI* 6.5 hr. 114 unplanned outages/month. HW outage sources: storage, CPU, memory
ASCI White (LLNL)	8,192	MTBF 5 hr (2001) and 40 hr (2003) HW outage sources: storage, CPU, 3^{rd} party hardware
Seaborg (NERSC)	6,656	MTBI 14 days. MTTR 3.3 hr Availability 98.74%. SW is main outage source
Lemieux (PSC)	3,016	MTBI 9.7 hr Availability 98.33%
Google	$\sim 15,000$	20 reboots/day. 2-3% machines replaced/year. HW outage sources: storage, memory
Abe	9,600 cores	MTBF 6 hours
BlueGene	65,535	MTBF time of the largest system installation is about 6.16 days (dominated by memory failure).

*MTBI: Medium Time Between Interruptions.

Table 3.1: MTBF values on Large-Scale HPC Systems.

Availability $A(t)$ is defined as the probability that a system operates correctly and is available at time t. Availability differs from reliability in that reliability involves an interval of time while availability involves an instant of time. A system can be highly available despite experiencing frequent periods of inoperability as long as the length of each period is extremely short. In other words, the availability of a system depends not only on how frequently it becomes inoperable but also, how quickly it can be repaired.

In particular, in case of a network, availability is the average fraction of total connection time that it is expected to be running. Availability can be computed as a function of the Mean Time Between Failures (MTBF) and the Mean Time To Repair (MTTR), which includes times for fault detection, dispatch, diagnosis/isolation, repair and actual restore of service.

$$Availability = MTBF/(MTBF + MTTR) \tag{3.5}$$

Dependability is used to encapsulate the concepts of reliability and availability. Dependability is the quality of service provided by a particular system [99]. Reliability, availability, safety, maintainability, performability, and stability are examples of measures used to quantify the dependability of a system.

We will use the terms **fault tolerance degree** and number of **singular cases** for characterizing the fault tolerant capabilities of the proposed mechanisms, and for comparison purposes between different fault tolerant techniques. **Fault tolerance degree** corresponds to the maximum number of faulty components in the network that can be tolerated simultaneously by the routing algorithm (the network is still

working providing the same connectivity, that is, there is at least a path for every source-destination pair).

We define a **singular case** as the fault combination that can not be tolerated by the routing algorithm, thus, the network is logically disconnected. In other words, those fault combinations for which the routing algorithm is not able to obtain a valid path for at least one source-destination pair. Notice, though, that the routing algorithm could continue to provide healthy paths for the rest of source-destination pairs. However, fault combinations that lead to physically unconnected end nodes as a consequence of the fact that all their link become faulty are not considered as singular cases.

Notice that when the number of faults is lower than the fault tolerance degree the number of singular cases is zero. Notice also that for each number of faults the total number of combinations is different. Therefore, when evaluating the number of singular cases, we will represent them as the percentage of not tolerated fault combinations from the total number of fault combinations.

3.2 Fault Models

Faults can be classified as permanent and transient depending on its duration along the time. A transient fault disappears after a short interval of time. For example, it can produce a bit-flip which corrupts the header or the payload of a packet. For such type of faults, error control can be implemented at either link level or end-to-end level [67]. Routers at the ends of a link work together to deal with transient faults. Each router stores and checks the incoming flit before forwarding it to the next router. Alternatively, error control can be implemented at the end-to-end level, that is, at the end nodes themselves. Transient faults are usually modelled with a Bit-Error Rate (BER).

On the other hand, permanent faults do not disappear over the time. For instance, permanent faults can be related to permanent damages of circuits or wires. Permanent faults are usually described in terms of MTBF, which is usually expressed in hours.

In order to deal with permanent faults in a system, two fault models can be considered. The first one is the static model. Once a fault is detected, all the processes running in the system are stopped, the network is emptied, and a process to recover from the fault is launched. This process computes the new routing tables so that the use of the failed component is avoided. The network is resumed once all the new routing tables are updated[1]. The practical use of the static fault model requires checkpointing techniques. Checkpointing allows processes to resume their work from a previous known state when the last checkpoint was performed.

[1]Commonly, routing tables are computed at a single end node and later are distributed to the rest of end nodes and switches.

The second model is the dynamic fault model. In this model, applications do not stop and the network is not emptied. Once a fault is detected, actions are taken in order to appropriately handle the faulty component without stopping the network traffic. For instance, a source node that detects a faulty component through a path may change to an alternative path that does not use the failed component. In this way, the system is kept working, without draining the network traffic, and without doing any checkpoint to deal with the occurrence of faults.

In this thesis we focus on permanent faults and on both faults models (dynamic and static).

3.3 Fault-Tolerance Mechanisms

The use of fault tolerance mechanisms will assure, in the case of a component failure, that the system keeps working, although in a degraded mode, until the failed component is repaired. Basically, there are three ways to cope with faults in the interconnection network: component redundancy, fault tolerant routing algorithms, and reconfiguration techniques.

Using component redundancy has been the easiest and costly way to provide fault tolerance. Components in the system are replicated and once a failed component is detected, it is simply replaced by its redundant copy. An example of using component redundancy can be found in the Tandem's Himalaya Servers [12].

Fault tolerant routing algorithms aim at avoiding messages traverse faulty components by providing some kind of routing path redundancy. To this end, messages must be able to be routed trough alternative paths, so circumventing or avoiding faulty regions over the network. Fault tolerant routing schemes should be designed to tolerate a certain number of faults, while still guaranteeing deadlock freedom in the network. However, to fulfil these requirements, fault tolerant routing strategies often need to use additional network resources, as virtual channels or additional hardware at switches or routers.

Applying reconfiguration [51] any number of faults can be tolerated, provided the network is physically connected. The reconfiguration consists, once a fault is detected, of discovering the new topology, computing a new routing scheme and updating the required components in the network. The main disadvantage of reconfiguration is the high delay packets may suffer during the reconfiguration process. Reconfiguration techniques often require additional network resources.

Each mechanism or fault tolerant approach has different requirements on resources, and therefore different costs. Those fault tolerant approaches with hardware requirements usually are costly, more expensive than software-based mechanisms. When the failure probability is high, or the system has strong fault tolerant requirements, like life support systems, usually it is preferable to use hardware-based solu-

tions that are generally more expensive. On the contrary, for less critical applications, or when the probability of faults is low, it is preferable to use cheap mechanisms based on software [143].

3.3.1 Fault-Tolerant Routing Algorithms

A large number of fault-tolerant routing algorithms for multiprocessor systems have been proposed, especially for systems with mesh and torus topologies. Most of the techniques have been proposed for wormhole switching. We can classify the fault tolerant routing algorithms with the taxonomy provided in Figure 3.2. This taxonomy considers the support type (hardware/software), the need for disabling healthy nodes, the tolerated fault type and how packets advance through the network.

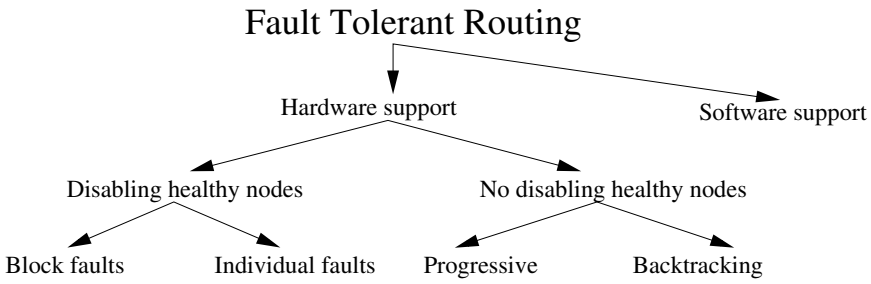


Figure 3.2: Taxonomy of fault tolerant routing algorithms.

Next we hint the most relevant fault tolerant routing algorithms. First, we focus on techniques which require hardware support. These techniques are further divided into two groups depending on whether they disable healthy nodes or not. Later, we describe techniques based on software support.

3.3.1.1 Hardware-Based Fault Tolerant Routing Algorithms

Healthy nodes can be disabled to provide fast and simple solutions for tolerating faults in the network. Some of these approaches are based on block faults [50, 55, 56, 58], whereas others allow individual faults [61, 72, 83].

Techniques based on block faults consider that an entire network area (region or block) has failed. These techniques can be further classified depending on the shape of the region: convex or rectangular region and non-convex region. To form the region some healthy nodes placed in the vicinity of the faults must be marked as failed. Then, messages are routed around faulty region in a manner that avoids introducing cyclic channel dependencies in the CDG. However, as several useful end nodes are discarded, the global capacity of the system is reduced unnecessarily.

Chien and Kim [58] proposed a planar-adaptive routing algorithm for 2D meshes. This approach adds one virtual channel in the vertical dimension, so defining two virtual networks. Each virtual network provides movement in a unique direction through the horizontal dimension. Misrouting is allowed only in one virtual network, thus deadlock is avoided. This routing algorithm can be extended to multidimensional meshes and tori, and requires three and six virtual channels in meshes and tori, respectively. Routing is adaptive into each plane (formed by two consecutive dimensions), but planes must be visited in order. Fault regions in these topologies are constrained to be block faults, so that any (two-dimensional) cross section has a rectangular form.

An approach to reduce the number of healthy nodes marked as faulty was introduced by Chalasani and Boppana [54], which is based on the concept of fault rings to support more flexible routing around fault regions. A fault ring is the sequence of links and nodes that are adjacent to and surround a fault region. If a fault region includes boundary nodes, the fault ring reduces to a fault chain. Figure 3.3.a shows an example of fault rings and a fault chain. This method uses four virtual channels under the assumption that fault rings do not overlap. When applying this technique to tori four virtual channels are required, although routing restrictions are required also on the wraparound links. This was improved by Chen and Chiu [56] for meshes by reducing the number of virtual channels down to three and still allowing fault rings to overlap.

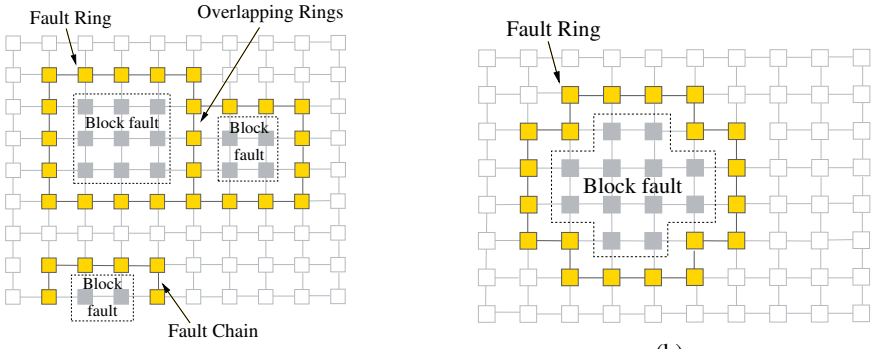


Figure 3.3: Example of block faults: (a) rectangular block faults, and (b) a solid block fault (nonconvex).

A technique for non rectangular regions was proposed in [55] to reduce the under utilization of resources. It is based on extending the concept of fault rings to certain classes of nonconvex fault regions, referred to as a solid fault model. Figure 3.3.b shows an example. As in previous techniques, messages can be routed along the fault ring. This technique requires a set of four virtual channels for implementing different virtual networks in 2D mesh.

Origin-based fault-tolerant routing is a paradigm that enables fault-tolerant routing, but without the addition of virtual channels [87]. Each message progresses through two phases. In the first phase, the message is adaptively routed toward a special node (labeled as $origin$). On reaching this node, the message is adaptively routed to the destination in the second phase [98]. This phase ordering prevents the formation of cycles in the channel dependency graph (CDG).

The fault-tolerant implementation of *origin-based* routing requires the fault regions to be square. The regions are formed by starting from the rectangular fault regions created by preceding techniques and disabling additional nodes to ensure that the fault regions are square. Finally an origin node is selected such that the row and column containing it has no faulty nodes. If this is not possible, this technique can be applied to the largest fault-free submesh, disabling all remaining nodes. In such a case a larger number of nodes are disabled.

Previous methodologies provide fault tolerance by disabling blocks of nodes. However, a method based on disabling individual healthy nodes instead of blocks of nodes was proposed in [88]. This method reduces the number of healthy nodes disabled, although it still disables some. It is based on declaring some non faulty nodes to be "lambs". Each lamb node is used for routing but not for processing, so a lamb is neither the source nor the destination of a message. The lambs are chosen so that every "survivor node", a node that is neither faulty nor a lamb, can reach every survivor node by at most two rounds of a deterministic routing algorithm (such as e-cube). In order to guarantee deadlock freedom, a different virtual network is used in each round. Thus, when performing two rounds a virtual channel transition is required.

Contrary to the previous techniques, other methods do not require to disable any healthy node [68, 82]. They rely on either applying adaptive routing (making use of a certain number of virtual channels), or using a different switching technique (pipelined circuit switching [68] or scouting [69]).

These mechanisms can be classified into two groups, depending on the way packets advance in the network. The first group is referred to as progressive as the mechanism continuously routes packets towards their destination. The second group is referred to as backtracking as they allow packets to be backtracked when they can not advance toward their destination.

A. Progressive

A technique based on the turn model is presented in [84], which tolerates up to $n - 1$ faults in n-dimensional meshes. The advantage of this approach is that no virtual channels are required. However it only tolerates one fault in 2D meshes.

The technique proposed in [72, 73] combines the theory of deadlock avoidance based on escape paths [71] with the turn model, for n-dimensional meshes

using wormhole switching. This technique uses four virtual networks, and each one has a virtual channel associated.

In [85] a fault tolerant routing technique based on routing adaptively for some source-destination pairs to an intermediate node, and routing also adaptively from it to the destination is proposed. The methodology assumes a static fault model and the use of a checkpoint/restart mechanism. However, there are scenarios where the faults cannot be avoided solely by using one intermediate node. Thus, an extension of the methodology consists of disabling adaptive routing, and/or using misrouting on a per-packet basis, and/or the use of more than one intermediate node for some paths. This technique requires three virtual channel for meshes and tori and tolerates up to five faults. An additional virtual channel (total of four) is required for obtaining deadlock free minimal routing (in meshes and tori).

The *dimension reversal* (DR) approach defined by Dally and Aoki [63] produces gracefully degradable network performance in the presence of faults by dividing messages into classes. Each message is permitted to be routed in any direction. Misrouting is also permitted, although it should be controlled. When a dimension reversal takes place, the message moves into the next virtual network. The total dimension reversals is limited to the number of virtual channels used for this technique (r). Thus, messages in the last virtual network ($r - 1$) are routed in dimension order (non adaptive). This approach may require a large number of virtual channels.

In [128] a low cost fault tolerant routing algorithm for VCT switching valid for any topology was proposed. It uses adaptive routing in a virtual network, and uses another one with non-minimal routing as escape channel when a fault is encountered[2]. The non-minimal routing is obtained using a Hamiltonian path. This technique requires additional hardware at the switches, for supporting the escape routing.

B. Backtracking

These fault-tolerant routing strategies were proposed for pipeline circuit switching (PCS) and/or scouting switching techniques[3].

By permitting backtracking and a certain number of misroutes a larger number of faults can be tolerated. Despite tolerating any number of faults, these strategies often strongly penalize network performance. One of the families of fault tolerant routing algorithms based on these strategies is referred to as

[2]Bubble is also assumed in the escape paths.

[3]In PCS the path is reserved by the packet header before sending the data flits, whereas in scouting the data flits follow the packet header by a certain distance. Therefore, in both cases, like wormhole switching, the header can go backtrack to the previous node searching for a path to the destination.

MB-m algorithms, where m represents the maximum number of misroutings. This family of algorithms are proposed for pipeline circuit switching.

Channels are labeled as safe or unsafe [101, 147] depending on the number of faulty links/nodes within the immediate neighborhood. The routing algorithm to obtain the path is based on two phases depending on the label of the channel. Such protocols are referred to as *Two-Phase* protocols (TP). The first one uses adaptive deadlock-free routing function based on *Duato's protocol* (DP) [71]. The second one (conservative phase) uses a form of MB-m. The switching technique used is the scouting switching [75], which has a fixed scouting distance. Each of these phases use different channels for guaranteeing deadlock freedom.

These techniques require a higher router complexity than the previously described fault tolerant routing techniques.

3.3.1.2 Software-Based Fault Tolerant Routing Algorithms

The addition of virtual channels and the enforcement of routing restrictions between them can impact the design and implementation of the switches. However, in environments where the fault rates are relatively low, the use of expensive, custom, fault-tolerant switches often cannot be justified. Moreover, contemporary routers are compact, oblivious, and fast. To overcome the drawbacks of the hardware-based solutions, a software-based fault-tolerant routing approach [143] can be used. This technique, referred to as *e-sft*, provides a solution for tolerating failures with concave fault regions, which are not tolerated by some of the previous techniques, based on the removal of messages from the network or absorbed by the local router when they encounter a faulty component, and later reinjecting them at an intermediate node. However, some packets may suffer high latencies due to the packet ejection and reinjection. Notice that memory allocation must have sufficient buffer to store the messages absorbed to avoid deadlocks. The important characteristic of this approach is that messages are still routed in dimension order between any pair of intermediate nodes.

The development of these techniques is governed by the relationship between the MTBF and the MTTR. When MTTR is much lower than MTBF, the number of existing faulty components in a repair interval is small. In fact, the probability of the second or the third fault occurring before the first fault is repaired is very low. In such environments, software-based rerouting can be a cost-effective and viable alternative.

The software-based approach is based on the observation that the majority of messages do not encounter faults and should be minimally impacted, while the relatively few messages that do encounter faults may experience substantially increased latency, although the network throughput may not be significantly affected.

3.4 Applying Fault Tolerant Mechanisms to PC Clusters

As previously described there are three ways to cope with faults in interconnection networks. The first one consists of component redundancy, which is the easiest but the most costly option. The second one consists of applying fault tolerant routing strategies. However, most of the fault-tolerant routing strategies proposed in the literature and that we have described above are not suitable for clusters. This is because they often require certain hardware support that is not provided by current commercial interconnects [49, 89]. Other strategies rely on the use of adaptive routing. However, they cannot be applied, as routing in clusters is usually deterministic. Additionally, some of these routing strategies need to perform dynamic virtual channel transitions. However, virtual channels either are not supported (e.g., Myrinet) or they cannot be dynamically selected at routing time (e.g., InfiniBand).

The last fault tolerant techniques consist of applying network reconfiguration. Recent proposals try to diminish the impact of the reconfiguration process in the system, the expense of requiring additional hardware support, which might prevent its use in current commercial networks.

Recently, however one interesting mechanism, referred to as FRoots, has been presented in [144]. This mechanism is suitable to IBA networks as it uses virtual channels (VLs) in a static manner. FRoots requires as many virtual channels as required to guarantee that every switch is a leave in an up*/down* tree. On the occurrence of a failure connectivity is ensured by routing packets through the associated up*/down* tree where the failed switch was a leave. Each up*/down* tree is mapped onto a different virtual channel. Therefore, as can be detected, the resource requirements to implement FRoots are high. (5 VLs for a 16-switch torus network).

Next, we describe the key aspects for fault tolerance in IBA, as the interconnection network commonly used in current PC clusters, and later we outline the fault tolerance mechanisms we have proposed in this thesis for PC clusters (focused in IBA networks).

3.4.1 Fault Tolerance in IBA

As commented in section 2.2, an IBA network is divided into subnets The Subnet Manager (SM) is the entity that discovers all the devices on an IBA subnet, configures them, and detects any change in the subnet's topology. The SM is allocated in a particular node in the subnet. A change in the topology can be due to devices being added or removed, or because of a fault. In each network device there exists a *Subnet Manager Agent (SMA)*, which is responsible for monitoring port's link integrity.

The IBA standard defines two complementary mechanisms for detecting changes. On the one hand, the SM performs periodic sweeps of the subnet requesting information to each SMA associated to each component. The frequency of these sweeps

is not defined by the IBA standard, thus it can be adjusted accordingly to parameters like the size of the subnet or the desired detection time for changes. On the other hand, each SMA could actively notify to the SM by a control packet whenever a change is detected. This method is described in the IBA specs as optional, and leaves to the vendors the decision to implement it or not.

In a congested scenario, control and sweep packets may be discarded. This is because there is not flow control in the virtual channel reserved (VL15) for control and sweep packets. Thus, if the SM does not receive an answer before a timeout (this time will be defined as a function of subnet's size), it will consider the packet has been discarded and it will resend it again.

A possible way to provide tolerance to faults in IBA would be by using several alternative paths between each source-destination pair of end nodes, selecting one of them in the source node accordingly to the fault pattern. This is possible because, in IBA, the routing and the selection of virtual channels are made based on the local identifier of destination (*DLID*) and the service level (SL) of the packet header. These two fields are calculated in the source node and they do not change along the path. Hence, the routing path and the traversed virtual channels are completely determined by the DLID and SL values established at the source node. Modifying at its source node the DLID of the packet it is possible to rout it with a different path.

IBA provides a mechanism supported by hardware, denominated Automatic Path Migration mechanism (APM) [89], which can be used to select one among the available alternative paths. According to this mechanism, previously to the detection of the fault, the subnet manager (SM) has loaded the information of the alternative path (the DLID on the SMA and the entries in the routing tables on switches) and has armed the APM mechanism for each of the paths in all the end nodes. APM provides a fast mechanism for the migration of the initial path to the alternative one when a fault is detected. Once the path migration is completed, the alternative path becomes the new default path. Therefore, the SM is able to load new alternative paths and to setup the APM mechanism.

3.4.2 Contributions of the Thesis

In this chapter, we have described fault tolerance as one of the key issues in clusters, we also showed the most important fault tolerance mechanisms proposed in the literature, concluding they are not suitable for PC clusters.

Thus, in this thesis we focus on providing adequate levels of fault tolerance to PC clusters, by proposing several mechanisms suitable for interconnection networks commonly used in clusters. In this section we briefly outline the proposals that we have developed, which will be described in more detail in the following chapters.

In the thesis, we cover the design a wide set of approaches to cope with faults in interconnection networks for PC clusters. Some of these approaches are focused

on fault tolerant routing mechanisms. They are described in Chapters 4 and 5. Concretely the proposals presented in Chapter 4 are based on providing a certain number of disjoint paths between every pair of end nodes. Paths should be disjoint in order to guarantee that the same fault can not be found along more than one path. Thus, to tolerate n faults it is necessary to provide $n+1$ disjoint paths. These mechanisms are tailored for InfiniBand networks. Figure 3.4 shows an example of four disjoint paths from switch S to switch D.

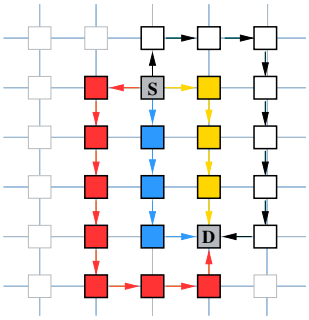


Figure 3.4: Example of disjoint paths.

The main goal of the mechanisms, after a fault is detected, is to choose a different non-faulty path for each source-destination pair whose default path is affected by the fault. As we will see, the mechanisms will focus on different solutions to provide good sets of disjoint paths in a time efficient manner and requiring the lowest number of IBA resources (VLs and SLs).

In Chapter 5 we describe a fault tolerant technique based on the concept of reachability. This mechanism will provide in a fast manner, once a fault is detected, all the needed alternative paths to tolerate the fault. For example, Figure 3.5 shows a network with a faulty link, thus the path from switch A to switch B is faulty. However, the paths from switch A to C and from C to E are fault-free. Therefore, a new path can be composed by joining these two paths. The provided alternative paths will depend on the resources available for this technique.

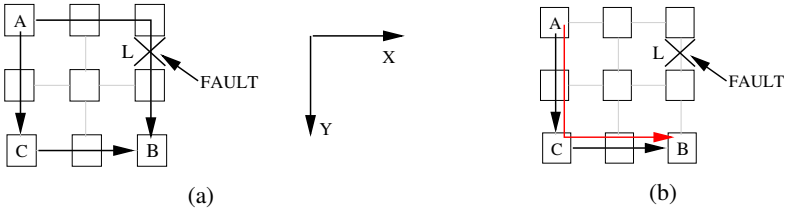


Figure 3.5: Example of composing an alternative path from two already existing routing paths in the presence of faults.

Finally, in Chapter 6 the last fault tolerant proposal is presented The new proposal is based on reconfiguration techniques. Notice that when using reconfiguration, any number of faults can be tolerated as long as the network remains connected. The proposed mechanism will provide a better reconfiguration mechanism than existing techniques when considering the global reconfiguration time, throughput, and latency. Additionally, the mechanism will be tailored for interconnection networks commonly used in clusters, where simplicity and resource requirements saving are a prime issue.

Chapter 4

Fault-Tolerant Routing Methodologies Based on Disjoint Paths

In this chapter we present the methodologies based on the use of disjoint paths to provide fault tolerance. The aim of the proposed solutions is, when a fail occurs, to migrate each affected path to an alternative disjoint path which is non-faulty. Two mechanisms are presented. First, the TFTR methodology uses virtual channels in order to achieve disjoint paths for any given pair of end nodes. Later, the SPFTR method is presented, which relies on the use of path patterns to provide scalability and speed up the process of computing disjoint paths. For each method, implications on InfiniBand are discussed. Later, in Chapter 8 both methodologies are fully evaluated and compared.

4.1 Preliminaries

Before describing the proposed mechanisms we detail in this section the basic foundations. The aim is to provide the maximum fault tolerance degree at the lowest cost from the network point of view. As described in previous chapters, InfiniBand provides a simple mechanism to tolerate faults in the network. The mechanism is referred to as APM (Automatic Path Migration) (see section 3.4.1) and is intended to allow a fast transition from a failed path to a new (non-faulty) path. APM, however, does not deal with the computation of the paths. Thus, the methods proposed in this chapter complement the APM mechanism.

Figure 4.1.a shows an example where two paths are available for a given pair of end nodes[1]. At the end node connected to switch S APM is configured with both paths, and one of them is used by default. For instance, path #1 is used by default and path #2 is reserved for backup purposes. Notice that APM offers, at maximum, two paths for each possible destination at each source end node.

Let us suppose that at a given point in time link labeled A fails. In such situation, path #1 becomes unavailable and the APM mechanism is triggered. The result is that the backup path becomes the default one and the faulty path becomes the backup one. Thus, both paths are swapped.

We can consider also a different fault scenario, when the faulty component is link B or switch X. In either case, notice that the fault affects both paths. Thus, the APM mechanism can not migrate to a non-faulty path.

Therefore, when computing alternative paths for the APM mechanism is important to take into account the components (links or switches) shared by the alternative paths. Notice also that in the presence of a switch failure two paths may fail even if they do not share any link. An example is shown in Figure 4.1.b in which both paths share switch C. Thus, the number of shared switches must be also considered when computing alternative paths.

A possible set of alternative paths for APM can be viewed in Figure 4.1.c. In this case, paths do not share neither any link nor any switch. Thus, in the presence of a link or switch failure only one path may be affected. However, notice that in this case some components are still shared by both paths. The NIC at both end nodes are shared by both paths. Hence, in order to provide truly disjoint paths it would be necessary that end nodes have more than one NIC each one attached to a different switch. Figure 4.1.d shows an example with end nodes having two NICs connected to different switches. In this thesis, however, we focus on fault tolerance at the network level, and not at the system level. So, we assume that end nodes have only one NIC attached to a unique switch. We consider, thus, that if the link connected to the NIC fails, then the end node is disconnected.

We define a group of paths from the same source-destination pair as *disjoint paths* if they do not share any network resource among them, not considering the source switch, the destination switch and the links connecting both end nodes to the network. Similarly, we define a group of paths as being *partially disjoint paths*, when they share some resources (but not all the resources). In this case, we use the term *degree of disjoint paths* as the number of network resources (switches or links) shared by the paths. Notice that we do not consider percentages. For example, in Figure 4.2 two different groups of paths are plot. In Figure 4.2.a paths #1 and #2 share one link and

[1]When we describe the paths, we will refer to the first switch of each path as *source switch* S (the source end node is connected to switch S), and the last switch as *destination switch* D (the destination end node is connected to switch D).

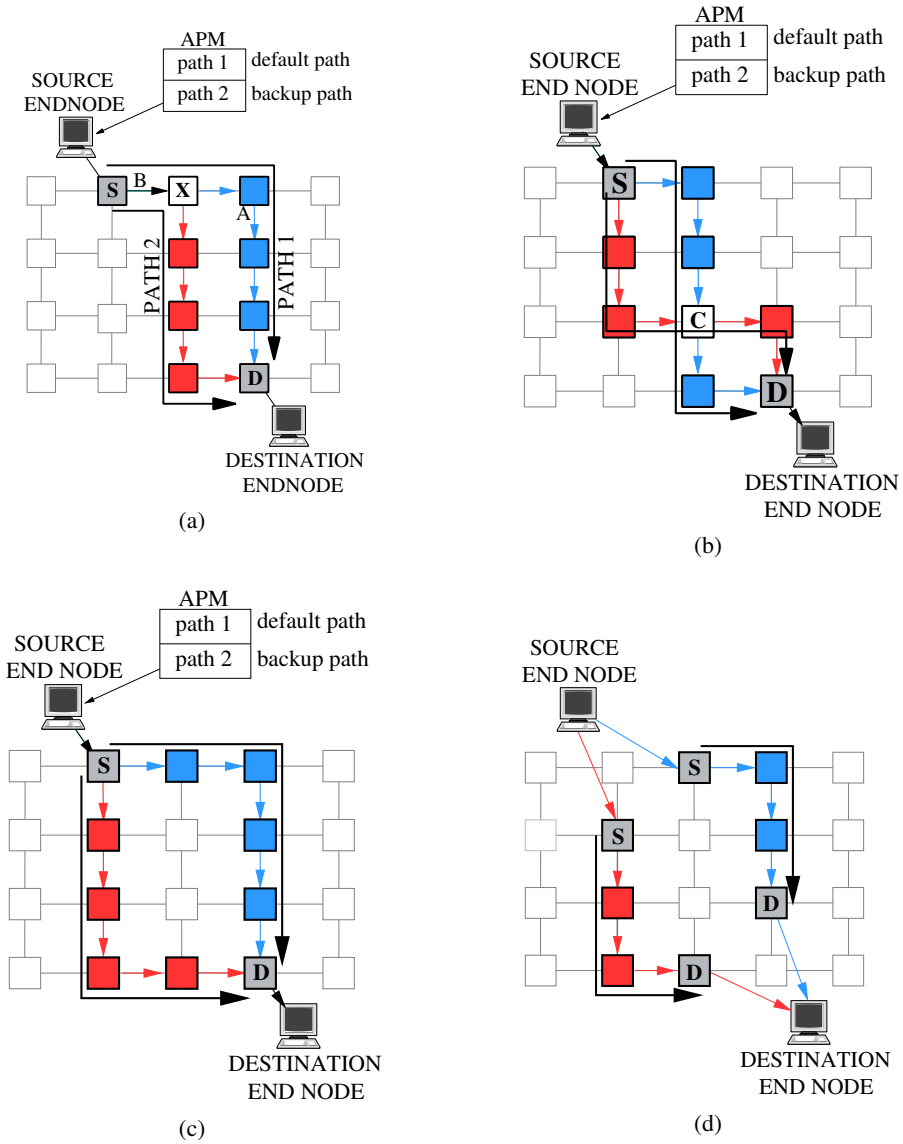
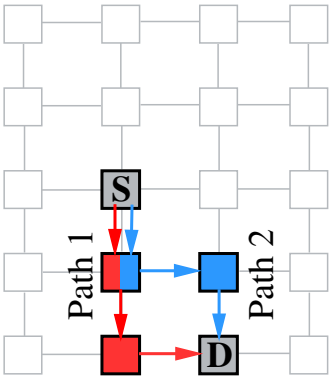


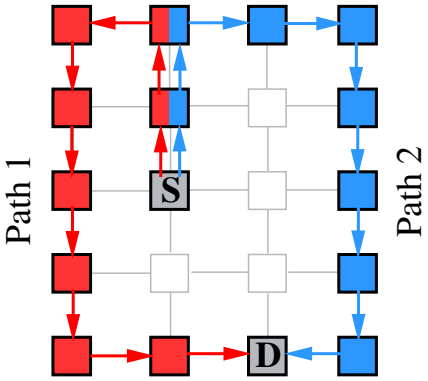
Figure 4.1: Examples of disjoint paths: (a) Sharing one link, (b) Sharing one intermediate switch, (c) Disjoint paths, (d) Multiple NICs at end nodes.

one switch, and in Figure 4.2.b paths #1 and #2 share two links and two switches. Notice that paths defined in Figure 4.2.b have a smaller percentage of shared links or switches, but there are twice more shared components between the paths. Thus, the probability of one fault affecting both paths is higher (although the shared percentage



Path 1: uses 4 switches and 3 links.
Path 2: uses 4 switches and 3 links.
Shared: 1 link and 1 switch.
Shared percentage: 30.0% of links and 25.0% of switches.

(a)



Path 1: uses 10 switches and 9 links.
Path 2: uses 10 switches and 9 links.
Shared: 2 links and 2 switches.
Shared percentage: 22.2% of links and 20.0% of switches.

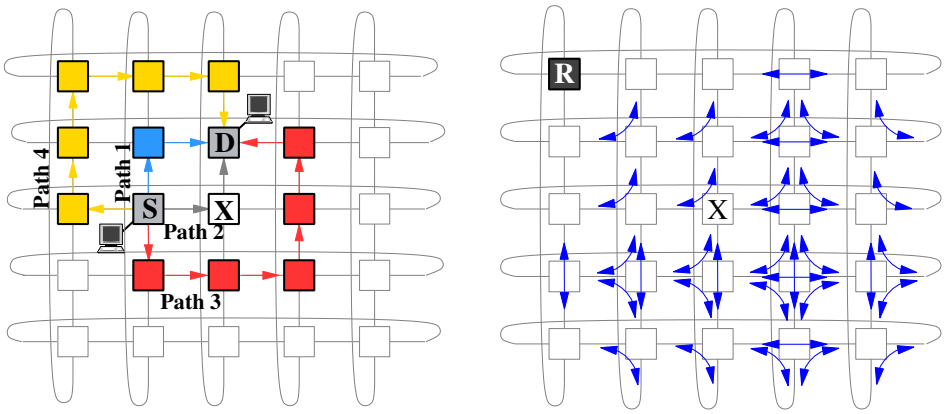
(b)

Figure 4.2: Examples of the degree of disjoint paths: (a) sharing one link and one switch, and (b) sharing two links and two switches.

is lower). Therefore, percentages are not representative to qualify the best group of paths in fault tolerance terms.

Although it is easy to compute disjoint paths for any source-destination pair, many combinations are not allowed. The reason is that they either may induce deadlock situations or they may require too many network resources (to guarantee deadlock freedom). Let us focus on the deadlock freedom condition. Figure 4.3.b shows the routing restrictions enforced by an *up*/down** tree in a 5×5 torus network. A routing restriction indicates an illegal turn applied by the routing algorithm. Each restriction on a switch prevents traffic to be routed between a pair of input and output links. These restrictions are represented by arrows in the figure. Although the figure shows the restrictions for an *up*/down** tree, any routing algorithm imposes routing restrictions in order to guarantee deadlock freedom, and this can be represented in the same way (an exception is when using the bubble flow control mechanism).

When looking for four disjoint paths between each pair of end nodes in the torus network, it is not possible to find all the paths without crossing any routing restriction. For example, Figure 4.3.a shows four disjoint paths for a given source-destination pair. Notice that paths #1 and #2 are computed taking into account that they must be disjoint and minimal. Any combination of four disjoint paths (minimal) ends up with these two paths being included. In this situation path #2 requires crossing a forbidden transition. See Figure 4.3.b.



(a) (b)

Figure 4.3: Example of disjoint paths. (a) Four disjoint paths in a 5×5 torus and (b) the set of routing restrictions enforced by *up*/down** on the same 5×5 torus (R means root switch).

As we have seen, routing restrictions reduce the possible routing paths that can be computed, thus it becomes a challenge to obtain *disjoint paths* for all source-destination pairs in the network.

The main objective of the mechanisms presented in this chapter is to provide fault tolerance based on disjoint paths. As a first goal, we provide the maximum number of disjoint paths for each pair of end nodes. Alternatively, we provide *partially disjoint paths* with the highest *degree of disjoint paths* for those cases when the number of resources available is bounded. These sets of paths are provided to the APM mechanism in InfiniBand.

The second goal in this chapter is to obtain these paths with a low computation time. Thus, a mechanism is presented to speed up the computation process in torus topologies for IBA networks.

4.2 TFTR Methodology

In this Section we present the first methodology referred to as $Transition\text{-}Based\ Fault\ Tolerant\ Routing$ (TFTR). TFTR provides a set of alternative disjoint paths between each pair of end nodes. To this end, the $TFTR$ methodology uses the APM mechanism to tolerate faults. To provide reliability, while still guaranteeing deadlock freedom, some virtual channels (VLs) and service levels (SLs) are used in InfiniBand. This is done in order to increase the flexibility of the computed paths. It is important, however, to consider the trade-off between the achieved fault tolerance degree and the number of network resources used.

Notice also that the methodology presented to compute alternative paths is not affected by the hardware used for detecting or notifying faults. Indeed, TFTR computes paths and dictates in which order they must be used (shortest paths being used first).

4.2.1 TFTR Stages

TFTR is divided into six different stages as shown in Figure 4.4. The key point to achieve an optimal set of alternative paths (from the fault tolerance point of view) is to have the maximum flexibility when computing those paths. As previously described, the main limitation in the computation of disjoint paths is due to the routing restrictions imposed by the applied routing algorithm. Routing algorithms must provide a set of paths free from deadlocks. This is usually achieved by imposing routing restrictions that limits the flexibility when computing paths. To overcome this limitation, TFTR relies on the use of additional resources to achieve much higher flexibility.

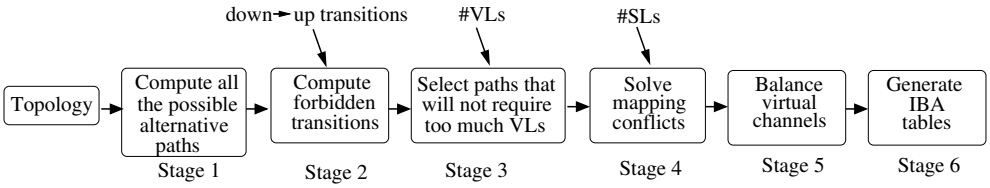


Figure 4.4: TFTR Stages.

Stage 1 Compose the list of virtual addresses. TFTR, at the first stage, computes a certain number of disjoint paths between every pair of end nodes considering only the network topology, and without taking into account the occurrence of possible deadlocks. Therefore, at this stage, the methodology does not limit the use of any possible path, thus obtaining the maximum flexibility and potentially the best set of alternative paths. At a latter stage, deadlock situations will be solved most of the time without modifying the computed paths.

Stage 2 The second stage of the methodology is focused on analyzing deadlocks. For this, the methodology builds and searches for possible cyclic channel dependencies. In particular, an underlying routing algorithm is used in order to detect possible cycles. We have chosen the *up*/down** routing scheme[2] because it can be applied to any topology. We check whether every computed routing path contains any forbidden transition according to the up*/down* rule (a link in the up direction

[2]The assignment of up and $down$ directions to links has been performed according to the DFS methodology [132], instead of the original BFS methodology [138] due to the smaller number of forbidden transitions imposed.

is used after having used one in the $down$ direction), which could lead to a cycle in the channel dependency graph and, therefore, to a deadlock situation. In order to guarantee deadlock freedom, we must remove all those forbidden transitions.

Figure 4.5 shows a detailed example on how the forbidden transitions are obtained. Figure 4.5.a shows a 4×4 torus network. In this example the *up*/down** routing is used, with the root switch labeled R on the top left corner. Figure 4.5.b shows the assignment of up and $down$ directions to channels and Figure 4.5.c shows the up*/down* tree. Finally, Figure 4.5.d shows with arrows the routing restrictions which correspond to transitions from a $down$ channel to an up channel (notice that all routing restrictions are bidirectional).

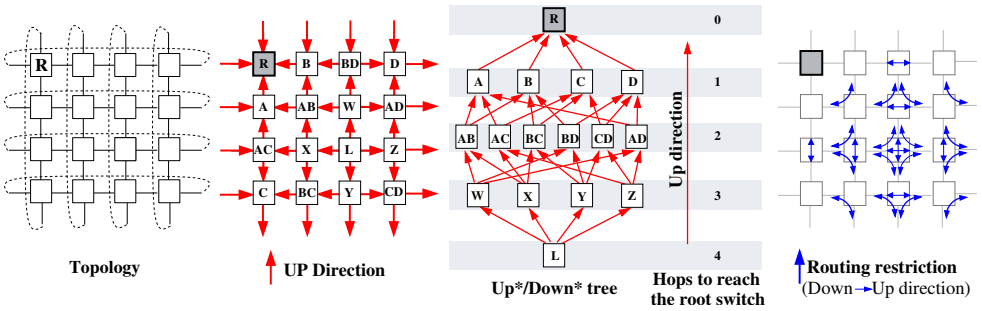


Figure 4.5: Example of up*/down* routing restrictions in a 4×4 torus.

Virtual channels may help when dealing with the deadlock problem. Indeed, the use of virtual channels allows defining virtual networks, each one assigned to a different virtual channel. The traffic can, then, be separated into different virtual networks although physically there is only one network. To flexibilize routing and thus avoid deadlocks we can use virtual channel transitions. In particular, a packet may change from one virtual network to another within the network.

The transition between virtual networks is done at switches when routing the packet. As an example, in InfiniBand each packet is assigned a label referred to as *Service Level*, and each switch has the *SLtoVL Table* in which it defines the virtual channel to be used as a function of the SL and the input and output links used by the packet.

Transitions through virtual channel can be used, thus, to avoid the routing restrictions imposed by the routing algorithm. Figures 4.6.a and 4.6.b show the buffers of the switches involved in a cycle formed by four packets. Notice that packets form a cycle in Figure 4.6.a, but not in Figure 4.6.b. At switch D there is a routing restriction imposed by the routing algorithm. By allowing virtual channel transitions, the routing algorithm is deadlock-free. Indeed, as can be seen in Figure 4.6.b, the packet reaching switch D requests another virtual channel, thus breaking the cycle. Notice

that virtual networks (virtual channels) must be crossed by a packet in a stabilised order to avoid cycles between different virtual networks (virtual channels). Figures 4.6.c and 4.6.d show the same case in a 3D view.

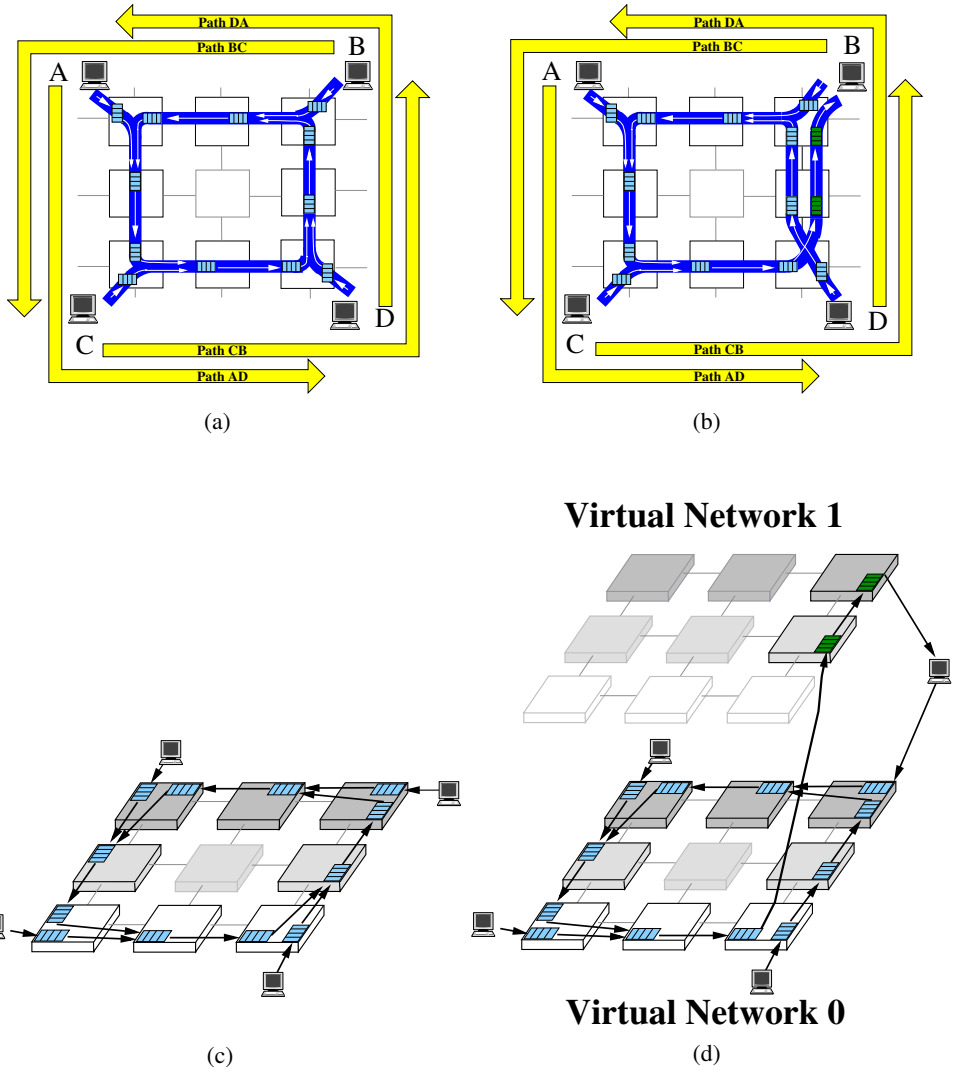


Figure 4.6: Example of a cycle and how it can be avoided when using virtual networks: (a,c) example of a cycle, (b,d) example of virtual network transition.

In our methodology, each packet is injected into the network through the VL0 virtual channel. The packet is routed through the same virtual channel until it crosses a forbidden transition. In that case, the packet is routed through the next virtual

channel: VL1. So, the packet will change of virtual channel whenever it crosses a forbidden transition. Notice that the paths that cross the highest number of forbidden transitions impose the number of virtual channels required. This number depends, however, on the network topology, the underlying routing algorithm, the set of computed paths, and the number of faults to be tolerated.

Stage 3 In IBA, the use of virtual channels is devoted to purposes like *QoS*. Therefore, the number of virtual channels for fault tolerance issues should be limited. For example, if there are 4 VLs available, and the routing algorithm uses only one VL, then four QoS levels can be defined. However, if the routing algorithm uses 2 VLs, then only 2 QoS levels can be used. Therefore, the methodology must be designed to use a low number of virtual channels. Notice that using n virtual channels for fault tolerance, the number of ordered transitions between VLs can be at maximum $n - 1$.

Hence, at the third stage of the methodology (Figure 4.4) we only select those paths that traverse the lowest number of forbidden transitions. This selection is driven by the number of VLs available for fault tolerance. Notice that at this stage some alternative paths may be removed, thus affecting the obtained fault tolerance degree.

The number of computed alternative paths depends on the number of faults to be tolerated as well. In particular, under a dynamic fault model, $n + 1$ disjoint paths for every pair of end nodes are required to guarantee that all the possible combinations of n faults are tolerated. However, notice that the switch with the smallest number of ports connecting switches, and with at least one end node attached to it, bounds the number of tolerated faults. For instance, in a 2D torus network with one end node attached to each switch, the maximum number of disjoint paths for every source-destination pair is equal to four. Figure 4.7 shows an example with four disjoint paths in a 2D torus.

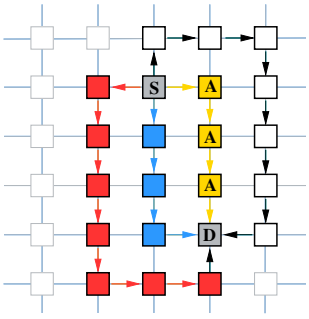


Figure 4.7: Example of a set of four disjoint paths in a 2D torus.

Stage 4 At the fourth stage the methodology deals with mapping conflicts. In Section 2.2.3 we showed that mapping conflicts may occur in InfiniBand. They occur when two paths (used by two packets) pass through the same input and output ports in

a given switch, they are labeled with the same SL identifier (this label never changes along the path), but each path (packet) requests a different output virtual channel. In that situation, the SLtoVL table can not be defined to accomplish the requirements of both paths.

Whenever a mapping conflict is encountered, the methodology tests firstly if the path (packet) requiring a smaller number of virtual channels could be modified from the switch with the conflict until its destination. The modification to be done is to use the same virtual channel in both paths (packets). As a second alternative, the methodology searches an alternative path for one of the paths causing the mapping conflict, preferably with the smallest possible length. Notice that short paths tend to be the best option for fault tolerance and to minimize mapping conflicts. This is because they use a smaller number of links and components which could be shared with other paths. Additionally, shorter paths have a lower probability of being affected by a new fault. The alternative new path must be disjoint or, at least, exhibit the minimum sharing with the rest of paths between the same source-destination pair. As a third alternative, the methodology uses an available new SL.

Otherwise, if none of the previous solutions work, one of the paths that incurs in the mapping conflict will be discarded. Notice that, in this case, the total number of alternative paths is reduced, and it will lead to a decrement in network reliability. Therefore, the effectiveness of TFTR will depend on the number of service levels available.

Figure 4.8.a shows an example of a given set of four disjoint paths for a given source-destination pair. In the switch labeled as T path A requires a transition and path B (which has a different source and destination) requires to keep routing trough the same virtual channel. Remember that each packet has an associated SL, and it can not be changed along their path. In the example, paths A and B have the same SL, therefore there is a mapping conflict at switch X as both paths require different output virtual channels and both use the same input and output port. The four possible solutions to the problem are shown in the figure. Firstly, both paths use the same VL at switch X (Figure 4.8.b). Secondly, an alternative path not introducing mapping conflicts is computed (Figure 4.8.c). Thirdly, a different SL is used for one of the paths (Figure 4.8.d). And finally, one of the paths is removed (Figure 4.8.e).

Stage 5 From the previous stage, the use of virtual channels is unbalanced, as traffic is mainly mapped to virtual channels with lower IDs. In particular, most of the traffic is usually mapped to VL0. All paths start using VL0, whereas only a relative low percentage of paths cross at least one routing restriction. This causes a traffic unbalance over virtual channels. Indeed, near all the traffic is mapped to the first VL. This may lead to a high degree of Head-of-Line blocking (HoL) at each switch and, thus, it will impact network performance.

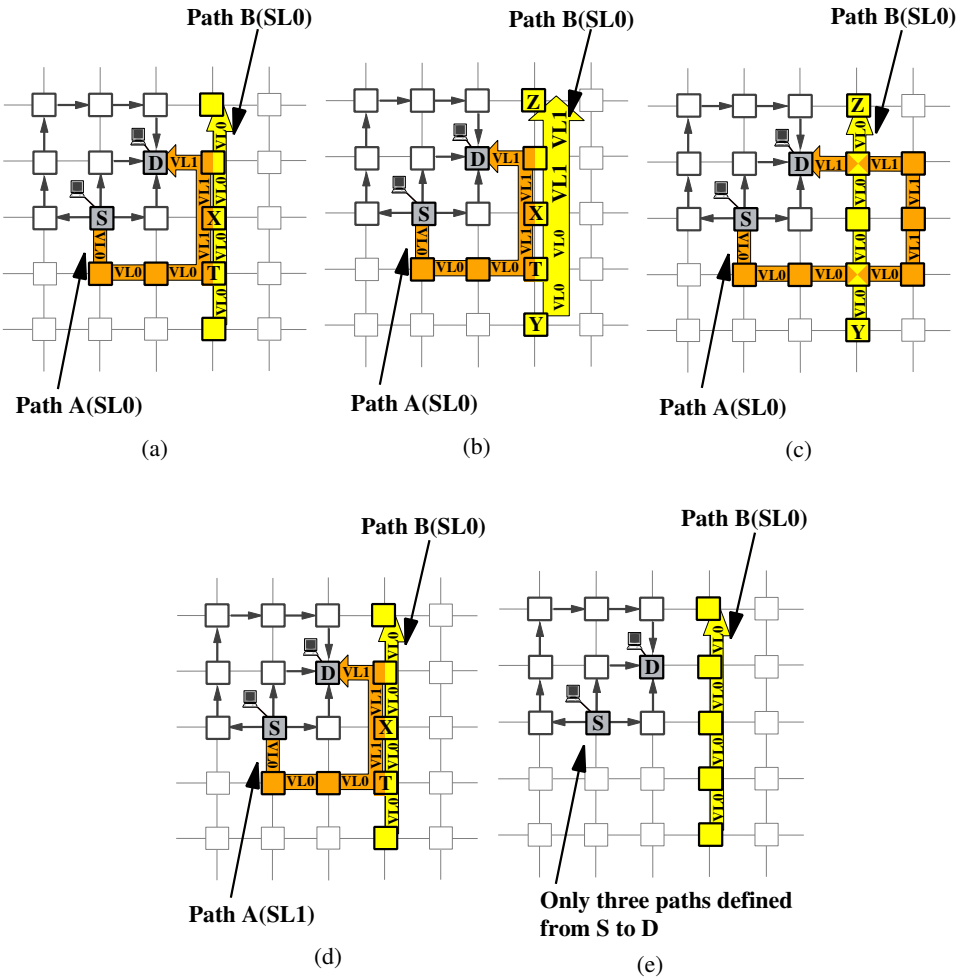


Figure 4.8: Examples of different ways to solve a mapping conflict. (a) Example of a mapping conflict, (b) mapping conflict solved by changing the VL, (c) by using an alternative paths, (d) by using a new SL, and (e) by discarding one path.

The Head-of-Line blocking phenomenon happens when the packet placed at the header of a buffer (switch input buffer) is blocked as the required output channel is busy (different packets compete for the same channel), preventing the packets behind the blocked packet in the buffer (probably requiring a free output channel) from being routed. The main problem of the HoL blocking is the reduction of the transmission speed of the buffers while packets are waiting. The contention is back propagated, and packets which do not require the contented resource become contented also. This effect influences negatively network performance, as shown in [136].

At the fifth stage, traffic is distributed among virtual channels. Most the paths do not traverse any forbidden transition, they would be mapped only on VL0. To balance traffic among virtual channels, some of these paths can be mapped on other VLs. This can be easily achieved by modifying the SL assigned to the path. This change is carried out only if it does not introduce a new mapping conflict. On the other hand, it is also possible that a packet transitions to a new virtual channel even if it is not crossing a forbidden transition (with the aim of avoiding mapping conflicts). Thus, when modifying the SL of a path an additional VL transition needs to be introduced. Remember that packets are allowed to move from virtual network i to virtual network j only if $j >= i$ ($j > i$ in the case of crossing a forbidden transition). For instance, Figure 4.9 shows a path which is injected using virtual channel #0, and later uses virtual channels in increasing order.

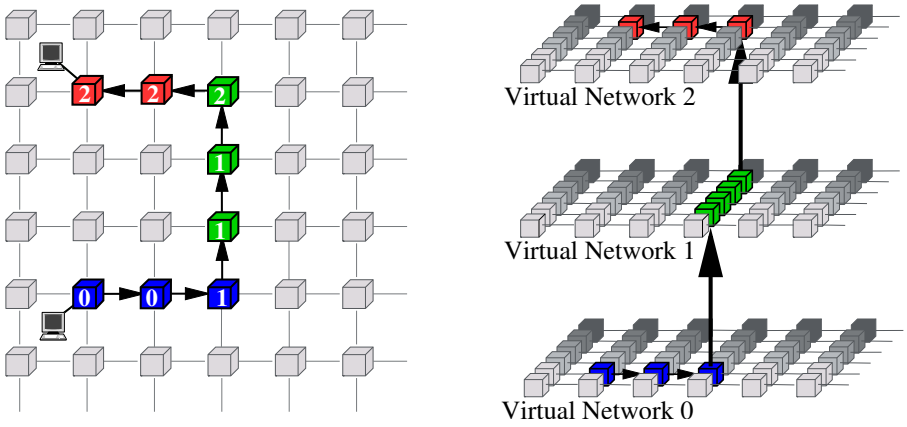


Figure 4.9: Example of multiple transitions between virtual networks.

Stage 6 Finally, at the sixth stage, the routing information is computed using the IBA format. This includes obtaining the forwarding tables, the SL-to-VL mapping tables, and the tables that map destination ID's with SLs at end nodes.

Notice that in a network with deterministic distributed routing, packets are routed in the network only using the destination ID located at the header of the packet. In order to provide multiple paths to reach the same destination we need to use the virtual addressing mechanism provided in IBA. Each destination will have, then, an address range. Each of these addresses (DLIDs) will have a different entry at the forwarding table on every switch. Therefore, because DLIDs are different (though they represent the same physical destination) it is possible to define multiple routing paths for the same source-destination pair. In order to use the smallest number of DLIDs for each destination, the same DLIDs will be reused from different sources when possible.

As a summary, Figure 4.10 shows a simple example of the TFTR method when applied to a particular source-destination pair. The example shows the case for the path starting at the source end node attached to switch S and ending at the destination end node attached to switch D. In the first stage all paths are computed. In the example, up to 9 paths are considered. At the second stage, all the routing restrictions due to the applied routing algorithm are computed. In the example, the routing restrictions obtained from the routing algorithm are plotted in blue. At the third stage, all those paths which require more VLs than the available ones are discarded. In the example only two virtual channels are available for fault tolerant issues. As can be seen, paths #8 and #9 in the figure cross more than one forbidden transition and, thus, are discarded. At the fourth stage, mapping conflicts are solved, and the final SL value is assigned to each path. Additionally, those paths labelled with a SL higher than the bounded number of SLs are discarded. Paths # 2 and #4 are discarded at stage 4 as they introduce mapping conflicts. At the fifth stage, the final set of disjoint paths is selected, choosing those paths with the maximum degree of disjoint paths. Also, some SLs are modified in order to balance the traffic over the different virtual channels. For instance, paths #1 and #6 are discarded at stage 5, keeping paths #3, #5, and #7 since they exhibit the highest degree of disjointness. Finally, in the sixth stage, the IBA tables are obtained.

4.2.2 Enhancements to the TFTR Methodology

At the first stage, the way paths are computed is critical for the methodology. As network size increases, the number of possible paths for each source-destination pair grows exponentially, and therefore the computation time can be extremely high. Even, computing all the paths may become infeasible. Thus, an efficient way of computing paths must be used to keep the computation time low. Two solutions are provided to solve this problem. When both solutions are combined the computation time of disjoint paths is significantly reduced.

A first improvement consists of bounding the complexity of the first stage of TFTR by reducing the total number of possible paths. We limit the length of the disjoint paths to be searched. In particular, for 2D tori we have limited the length of the paths to the topological distance between nodes plus 4. Notice that when applying the methodology we discard the longest paths and we keep the shortest ones. Therefore, is not worth computing large paths that will be discarded and will only contribute to increase the computation time.

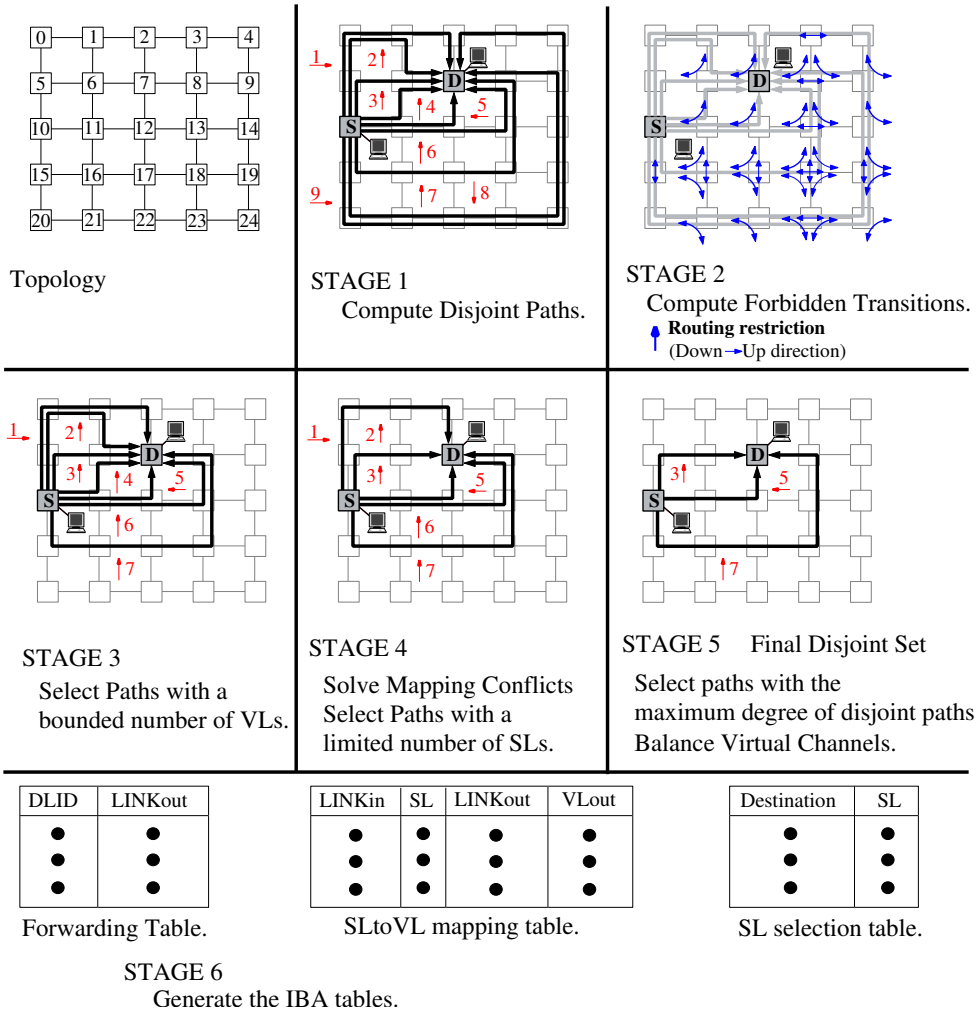


Figure 4.10: Detailed example for a given source destination pair when applying the TFTR methodology (in a 2D mesh).

The methodology takes into account that each possible disjoint path for a particular source-destination pair must use different ports both at the source switch (the switch where the source end node is attached to) and at the destination switch (the switch where the destination end node is attached to). Therefore, in the case of a 2D torus network and other planar topologies, the methodology can be further improved. In particular, an established order in which links are considered when computing the different disjoint paths may speed up the computation stage as well. Consider as an example the case of a 2D mesh in Figure 4.11.a. Three disjoint paths have been already computed for a particular source-destination pair. However, in that situation it

is impossible to obtain a fourth disjoint path as some links are not reachable without sharing a link or a switch with the already computed paths. Notice that the first computed path influences on the chances of obtaining the remaining three disjoint paths. The path starting from the west port at switch S and the path starting from the east port at switch S, prevent a path starting from the south port from reaching the destination switch D without sharing any link or switch with the other paths.

A way to solve this problem, and at the same time to compute paths in a fast and efficient manner, is by considering ports at the source switch in the clockwise direction and ports at the destination switch in the counter-clockwise direction. By doing so, the set of disjoint paths will be quickly computed. Figure 4.11.b shows an example of four possible disjoint paths that follow this rule.

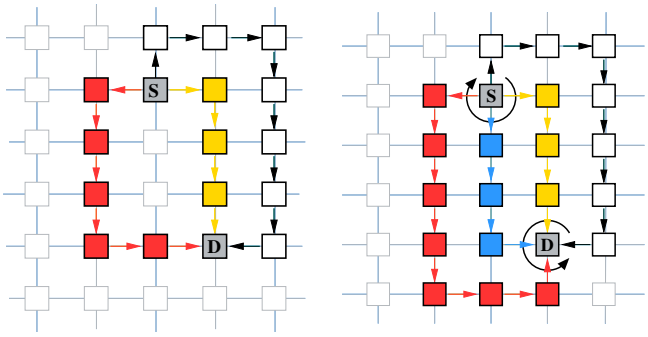


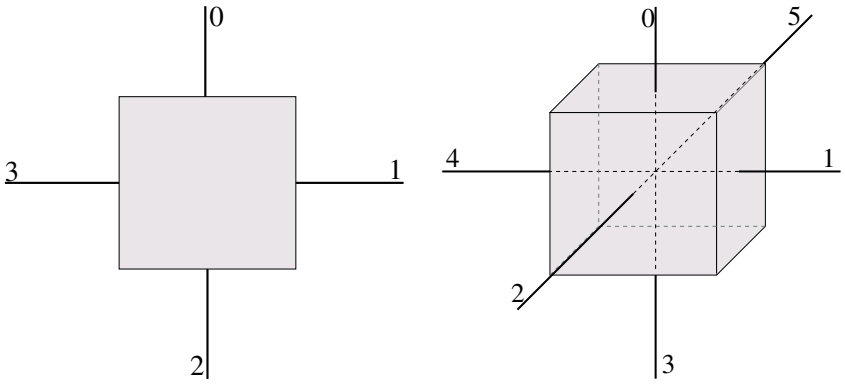
Figure 4.11: Example of disjoint paths.

This process will be repeated for the source-destination pair but considering a different initial port at the source switch and a final port at the destination switch. Thus, only 16 combinations are searched. This is the second optimization included in TFTR.

For other network topologies, similar rules should be extracted in order to keep a low computation time. Considering there are N dimensions and each link has two links in each dimension, links at the source switch and destination switch are ordered in such a way that one channel for each dimension is considered before considering the remaining channels. Figure 4.12.a shows the search order for 2D topologies, and Figure 4.12.b for 3D topologies.

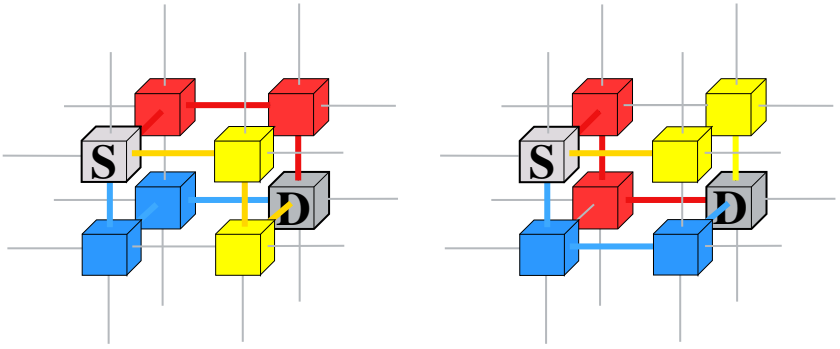
Figure 4.13.a shows two switches labelled as S and D. The minimum distance between them is three hops (one hop in each dimension). For these two switches, there are only two possible sets of three disjoint minimal paths. Figures 4.13.a and 4.13.b show the two possible sets of three minimal disjoint paths. As can be deduced, the solution follows the optimization just described above.

Alternatively, in the case both source and destination switches are in the same plane (XY, XZ or YZ), four of the disjoint paths must be in the same plane in order



(a) (b)

Figure 4.12: Link ordering in order to speed up the computing process of the disjoint paths: (a) for 2D tori/meshes and (b) for 3D tori/meshes.



(a) (b)

Figure 4.13: (a,b) Examples of two sets of the minimal disjoint paths in a 3D torus.

to optimize the average path length. The way these four paths (in the same plane) are computed follows the rule described for 2D topologies.

As a summary, Figure 4.14 shows the stages of the enhanced TFTR method (including both optimizations). Now, TFTR computes one by one the n paths for each source-destination pair of end nodes. Initially the entries in the SLtoVL tables are not defined. They are updated whenever a new set of n paths for a given source-destination pair of end nodes is computed. The process of computing disjoint paths is repeated for each source-destination pair.

4.2.3 Computational Cost

As may be expected, the complexity of TFTR is quite high (please refer to Section 8.1.7) since each path must be computed considering a large number of options (the

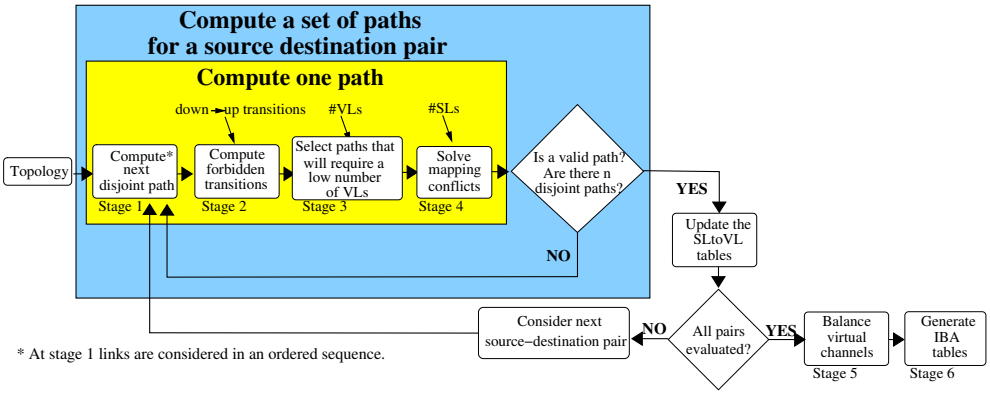


Figure 4.14: Stages of the enhanced TFTR version.

computational cost of the algorithm is driven by the first stage). However, it must be pointed that this process is done offline. Nevertheless, the mechanism in the following chapter will address this issue, because it has to compute the solution online.

4.3 SPFTR Methodology

In the previous section we have introduced the TFTR methodology. The most important benefit of the methodology is that it allows the computation of alternative disjoint paths. However, at the down-side the computation process is not scalable and it may take too much time. This is mainly due to the fact that mapping conflicts may occur during the computation process. To overcome this problem we present in this section a methodology that delivers the maximum number of disjoint paths between every pair of end nodes[3] allowed by the torus topology when implemented in IBA, in such a way that the computation time is extremely low and the number of resources used (VLs and SLs) is minimum and remains constant regardless of network size. The mechanism is referred to as *Scalable Pattern-Based Fault-Tolerant Routing* (SPFTR).

4.3.1 Motivation

The TFTR methodology provides fault tolerance by means of the use of alternative disjoint paths for any topology. In order to obtain a rich set of paths, this methodology allows transitions between virtual channels. This is achieved using an underlying routing algorithm.

As it will be seen in Chapter 8, the TFTR methodology requires the use of only two virtual channels to obtain four disjoint paths for every pair of end nodes in a 2D

[3]Remember that we do not consider the NICs at end nodes nor the first and last links along the paths.

torus network. Unfortunately, SL requirements increase with network size (results are shown in Chapter 8) due to the occurrence of mapping conflicts. Mapping conflicts are solved by using a different path (if any) or by using a new SL (not causing a mapping conflict). Taking into account that in InfiniBand the maximum number of SLs is 16 and they are intended for other purposes (mainly QoS), we may conclude TFTR does not scale. For instance, TFTR requires using 2 VLs and 7 SLs for a 10×10 torus network. Therefore, only two traffic classes could be used at most in conjunction with TFTR, which would significantly limit the QoS capabilities offered by IBA technology. Additionally, it has to be noted that, as network size increases, the average length of paths also increases. Therefore, more mapping conflicts arise, leading to an increase in the number of SLs required. This is a serious problem for IBA and requires an effective solution.

To this end, the SPFTR method uses a bounded set of resources (virtual channels and service levels) regardless of network size. To achieve such results, we need to apply a new methodology to compute disjoint paths, introducing novel concepts, models and computation strategies. SPFTR computes four disjoint paths for a 2D torus network by requiring only two VLs and four SLs regardless of network size. Thus, it guarantees the existence of up to four QoS channels and, at the same time, tolerates up to 3 faults. Moreover, the methodology generates a database of route patterns, which allows the computation of the disjoint routing paths in a time-efficient manner for any network size. In particular, the cost of the methodology is $O(N^2)$, where N is the number of switches in the system. Also, the methodology is easily extended to higher dimensional torus networks, without increasing the number of resources required.

The aim of SPFTR is to compute the forwarding and SLtoVL tables for IBA torus networks in such a way that they provide the maximum number of disjoint routing paths ($2n$) between every pair of nodes in a n-dimensional torus. Thus, up to $2n - 1$ faults are tolerated. To achieve this goal the methodology must deal at the same time with different issues:

- It must ensure that any combination of the computed disjoint paths ($2n$ for every source-destination pair) does not lead to deadlock. Deadlock freedom is ensured by the routing algorithm used.
- It must use only 4 SLs and 2 VLs, regardless of network size. This is achieved by computing the SLtoVL tables in an appropriate manner.
- It must be scalable. To achieve this, network regions whose switches require the same contents for their SLtoVL tables are defined.
- It must be simple enough in order to be computed in a time-efficient manner in large networks. SPFTR will obtain a set of route patterns to be used as templates to define the set of final disjoint paths.

All these issues have direct dependencies between them, and this fact makes obtaining the methodology quite challenging. The choice of the routing algorithm influences the route patterns that can be used. At the same time, the route patterns used influence the definition of the SLtoVL tables. In the other way, the definition of SLtoVL tables influences also the route patterns that can be applied.

Next, we describe the methodology using a 6×6 torus as an example. Later, the methodology is extended to larger 2D and higher dimensional torus networks. The methodology is evaluated in terms of fault tolerance, performance, and cost in Chapter 8.

4.3.2 SPFTR stages

Figure 4.15 shows the three stages followed by the methodology. Firstly, SLtoVL tables are initialized according to some initial rules: up to two virtual lanes can be used and transitions among virtual lanes can be done only in increasing order (first VL0, then VL1) in order to avoid deadlocks.

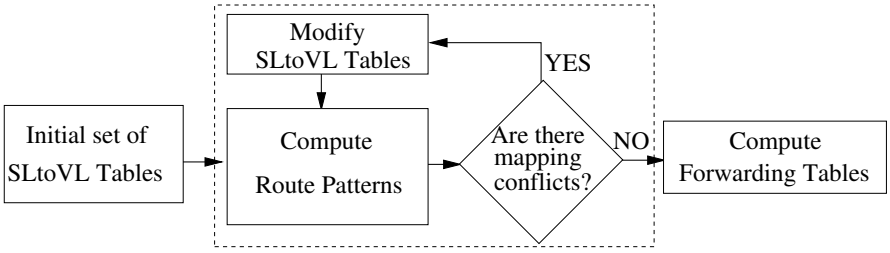


Figure 4.15: Steps followed by SPFTR.

In the second step, we look for a set of route patterns that accomplishes the restrictions imposed by the SLtoVL tables. This step tries to get short paths, and is repeated as many times as required in order to get the final set of paths without introducing mapping conflicts. In other words, once route patterns are computed, mapping conflicts are searched. If there is at least a mapping conflict, then the set of SLtoVL tables is changed accordingly and a new iteration is performed.

Obviously, the computational cost of this procedure is not bounded and could become very high (in the same sense TFTR is). However, SPFTR takes advantage on the fact that once all the routes have been successfully computed, it generates a unique definition of SLtoVL tables and route patterns that can be used in any 2D torus regardless of its size. Therefore, the methodology requires a low computational cost when applied to larger networks (once it was successfully applied to a small 2D torus). In particular, the computational cost is $O(N^2)$, where N is the number of end nodes. This is because the SLtoVL tables are already computed for each

source-destination pair. Therefore, we only need to apply the already computed route patterns to obtain the new forwarding tables.

Comparing TFTR and SPFTR, both are strategies are based on the use of disjoint paths. However, the stages followed by SPFTR seem to be in the opposite order. Mainly, TFTR computes the paths at the first stages, and the selected paths define the entries of the SLtoVL tables. On the contrary, SPFTR, in order to minimize the number of mapping conflicts and the number of required SLs, defines at the first place the entries of the SLtoVL tables, and later the paths that match with them. An important difference is that SPFTR is able to provide a set of disjoint paths by keeping minimal the SL and VL requirements, whereas the number of SLs required by TFTR may prevent its use on large networks. A second difference, is that SPFTR requires lower computational cost when applied to larger networks (once it was successfully applied to a small 2D torus) than TFTR.

4.3.3 Routing Algorithm

The methodology assumes a deterministic routing algorithm which guarantees deadlock freedom by building an acyclic channel dependency graph (CDG). This is achieved by enforcing some routing restrictions at the switch level, which prevents packets from traversing some consecutive links (forbidden transitions).

To do this, an underlying deadlock-free routing algorithm is used (in the same way as in TFTR). This routing algorithm indicates where the forbidden transitions are placed. In order to increase the routing flexibility when looking for disjoint paths, some virtual lane transitions are allowed. As commented, the proposed methodology will use only 2 VLs. This means that every path can traverse at most one forbidden transition (i.e. only one virtual lane transition can be carried out by a packet at most). In addition, virtual lanes must be used in an ordered way (for example, first VL0, then VL1), in order to avoid cycles in the CDG.

Thus, the key factor when defining our routing scheme is the selection of the appropriate underlying routing algorithm. This algorithm must be selected in such a way that it guarantees four disjoint paths for every pair of end nodes and it allows also route patterns that are valid regardless of the network size.

To achieve this we propose an ad-hoc routing algorithm. The algorithm is derived from the *up*/down** routing algorithm. In our case, the labeling of links is slightly different from that performed by the original *up*/down** routing. This assignment of directions to links is performed by building a certain spanning tree from the network graph. To do this, the switch in the top left corner is selected as the root (according to Figure 4.16.b). Unlike the original *up*/down** routing, which builds a spanning tree from the complete network graph, the proposed strategy proceeds to remove all the wraparound links of the torus before building the *BFS* spanning tree. The BFS spanning tree is then computed from the root switch, and switches are grouped in levels

based on the topological distance to the root switch. The root switch corresponds with the switch with the smaller identifier (in the plots the root switch is identified with an R label). This scheme has the property that once the root switch is selected, the rest of switches is distributed automatically on a tree. Once the BFS spanning tree is built, the wraparound links are added, and a particular set of routing restrictions is added in order to break cyclic channel dependencies in the CDG. In particular, we identify all the possible cycles that can be formed and proceed to remove them by imposing the corresponding routing restrictions. The distribution of the routing restrictions can be seen in Figure 4.20.b. Deadlock freedom is guaranteed by verifying that the resulting CDG is acyclic. Also, connectedness is guaranteed.

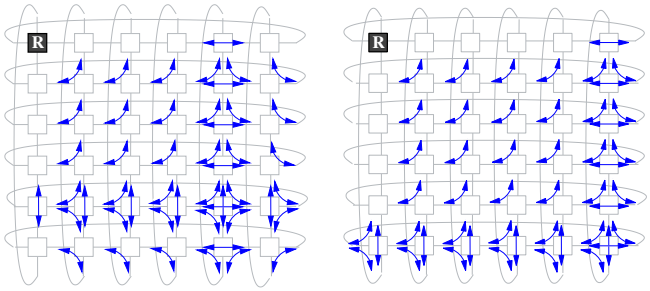


Figure 4.16: Routing restrictions of (a) *up*/down** routing algorithm and (b) the ad-hoc underlying routing algorithm used for SPFTR.

The main advantage of this new routing algorithm is that, in most places, the orientation of the routing restrictions is the same, as can be seen in Figure 4.20.a, which definitely will contribute to simplify the computation process of SLtoVL tables carried out by the methodology[4].

Notice that this algorithm has not been designed with the aim of providing minimal paths between every pair of nodes nor minimizing the number of routing restrictions. Instead, it is only used as an underlying routing algorithm to guarantee deadlock freedom and, most important, to provide the symmetry required to simplify the methodology. Notice that this scheme is still able to provide minimal paths by traversing at most a forbidden transition and carrying out the corresponding virtual channel transition.

4.3.4 Route Patterns and Network Regions

Route patterns constitute templates that can be used to obtain all the disjoint routing paths between every pair of end nodes. A route pattern is defined by a sequence of

[4]We have considered other routing algorithms, such as DOR and *up*/down**. However they are not able to provide an scalable methodology using the minimum number of SLs.

movements (e.g., “go to next switch on the left” or “go to the left until reaching the destination’s column”) and a SL, which defines the virtual channel to be used at each hop of the path. This sequence of movements must be compatible with the routing scheme applied in order to guarantee deadlock freedom. Moreover, the SL identifier must be selected in such a way that mapping conflicts are avoided.

In order to illustrate how the virtual channel is selected as a function of the SL, we consider as an example the path shown in Figure 4.17. The Figure shows a path between two end nodes. The links at switch Y are input link #3 and output link #1. Then, considering that the path uses SL0, the value of the virtual channel to be used when leaving switch Y is obtained from Table 4.1, which indicates virtual channel number #0.

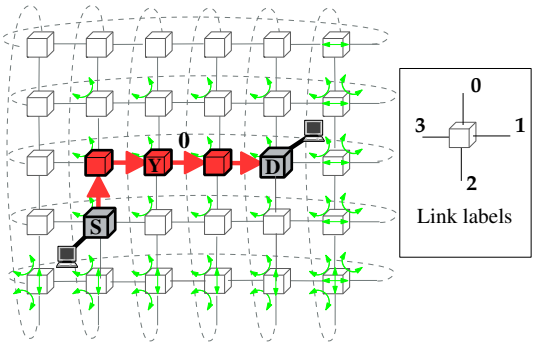


Figure 4.17: Example of a path using SL=0.

Input port	Output port	SL	Output VL
0	1	0	0
0	2	0	0
0	3	0	1
1	0	0	0
1	2	0	0
1	3	0	0
2	0	0	0
2	1	0	0
2	3	0	0
3	0	0	1
3	**1**	**0**	**0**
3	2	0	0
local	X	0	0
X	local	0	1

Table 4.1: SLtoVL table for switch Y and SL=0. X stands for any port.

Additionally, the methodology defines network regions. A network region is built by neighbour switches with the same SLtoVL table definition. Network regions are defined taking into account two conditions. The first one is that all the switches in a region must have the same routing restrictions (this is obvious as they must have the same SLtoVL table). And, second, in order to help the methodology to scale, the same distribution of regions should be kept regardless of network size. Figure 4.18 shows the final regions defined for the 6×6 torus network. As can be observed, four regions have been defined so that fulfil the two conditions referred above: every switch in a region has the same SLtoVL table definition, and the same definition of regions can be obtained for larger networks.

The main advantage of using regions is that it minimizes the number of required route patterns. Route patterns depend on the regions where source and destination end nodes are located (the same or different) and their relative positions (i.e., they

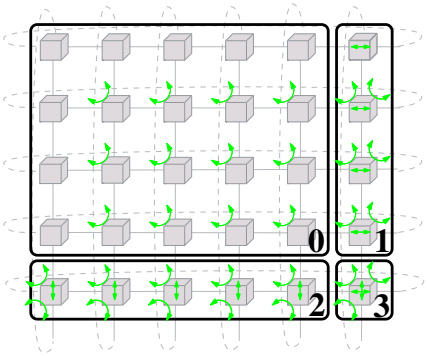


Figure 4.18: Regions in a 6×6 torus network when using SPFTR.

may be located either in the same row and/or column or in different rows and/or columns). To generate the route patterns, we try to obtain four disjoint paths (as short as possible) between every pair of end nodes. To this end, it is necessary to establish the path followed by the pattern (sequence of movements) and the SL to be used. The former is accomplished by following the restrictions imposed by the applied routing algorithm (required VL transitions were already introduced in the initial SLtoVL tables). The latter is carried out according to the current SLtoVL table entries and the bounded number of SLs used (4 SLs), checking that mapping conflicts are not introduced. If it is not possible to obtain a set of valid route patterns, SLtoVL tables must be slightly updated (manually) and a new try to obtain route patterns is carried out. Thus, an iterative process is performed through the two steps until a valid set of route patterns is found.

To illustrate the process of computing route patterns, let us consider a source-destination pair of switches belonging to the same region. Without loss of generality, let us consider neighbor switches sharing the same row. Figure 4.19 shows possible route patterns able to provide four disjoint paths between the end nodes (depending on their position). Several alternative sets of route patterns must be considered. For example, in sets #1 and #4, if both S and D switches are moved one row down, the first set of patterns can not be applied, as one of the paths would cross three forbidden transitions (two additional forbidden transitions). In this case, the other sets of paths can be used: set #2 substitutes set #1, and set #3 substitutes set #4.

4.3.5 SLtoVL Table Initialization

As commented above, the first step of the methodology is to initialize the SLtoVL tables. In particular, SLtoVL tables must initially contain those entries corresponding

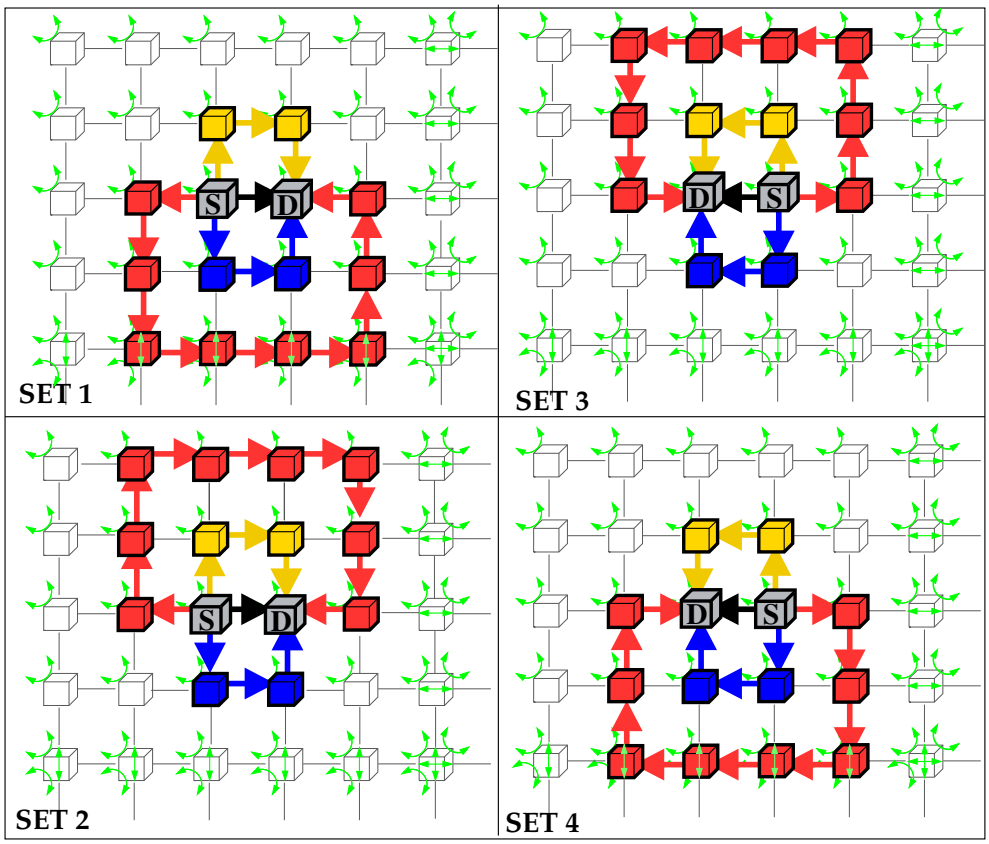
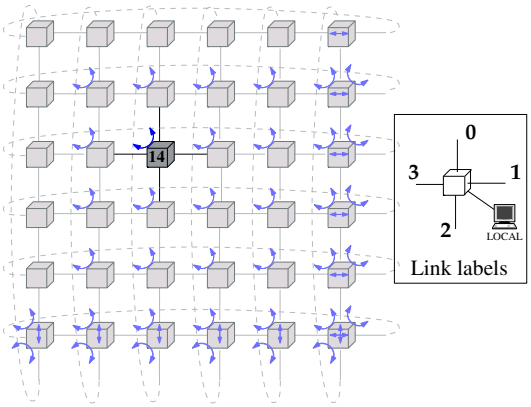


Figure 4.19: An example of route patterns for a source-destination pair of nodes placed at the same region #0.

to switch transitions (In Port - Out Port) that require performing a virtual lane transition (i.e., transitions forbidden by the underlying routing algorithm). For instance, Figure 4.20.b shows the initial SLtoVL table corresponding to a switch in the center of the network. The labeling of links[5] is shown in Figure 4.20.a. Specifically, the entries corresponding to link transitions 0 - 3 and 3 - 0 are forced to use VL1 (shown in bold face in Figure 4.20.b) because both link transitions are forbidden, as can be seen in Figure 4.20.a.

However, note that setting the SLtoVL table in such a way does not guarantee on its own deadlock freedom. In order to enforce deadlock freedom packets using these entries should enter the switch exclusively through VL0. This fact must be taken into account when computing the routing paths. The rest of SLtoVL entries

[5]This labeling in 2D topologies is used throughout the entire document.



(a)

Input port	Output port	SL	Output VL
0	1	X	N/D
0	2	X	N/D
0	3	X	**1**
1	X	X	N/D
2	X	X	N/D
3	0	X	**1**
3	1	X	N/D
3	2	X	N/D
local	X	X	N/D
X	local	X	N/D

(b)

Figure 4.20: Underlying routing algorithm applied: (a) routing restrictions, (b) SLtoVL table initialization for switch 14. N/D stands for Not Defined. X stands for any value.

are not enforced at this stage by SPFTR. The methodology will update those entries accordingly in the next stage.

Entries corresponding to packets coming from an end node (input port is local) are assigned to 0 (VL0). This is done to allow the packet the chance to migrate later to a higher VL if needed. Similarly, packets leaving the network (output port is local) will use VL1. These entries will not be modified at any stage of the methodology. The remaining entries, however, will be modified when searching for SLtoVL tables and route patterns.

Within each network region, all the switches have the same contents in their SLtoVL table. Thus, when mapping a path into the region, each time the path crosses a switch, the output VL required by the path must coincide with the appropriate entry in the SLtoVL table of the region (corresponding to the SL and the input and output links used). Figure 4.21 shows some examples of paths mapped into the same region for four different values of SLs. At each switch a number indicates the VL value, which matches the appropriate entry on the SLtoVL table. Notice that the two paths for each SL define all the possible entries in the SLtoVL table. Therefore, because all switches in the region have the same SLtoVL table, any of these paths can be placed at any place on the region without introducing mapping conflicts. Notice also that in figure, virtual channels transitions only occur when the packet experiences a change in its direction. This fact allows paths (in the figure) to be extended along the same axis. In particular, at switches where the path does not change its direction. Indeed, mapping conflicts at those switches can not happen. This, (taking this into account)

final SLtoVL tables and route patterns are computed in a fast manner. In particular, the VL values used in the examples (Figure 4.21) are used as the initial values for the SLtoVL tables of region #0. This reduces the computation time of the final route patterns and SLtoVL tables.

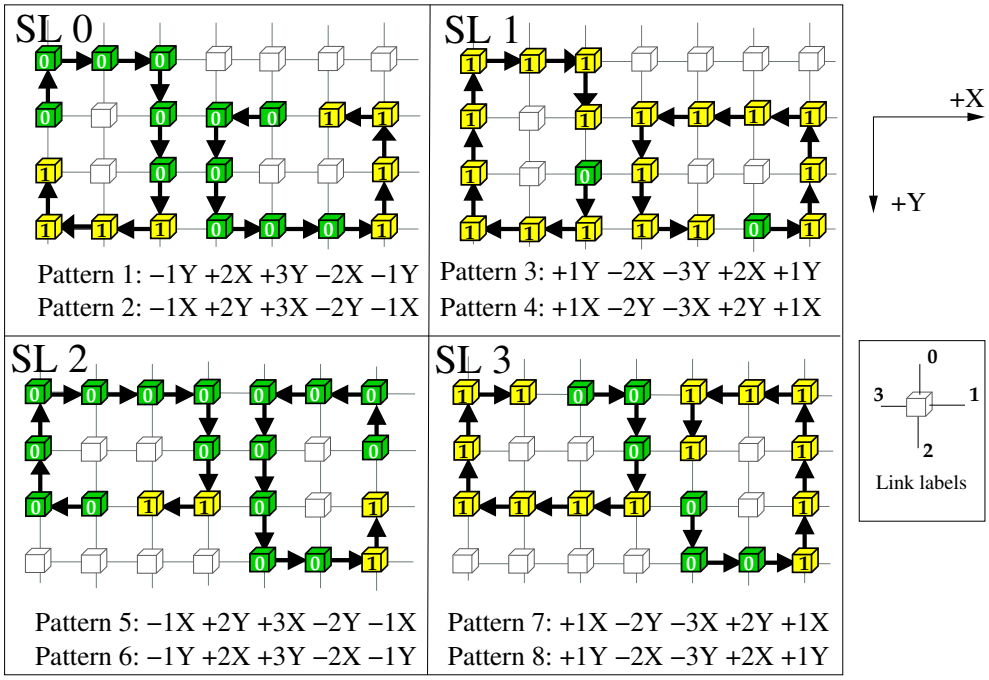


Figure 4.21: Initial route patterns used for initial definition of SLtoVL tables.

The remaining SLtoVL entries are set when defining each route pattern. This process requires some iterations until all the final patterns (not introducing any mapping conflict) are obtained. The SLtoVL table entries are progressively modified along the iterations. Notice that the table entries corresponding to forbidden transitions (those imposed by the underlying routing algorithm) should not be modified.

Table 4.2 shows the final SLtoVL table entries for every region defined on a 6×6 torus. These tables have been computed taking into account the route patterns applied for every source-destination pair. The contents of the final SLtoVL tables are the same for all the switches belonging to the same region. According to the defined regions and the possible relative positions of every pair of nodes, 236 route patterns have been obtained (see Appendix A).

To illustrate the process of computing the routing paths from the route patterns, without loss of generality, let us consider a pair of switches located in the same row in region 0 (the relative position of the switches and the regions will determine the

In port	Out port	Region 0 SL 0-1-2-3	Region 1 SL 0-1-2-3	Region 2 SL 0-1-2-3	Region 3 SL 0-1-2-3
0	1	0 1 0 0	**1 1 1 1**	0 1 0 0	**1 1 1 1**
0	2	0 1 0 0	0 1 0 0	**1 1 1 1**	**1 1 1 1**
0	3	**1 1 1 1**	**1 1 1 1**	**1 1 1 1**	**1 1 1 1**
1	0	1 1 0 1	**1 1 1 1**	1 1 0 1	**1 1 1 1**
1	2	0 1 0 1	0 1 0 1	1 1 0 1	0 1 0 1
1	3	1 1 0 1	**1 1 1 1**	1 1 0 1	**1 1 1 1**
loc.	X	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0
X	loc.	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1

In port	Out port	Region 0 SL 0-1-2-3	Region 1 SL 0-1-2-3	Region 2 SL 0-1-2-3	Region 3 SL 0-1-2-3
2	0	1 1 0 1	1 1 0 1	**1 1 1 1**	**1 1 1 1**
2	1	0 1 0 1	1 1 0 1	0 1 0 1	0 1 0 1
2	3	1 1 0 1	1 1 0 1	**1 1 1 1**	**1 1 1 1**
3	0	**1 1 1 1**	**1 1 1 1**	**1 1 1 1**	**1 1 1 1**
3	1	0 1 0 0	**1 1 1 1**	0 1 0 0	**1 1 1 1**
3	2	0 1 0 0	0 1 0 0	**1 1 1 1**	**1 1 1 1**
loc.	loc.	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Table 4.2: SLtoVL tables for every region in 2D torus network for SPFTR method. Values imposed by routing restrictions in each region are written in bold face. X stands for any value.

pattern to use). The route patterns defined for computing the routing paths can be shown in Figure 4.22. The computed routing paths in a 6×6 torus network can be shown in Figure 4.23.a.

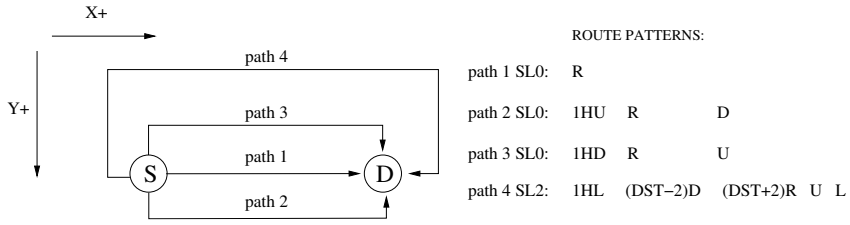


Figure 4.22: Example of a route pattern.

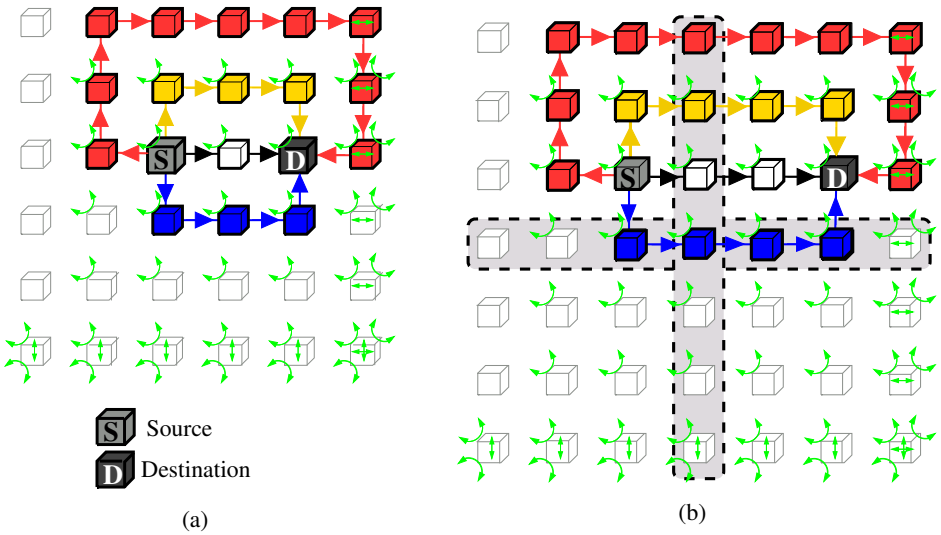


Figure 4.23: Example of pattern applied in a (a) 6×6 and (b) 7×7 torus network.

4.3.6 Extending the Methodology

In this Section we show how SPFTR can be applied to larger and higher dimensional torus networks. First, the methodology is extended to larger 2D torus networks, thus scaling the methodology. In a second effort, we extend SPFTR to 3D torus networks.

Extending SPFTR to a larger 2D torus. A larger 2D torus network can be viewed as a 6×6 torus network with additional rows and/or columns of switches. These new rows and/or columns are added in the middle of the torus network (taking as a reference Figure 4.24). Therefore, the new components will belong to regions 0 and 2 in the case of a column or to regions 0 and 1 in the case of a row. By doing this, the distribution of regions will remain the same as in the 6×6 torus. Therefore, SLtoVL tables of new switches are already defined. Also, the route patterns to use for every new switch to all the destinations have been already computed as each switch can use the route patterns of any of the switches within their same region. As an example, Figure 4.23(b) shows the paths computed for a given pair of switches for a 7×7 torus when using the route patterns shown in Figure 4.22.

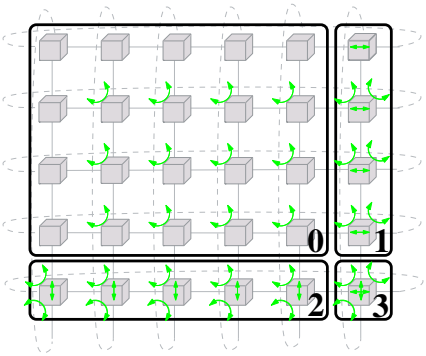


Figure 4.24: Regions in 2D torus when using SPFTR.

Extending SPFTR to larger n-D torus. In order to extend the methodology to n-dimensional torus networks, we use the same underlying routing algorithm described in Section 4.3.3, which defines the routing restrictions (Figure 4.28.a shows an example for a $3 \times 3 \times 3$ torus network). In particular, for every plane in the 3D torus (see Figure 4.25) we can find an orientation with all the restrictions located in the same position as in the 2D torus shown in Figure 4.24. As an example, Figure 4.28.a shows a frontal view of the routing restrictions found in a 2D plane of the 3D torus.

Taking into account that each entry in the SLtoVL table is defined for a certain SL value and a certain pair of links, and that both links are located in a plane, we can use the SLtoVL tables already defined for a 2D torus (shown in Table 4.2) in order to compute all the entries in the SLtoVL table for the n-dimensional torus.

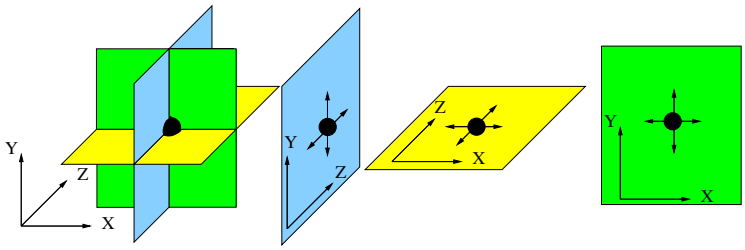


Figure 4.25: 2D planes in a 3D torus.

SLtoVL tables must be compatible at the plane level in order to extend the methodology to more than two dimensions. To achieve this, it is important to emphasize that every pair of links placed in the same direction is included in more than one plane (in two planes in the case of the 3D torus). Therefore, given that we define the SLtoVL entries for each plane independently of its position, and each table entry can only define a unique VL value, we add an additional rule: entries for the pair of links shared between planes (links along the same direction) must contain the same values in all these planes. We refer to this property as $symmetry$, and it restricts the definition of network regions and SLtoVL tables.

For instance, Figure 4.26 shows in the left hand side the six XZ planes in a $6 \times 6 \times 6$ torus. Switches colored in red are in the same position when considering only the *XZ* planes, thus, they are in the same region in the *XZ* plane and have the same entry in the SLtoVL table when the input and output links are in the X direction. Next, we consider the *XY* plane which includes these switches (in red). This *XY* plane is shown on the right hand side of the figure. Notice that red switches in the *XZ* plane (in the same column) are in more than one region. Therefore, some entries in the SLtoVL table defined for a plane force some values in other planes. Also different regions in the same plane (red switches in the XY plane in the figure) must have the same definition. These relationships appear on each possible pair of plains that intersect (see the Figure 4.27).

The route patterns for all the pairs of end nodes in the same plane are defined using the patterns obtained for the 2D torus (236 patterns). Notice that there are three planes in a 3D torus (XY,XZ, and YZ), thus we consider three times the quantity of patterns for the 2D torus.

When a source-destination pair is in the same plane, the route patterns for the four disjoint paths included also in this plane are the same defined for the 2D torus

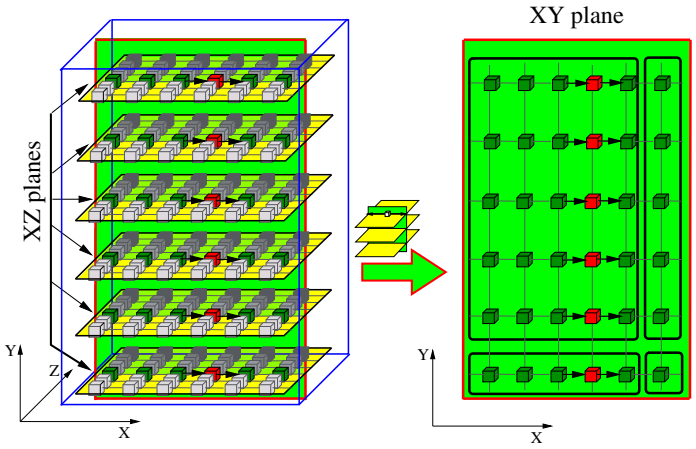


Figure 4.26: Switches in the same position in the *XZ* planes in a $6 \times 6 \times 6$ torus are included in the same position in a *XY* plane.

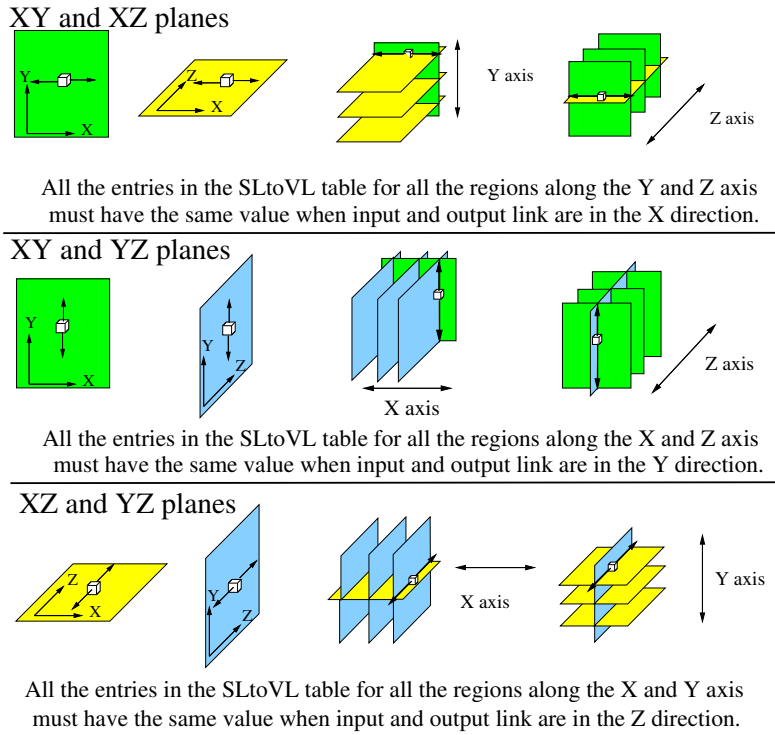


Figure 4.27: Pair of links shared between every pair of planes in a 3D torus.

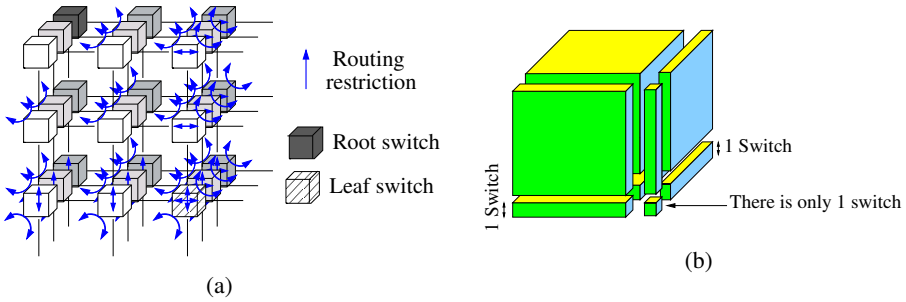


Figure 4.28: (a) Frontal view of routing restrictions in the XY plane of a 3D torus. (b) SPFTR regions in a 3D torus.

case. In particular, when the pair of end nodes are in the same axis, all the six disjoint paths can be defined also with the same route patterns for the 2D torus. However, new route patterns are needed for the pairs of end nodes which are not in the same axis.

The solution we obtain, which fits with these restrictions, provides in a 3D torus the total amount of 2322 route patterns and 8 regions. These regions are shown in Figure 4.28.b.

4.3.7 Optimizing the methodology: A-SPFTR

As an additional result from the proposed methodology, we have successfully obtained an alternative set of route patterns and SLtoVL table definitions that only requires 2 VLs and 3 SLs. The solution we obtain, 10 regions (show in Figure 4.29.a)in a 2D torus and uses similar set of route patterns as SPFTR for 2D torus.

This solution has been achieved by sacrificing the scalability in terms of higher dimensional torus networks, thus being only valid for 2D tori. However, scalability in terms of number of nodes is guaranteed. The resulting algorithm is referred to as Asymmetric-Scalable Pattern-based Fault Tolerant Routing (A-SPFTR). In this case, new rows can be added along regions 3, 4, 5, and 6, and new columns along regions 1, 5, and 8, keeping the same region structure shown in Figure 4.29.a. For instance, Figure 4.29.b shows an example of a 7×7 torus obtained from a 6×6 torus once an additional row and column is added. The final definition of the SLtoVL tables for each region is shown in Table 4.3.

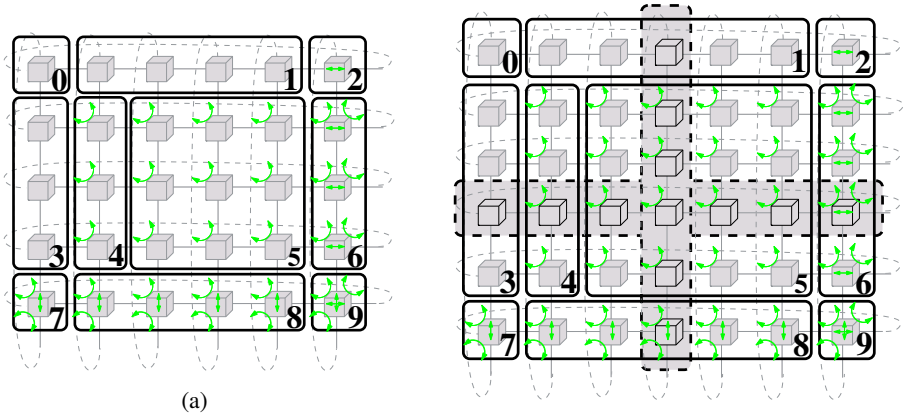


Figure 4.29: Regions in 2D torus when using A-SPFTR in (a) 7×7 torus network, and (b) in a 8×8 torus network.

		Regions									
		0	1	2	3	4	5	6	7	8	9
In Port	Out Port	SL 0-1-2	SL 0-1-2	SL 0-1-2	SL 0-1-2	SL 0-1-2	SL 0-1-2	SL 0-1-2	SL 0-1-2	SL 0-1-2	SL 0-1-2
0	1	0 0 0	0 0 0	1 1 1	0 0 1	0 0 1	0 0 1	**1 1 1**	0 0 1	0 0 0	**1 1 1**
0	2	0 1 1	0 1 0	0 1 0	0 0 1	0 0 1	0 0 1	0 0 0	**1 1 1**	**1 1 1**	**1 1 1**
0	3	1 1 1	1 1 1	1 1 1	1 1 1	**1 1 1**	**1 1 1**	**1 1 1**	**1 1 1**	**1 1 1**	**1 1 1**
1	0	1 1 1	1 1 1	1 1 1	0 1 1	1 0 1	1 1 1	**1 1 1**	1 0 1	1 0 1	**1 1 1**
1	2	0 0 1	0 1 1	0 0 1	0 1 1	0 1 1	0 0 1	0 1 0	1 1 1	0 1 1	0 0 0
1	3	0 0 1	0 1 1	**1 1 1**	0 1 0	0 1 1	0 1 1	**1 1 1**	1 1 1	1 1 1	**1 1 1**
2	0	0 1 1	1 1 0	1 1 1	0 0 1	1 0 1	1 1 1	1 1 1	**1 1 1**	**1 1 1**	**1 1 1**
2	1	0 0 1	1 0 1	1 0 1	0 0 1	1 0 1	0 1 1	1 1 1	0 0 0	0 0 0	1 1 1
2	3	0 1 1	1 0 1	1 1 1	0 1 1	0 0 1	0 1 1	0 1 1	**1 1 1**	**1 1 1**	**1 1 1**
3	0	1 1 1	1 1 1	1 1 1	1 1 1	**1 1 1**	**1 1 1**	**1 1 1**	**1 1 1**	**1 1 1**	**1 1 1**
3	1	1 0 0	0 0 0	**1 1 1**	1 0 1	0 0 1	0 0 1	**1 1 1**	0 0 1	0 0 1	**1 1 1**
3	2	0 0 1	0 0 1	0 0 0	0 0 1	0 0 1	0 0 1	0 0 1	**1 1 1**	**1 1 1**	**1 1 1**
loc.	X	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
X	loc.	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1	1 1 1

Table 4.3: SLtoVL tables for every region in 2D torus network for A-SPFTR method. Values imposed by routing restrictions in each region are written in bold face. X stands for any value.

Chapter 5

Fault-Tolerant Routing Methodology Based on Reachability

In this chapter we present a new mechanism to provide fault tolerance. Once a fault is detected some actions are performed and new alternative paths are computed to tolerate the fault. This is opposite to previous TFTR and SPFTR mechanisms, in which the alternative paths are computed ”a priori”. Thus, this new mechanism is dynamic in the sense that new paths are computed when needed. The main objective of this new approach is to tolerate more faults than the previous mechanisms.

5.1 Motivation

The methodology presented in this chapter is focused on the dynamic fault model (see section 3.2) when applied to networks with deterministic routing, which is the common case in commercial network technologies currently applied to PC clusters. In this scenario, there is no doubt that applying reconfiguration techniques is a good choice, because when using reconfiguration any fault scenario can be tolerated if the network keeps physically connected. That is, any fault combination could be tolerated with an appropriate reconfiguration mechanism. The term reconfiguration must be taken, however, carefully, since a single bit change or a change of the entire routing algorithm may be viewed as a reconfiguration process. However, the effects of changing a bit or the entire tables will be completely different.

At this stage, we can differentiate two situations where reconfiguration is needed. In a first scenario, we may need to compute a small set of new paths to overcome a new fault that has been detected within the network. With small modifications at the affected switches and end nodes the new fault can be tolerated. This procedure can

be defined as local reconfiguration since only a part of the network is reconfigured. As the process of reconfiguration is local it is expected that the entire process will be fast as well.

On the other hand, the occurrence of consecutive faults leads to consecutive local reconfiguration processes. Although the connectivity is maintained, the overall performance may be affected. In that situation, a complete change at the network level may be needed to recover the lost performance. To this end, a global reconfiguration process can be carried out where all the end node pairs are provided with a new path. As this process requires more paths to be computed it is obvious that it will take much time.

In this chapter we develop a solution for the first scenario. In particular, the proposed RFTR (Reachability-based Fault Tolerant Routing) methodology computes only new paths to those pair of end nodes affected by a new fault. This process is optimized to reduce the elapsed time between the fault detection until an alternative path is provided. The main goal, thus, is to speed up the processing of new fault-free paths. RFTR is not limited to a given set of faults tolerated. Indeed, it is able to compute new paths provided a sufficient set of resources (VLs and SLs in IBA) are available.

In the next chapter we propose a totally different approach based on a global reconfiguration process that deals with the second scenario: a total change of the routing algorithm. With this method all the paths for every source-destination pair are computed again and distributed among the end nodes and/or network switches. That means an important disturbance on the network. Additionally, dynamic reconfiguration often requires additional hardware resources in order to guarantee deadlock freedom during the reconfiguration process. This is because combinations of packets routed with both routing schemes (before and after the reconfiguration process) could form cycles. In the next chapter we will see how to handle this situation.

5.1.1 RFTR Preliminaries

When faults appear in the network they usually only affect to some paths. As an example, Figure 5.1 shows a 3×3 mesh using Dimension Order Routing (DOR). For the sake of simplicity, we assume that end nodes are attached only to switches A, B, and C. When link L fails, the path from A to B is affected, whereas the rest of paths are not affected at all by the fault. Thus, we need to compound only a new path from A to B. However, notice that DOR is no longer able to provide a fault-free path from A to B. Therefore, both end nodes are logically disconnected. In this situation, a reconfiguration process is required to restore connectivity.

As described in Section 4.2.1, virtual channels can be used to improve the flexibility of a routing algorithm. In this sense, it is possible to obtain a path from A to B with a $(Y \rightarrow X)$ transition at switch C. To this end, a virtual channel transition on C

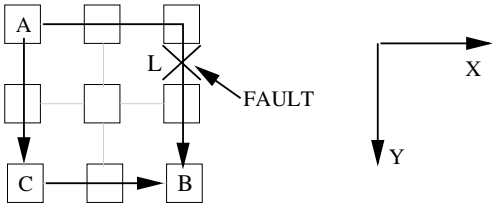


Figure 5.1: Paths in a 3×3 mesh applying DOR.

must be performed. Notice that by using virtual channel transitions the new path is fully compatible with the previous set of paths, thus guaranteeing that deadlocks can not arise. Indeed, a path performing a virtual channel transition can be viewed as two joined subpaths, each one on a different virtual channel or layer. Since each subpath does not introduce an illegal transition in its virtual channel, and virtual channels are used in an established order, deadlock freedom is guaranteed. Thus, a full reconfiguration process is not needed, and simply the new routing info for the new path can be distributed without stopping network traffic.

Notice that the new computed path can be obtained from the set of paths previously computed (before the fault). Indeed, the new path can be viewed as the A-C and C-B subpaths joined. Thus, instead of computing new paths (by a new routing algorithm), we can extract new ones from the already computed set of paths for only those pairs of end nodes affected by the fault. This will let us to achieve the following benefits:

- Firstly, the required amount of routing information to be updated (at end nodes and/or switches) will be lower, leading to send less control data.
- Secondly, the traffic not affected by the fault will be left unmodified (a kind of local reconfiguration process will suffice).
- Thirdly, the method will take less computation time (compared with a full reconfiguration method), as only affected paths will be computed.

As a consequence, the total required time would be smaller, causing a smaller percentage of packets to be lost during the reconfiguration process due to the fault.

Thus, the key issue consists of finding an effective methodology able to compute an alternative path for each path affected by the fault from the set of already computed paths. Moreover, this methodology should be able to compute the new paths in a time-efficient manner by using as few virtual channels as possible.

In order to undertake these challenges, we describe in this chapter a new fault-tolerant routing methodology referred to as Reachability-Based Fault-Tolerant Routing (RFTR). The method provides new paths by joining already computed subpaths and using virtual channel transitions when required. As a result, RFTR is suitable to

any topology, tolerates dynamically a large number of faults with a very small number of virtual channels, and exhibits a very low computational cost for any network size. This last benefit minimizes the number of lost packets during the process of fixing the fault.

It has to be noted that the methodology does not depend on neither the hardware used for detecting faults, nor the way faults are notified. Anyway, as an example of applicability, RFTR is applied to IBA. In Section 8.3 the methodology is evaluated in terms of fault tolerance, performance, and resource requirements when applied to IBA. Also, RFTR is compared with the previous fault-tolerant mechanisms proposed in this thesis.

5.2 RFTR Description

In this Section RFTR is described in detail. The methodology is based on the concepts of direct and indirect reachability. Thus, we first introduce both concepts and then present the methodology.

5.2.1 Direct and Indirect Reachability

Given a routing algorithm and a pair of switches (A and B), it is said that switch B is directly reachable from switch A if the routing algorithm provides a path from A to B. Similarly, switch A is indirectly reachable from switch B if the routing algorithm provides a valid path from A to B.

Notice that reachability can be defined regardless of the type of routing algorithm used (adaptive or deterministic). However, we are interested in deterministic routing. Moreover, a deterministic routing (oblivious routing) may provide several paths for some pair of end nodes. Thus, it can be viewed also as a set of deterministic paths. We select just one path for each source-destination pair. Each path consists of a list of links and switches. Based on this view, let us define the reachability concept related to a given deterministic path: Every switch listed in a path is directly reachable from the source switch of the path (the switch the source node is attached to) and indirectly reachable from the destination switch of the path (the switch the destination node is attached to) [1].

In order to achieve a simple and fast computation method of alternative routing paths we create two tables, one to identify directly reachable switches and one to identify indirectly reachable switches. As an example, the first table (Table 5.1.a), referred to as direct reachability table (DRT), allocates entries for each path depicted in Figure 5.2.

[1] Notice that the definition does not include all the reachability info provided by the path, as reachability among intermediate switches is not considered. However, as it will be seen in the evaluation, this will not impact on the fault tolerance properties of RFTR.

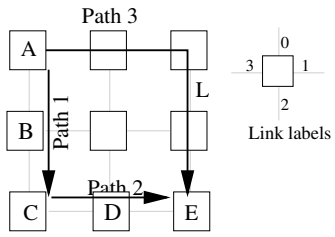


Figure 5.2: Paths in a 3×3 mesh.

Src	Dst	Reach	Trans.	In. Link	Hops
A	C	B	-	2	1
A	C	C	-	2	2
C	E	D	-	1	1
C	E	E	-	1	2

DRT (a)

Src	Dst	Reach	Trans.	Out. Link	Hops
A	C	B	-	0	1
A	C	A	-	0	2
C	E	D	-	3	1
C	E	C	-	3	2

IRT (b)

Table 5.1: Example of DRT and IRT tables.

Each path defines an entry on the table for every visited switch. These entries are consecutively allocated in the table in the same order switches are visited, starting from the source switch (switches with smaller number of hops from the source switch appear first). For each entry, the table contains the source and destination switches, the input link ID used at the reachable switch, the number of hops needed to arrive from the source to the reachable switch, and the number of transitions (if any) of virtual channels required along the path to arrive from the source to the switch (initially none).

The second table (Table 5.1.b), referred to as indirect reachability table (IRT), is similarly defined, but the info is acquired in the opposite order, i.e., starting from the destination switch. In the same sense, it contains the source and destination switches, the output link used at the reachable switch, the number of hops needed to arrive to destination from the reachable switch, and the number of transitions (if any) of virtual channels required along the path to arrive from the switch to the destination (initially none).

5.2.2 RFTR Methodology

The description of RFTR is done through a complete example, showing all the steps performed by the methodology in the presence of a fault in a 3×3 mesh network. The example and all the steps followed by RFTR are shown in Figures 5.3 and 5.4.

For the sake of explanation and ease of understanding, we only consider existing paths between few sources and destinations. In particular, we consider paths from end nodes attached to switch B to end nodes attached to switches G, H, and I, and from the end nodes attached to switch G to end nodes attached to switch I. In this way, reachability tables in the example can be small.

Initially, the paths for every source-destination pair are computed by applying a certain underlying deterministic routing algorithm. Then, the method computes the DRT and IRT tables from the computed paths. Figure 5.3.a shows four initial paths and the tables obtained. Whenever the notification of a new fault arrives to the SM (Subnet Manager) the method is triggered. Figure 5.3.b shows the localization of a link failure. The SM entity is placed on one or more end nodes, and is responsible to manage the network, that is, for computing the routing algorithm and applying this methodology. This entity stores DRT and IRT tables in local memory [2].

The mechanism requires an efficient way to identify the location of the fault within the network. This is achieved by using the ID of the switch attached to the failed link and the label of the port that has failed. The method tolerates switch and link failures. In the case of a switch failure, the switch ID will be provided. In the case of a link failure, the link label and the ID of one switch attached to the failed link will be provided.

The main goal of the methodology is to compute, in a time-efficient manner, alternative paths for every pair of source-destination end nodes whose path have been involved in a fault. Thus, as a first step, RFTR identifies the paths affected by the fault. This is achieved by sweeping both tables (DRT and IRT). Whenever the failed link and/or switch is found in an entry (the failed switch appears as reachable and the failed link appears as input link for DRT or output link for IRT), the associated path is annotated as failed. Figure 5.3.c shows the entries (affected by the fault) found in the (DRT and IRT) reachability tables and the obtained list of faulty paths. Notice that the list of faulty paths is very small compared with the total number of paths.

All the table entries affected by the failed paths are removed in both DRT and IRT tables (they are no longer reachable using the path). The entries to be removed are shown in the example as yellow shadowed in Figures 5.3.c and 5.3.d.

Once a given path is annotated as failed, RFTR builds two sets of switches. The first one is compounded by the switches directly reachable from the source of the affected path, and the second one by the switches indirectly reachable from the destination of the affected path. In order to build the set of directly reachable switches, the method sequentially sweeps the DRT table searching for directly reachable switches from the source. The set includes only the indexes to the DRT table (there is no need to store a copy of the entries of the table). The same is done for the IRT table in order to build the set of indirectly reachable switches. Note that a switch may appear in any set several times (replicas) as it can be reached by the source (or indirectly reached by the destination) through different paths. Also, each replica may require different number of virtual channel transitions[3] and may be at a different distance from the source (or destination). The method keeps the replicas on each set in order to get the

[2]The memory requirements are in the same order as the memory requirements of the routing tables.

[3]Initially, paths have no virtual channel transitions. They may appear later as long as the methodology computes new paths to cope with faults.

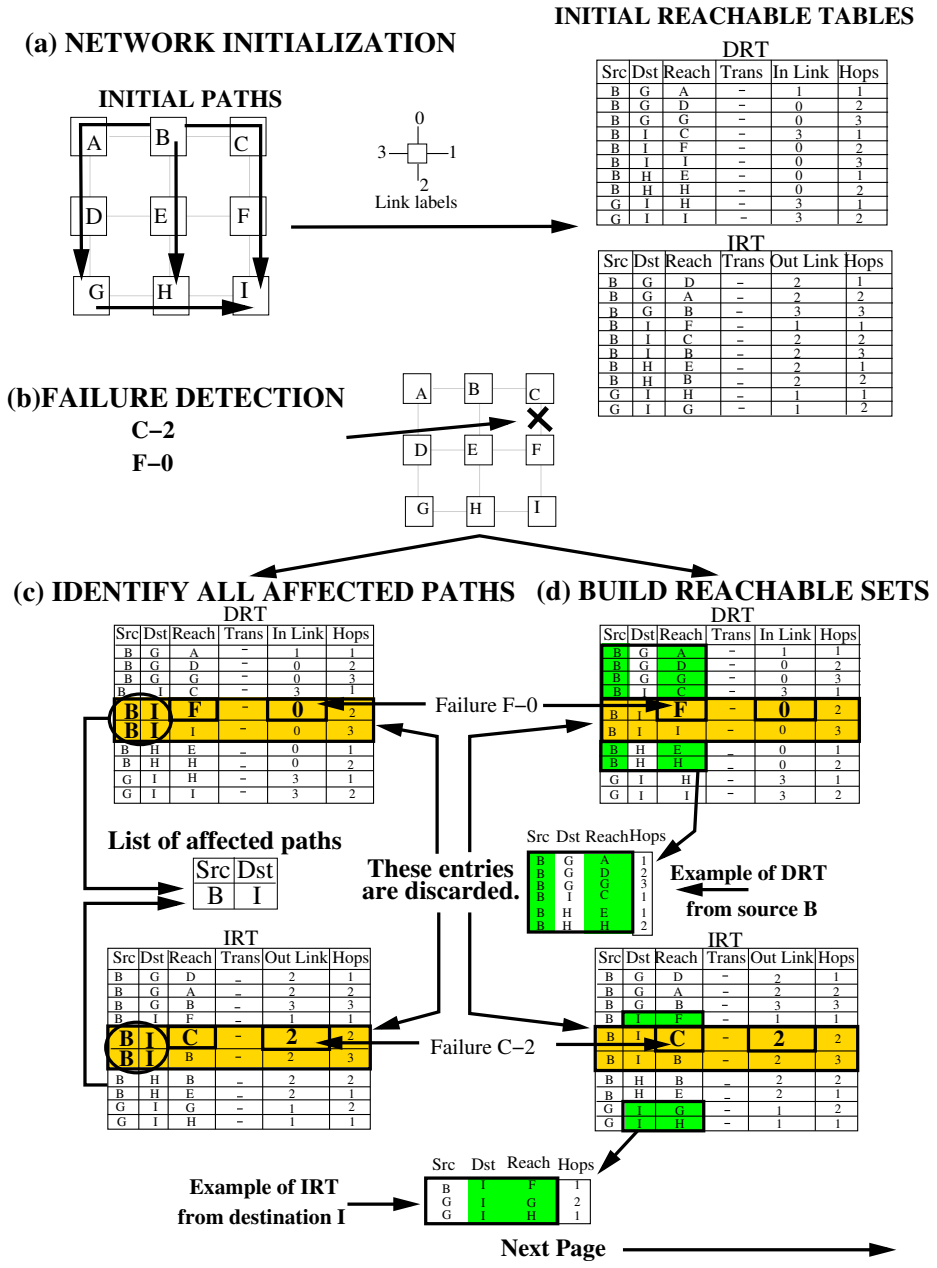


Figure 5.3: Applying RFTR for one link failure in a 3×3 mesh (part 1).

best option when computing the final path. Figure 5.3.d shows the directly reachable switches from switch B and indirectly reachable switches from switch I.

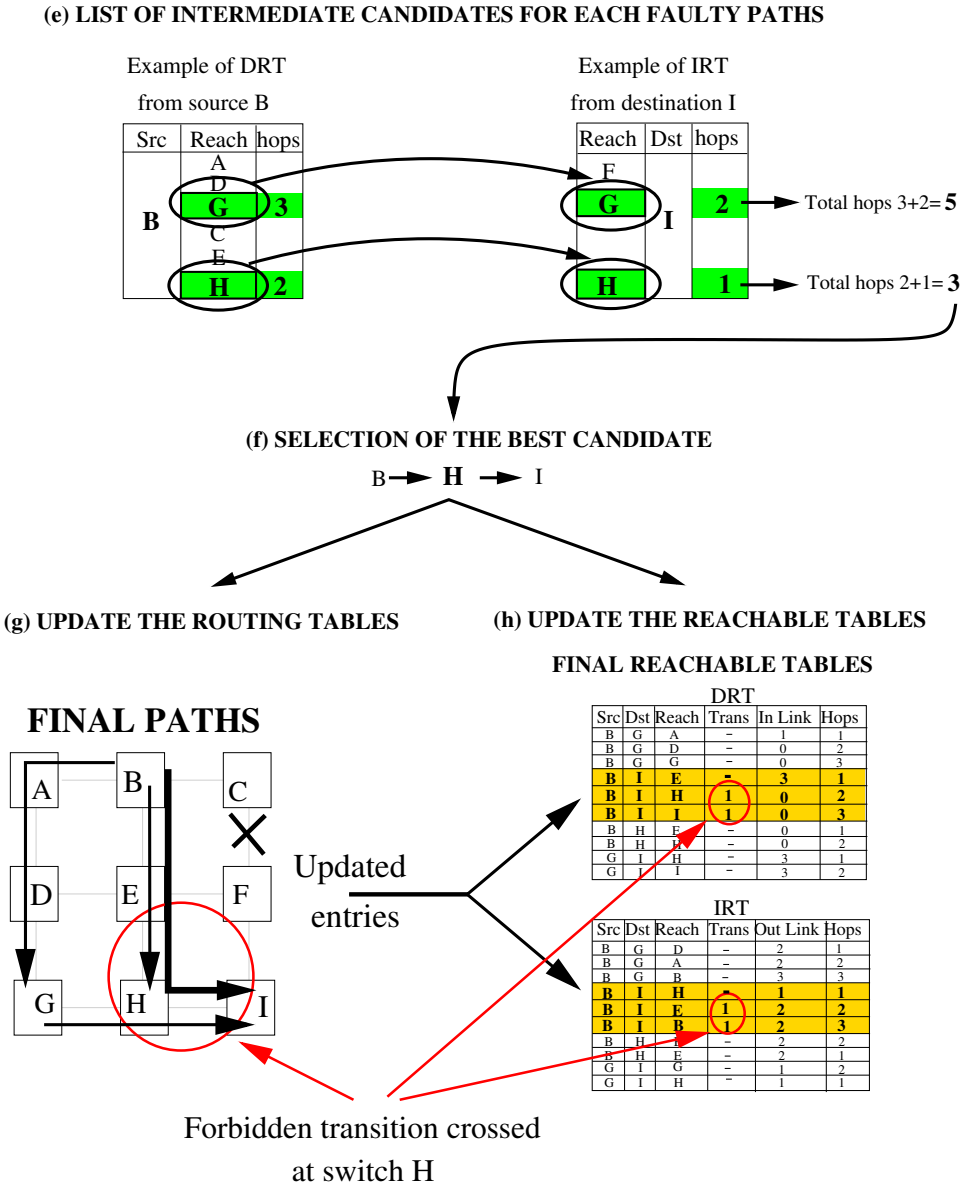


Figure 5.4: Applying RFTR for one link failure in a 3×3 mesh (part 2).

Remember that the aim of the methodology is to compute a new valid path by joining two already computed subpaths. The join point (switch) of both subpaths will be referred to as intermediate switch. Therefore, once sets are computed for a given affected path, the method will select the appropriate intermediate switch from the intersection of both sets. The new path will consist of the union of two subpaths,

one from the source to the intermediate switch, and the other from the intermediate switch to destination.

The method uses the following criteria to select the intermediate switch: the final path will have the minimum number of virtual channel transitions and, in case of a tie, the shortest path will be selected. In order to compute the number of required virtual channel transitions, the method takes into account the number of virtual channel transitions already used in each subpath (extracted from both tables) and whether a virtual channel transition is required (according to the routing algorithm applied). To compute this, the method needs to know the input and output ports used at the intermediate switch. This info is found in the DRT and IRT tables. Figure 5.4.e shows the set of directly reachable switches from switch B and the set of indirectly reachable switches from switch I. The common reachable switches from both sets are highlighted. Also, the figure shows the number of hops to reach the intermediate switch and the total number of hops for the possible final path. In Figure 5.4.e two candidates to become intermediate switch are considered: G and H. The selected one is H, with a total of three hops (G requires a total of five hops). Notice that the selection of the best candidate does not require to sort the candidate list.

Once a new path is obtained the new reachability info provided by the new path is added to DRT and IRT. Once the new paths are computed, the new routing info must be distributed to switches and end nodes. Figure 5.4.g shows the final paths obtained once RFTR is applied to the example. However, how info is distributed and routing tables are updated depends on the RFTR implementation and the network technology used. Finally, it has to be noted that once all the new paths are computed, DRT and IRT tables are ready to be used in the presence of a new switch or link failure. Figure 5.4.h shows the new reachability tables which have new entries corresponding to the new path.

5.3 RFTR Complexity

The computational cost is an important aspect of the proposed fault tolerant routing mechanism, since during that time a certain fraction of traffic is lost due to the fault. Also, the main goal of RFTR is to minimize its computation time.

In this section we analyze the cost of RFTR. This involves the amount of time from the fault detection until the complete routing solution is obtained. We do not consider here the time to distribute the new routing info, since this is technology dependent. To help in the description of the cost, we refer to Figures 5.3 and 5.4.

The main parameters RFTR depends on are the number of end nodes in the network (N), the average path length (L) and the number of faulty links in the network (F). DRT and IRT tables require each one N^2L entries.

RFTR, after the detection of the fault, identifies the faulty paths and builds the reachable sets. These stages are shown in Figures 5.3.c and 5.3.d, respectively. Each one of these tasks is done by one sweep of both tables (DRT and IRT), each one having a complexity of $O(N^2L)$. As an optimization, notice that both sweeps (the one for detecting affected paths and the one for computing the sets) can be done at the same time (only one sweep is needed). The remaining stages require a lower computation time since they operate over a reduced set of intermediate switch candidates.

This cost can be further reduced by using more memory structures at the SM. For instance, linked lists of pointers can be used for each link and each source-destination pair, thus, only entries affected by the faults are searched. This reduces the cost down to $O(FL)$. Anyway, in Chapter 8 we provide measurements of a real implementation.

5.4 RFTR on InfiniBand

In the previous section we described the methodology independently of the architecture of the network. As an example of applicability, in this section we adapt the RFTR methodology to InfiniBand (IBA). To do this, we describe how faults are detected and managed in IBA, and how the entire process can be integrated with RFTR.

Figure 5.5 shows the steps followed by the methodology once integrated in IBA. At the first stage SM carries out sweeps in order to detect changes or faults in the subnet. When the SM encounters a fault, it triggers the RFTR methodology. The method works as described in section 5.2.2. However, as SLs are used in InfiniBand, a new criteria for selecting the appropriate intermediate switch is needed. In particular, higher priority is given to those paths not introducing a mapping conflict (thus not requiring an additional SL). In case of a tie, the switch that leads to a shorter final path is preferred.

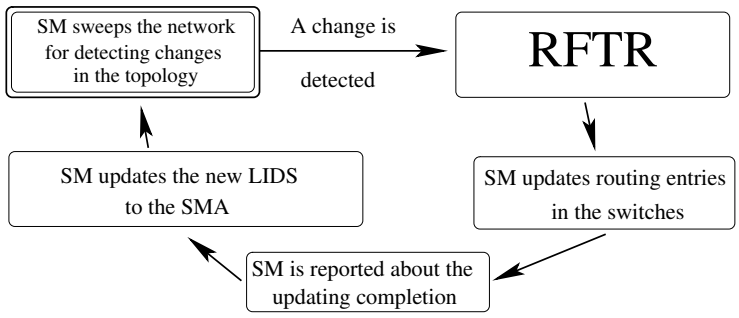


Figure 5.5: Stages followed by RFTR on IBA.

In order to differentiate both paths (the new one and the failed one) from the subnet point of view, the methodology assigns a new virtual address (LID) to each

new path. Once all the new paths are computed, SM sends the new routing info (only the additional info for the routing tables and SLtoVL tables) to the corresponding switches, and the new LIDs to the end nodes. Notice that the size of the information sent will be different for each switch or end node (even some switches will do not receive new routing information at all).

To send the new control information a special routing mechanism (*Directed-Route*) is used. According to the IBA specs, *Directed-Route* is used for routing control packets through VL15. The entire path of each packet is established at the header by specifying all the output ports along the switches to be crossed. As it defines the routing paths port by port, the fault will be avoided by the SM.

When SM has finished sending the new routing info, it waits until the reception of an acknowledgment from all the switches whose tables have been updated. This is required in order to ensure that messages can be appropriately routed when using the new LIDs. Otherwise, they might be discarded. Upon reception of acknowledgment from every updated switch, SM sends to the SMAs placed in all the end nodes the new LIDs corresponding to the new paths. Upon reception, the new injected data packets will use the new LIDs, thus being appropriately routed by the switches according to the new routing info.

Please, note that the underlying routing algorithm is not modified, only some paths are modified. When the new paths perform a transition between virtual networks when they require to cross a routing restriction. This is the reason of the required resources by the mechanism.

This page intentionally left blank

Chapter 6

Fault-Tolerance Methodology Based on Reconfiguration

In the previous Chapter we have proposed a local reconfiguration mechanism able to tolerate most fault combinations. The proposed mechanism, however, relies on the original routing algorithm for computing new paths (using virtual channel transitions if required). This may lead to achieve drops in performance as faults accumulate and the topology changes. In this situation a global reconfiguration may be required, thus using a completely different routing algorithm. In this Chapter we propose a new global dynamic reconfiguration mechanism to achieve such purposes.

6.1 Motivation

The main challenge of a dynamic reconfiguration process is to avoid deadlock anomalies. In a global reconfiguration process, the system migrates from a routing algorithm (R_{old}) to a new one (R_{new}). Although both routing algorithms are deadlock-free when isolated, during the reconfiguration process both algorithms coexist and they may induce cyclic dependencies between packets (old packets and new packets). As a second challenge, dynamic reconfiguration processes must reduce packet dropping rate while keeping restrictions on packet injection and forwarding minimal. Current approaches (described in section 2.1.8.2) fall in one of the next categories. Either they require the existence of extra network resources like e.g., virtual channels, their complexity is so high that their practical applicability is limited, or they severely impact on the performance during the reconfiguration process.

In particular, PPR [52] only works between two routing functions that adhere to the *up*/down** scheme. NetRec [120] is specially tailored for re-routing messages around a faulty node. It basically provides a protocol for generating a tree that

connects all the nodes that are neighbors to a fault, and drops packets to avoid deadlocks in the reconfiguration phase. The DoubleScheme [124] is more flexible, in the sense that it can handle any topology and perform a transition between any pair of deadlock-free routing functions. However, it requires the presence of two sets of virtual channels. The methodology proposed in [106] requires complex computation in order to derive a safe reconfiguration process once the new routing function has been decided. This will take time, and thus limits the practical applicability of the methodology. Regarding the mechanism proposed for InfiniBand [48], it discards packets in order to avoid deadlocks. Finally, the SR mechanism [106] requires a token to be distributed over the entire network. Although it separates old and new traffic, it has two important drawbacks. The first one is that its implementation is not straightforward. The token distribution is based on the dependencies of the old routing function. Also, the mechanism is suitable for routing functions that consider the input port for routing ($C \times N \rightarrow C$, provided the input channel C and the destination ID N the routing function provides the output channel C). For routing mechanisms based only on the destination ($N \times N \rightarrow C$) the SR mechanism is not directly applicable. The second drawback is that inherently adds some extra blocking to packets since new packets must wait for the tokens to advance and these tokens have many dependencies. This is analyzed on Chapter 8.

To overcome these problems we present in this chapter a simple and powerful method for dynamic network reconfiguration. The mechanism, referred to as *Epoch-Based Reconfiguration* (EBR), guarantees a fast and deadlock-free transition from the R_{old} to the R_{new} routing function, but instead of avoiding deadlocks the mechanism is based on regressive deadlock recoveries. Thus, EBR allows cycles to be formed, and in the situation of a deadlock some packets may be dropped. However, only in high traffic conditions beyond congestion packets need to be dropped to recover from deadlocks. Therefore, no packets need to be dropped in the working zone of the system. Also, the mechanism works in an asynchronous manner as switches start using the new tables as soon as they are received. By doing this, EBR greatly reduces the dropping rate of packets due to faults in the network, when compared to previous dynamic reconfiguration proposals.

EBR does not require additional resources and works on any topology using the virtual cut-through or store-and-forward switching mechanisms (as in InfiniBand). Also, EBR works with any distributed routing function based on the destination ID of packets for routing ($N \times N \rightarrow C$). For minimizing the number of dropped packets, EBR uses an *epoch* marking system that guarantees that only packets potentially leading to a deadlock will be removed, and only during the reconfiguration process. The previous contributions described in previous chapters can be applied in IBA without any modification. However, EBR can not be directly applied to IBA since the implementation of the deadlock recovery mechanism requires some hardware modification at switches, although this modification is small.

6.2 Basic EBR Description

We consider the use of two routing algorithms, that we refer to as R_{old} and R_{new}. Both are separately deadlock-free and are based on the destination IDs ($N \times N \rightarrow C$) for routing. Routing information is distributed along the switches by using routing tables. Also, we do not consider data virtual channels.

The mechanism is based on the fact that deadlock situations are rare. Indeed, the probability of deadlock is proportional to the traffic injection rate and inversely proportional to the availability of resources. Assuming that the network works on a non-congested scenario, the probability of deadlock is minimum. Notice that in small and medium traffic loads the contention levels within the network is low, thus, packets roughly leave queues at the same rate they reach input ports. Therefore, on average, there is at maximum only one packet at each queue at any time.

The EBR mechanism will allow cycles to be formed (see Figure 6.1). A deadlock is potentially formed when a packet can not advance due to a cycle occurrence. Such a cycle can be only formed when different switches route packets with different routing algorithms (R_{old} and R_{new}) at the same time. For example, Figure 6.1 shows a cycle formed in a reconfiguration process. It is formed since there is a cyclic dependence among all the packets/queues. Notice that a packet routed with the new routing function is waiting (in the cycle) behind a packet routed with the old routing function, and in the same cycle a packet routed with the old routing function is waiting behind a packet routed with the new routing function.

However, notice that if queues involved in the cycle have space for the packets to advance, the network will not deadlock. Also, if we assume the use of virtual cut-through switching the packet located at switch 1 (corresponds to a packet routed with the old routing function) can also be forwarded even if it is using a forbidden transition defined in R_{new}. Indeed, once forwarded, the cycle disappears.

In the case queues are full, however, packets can not advance and, thus, a deadlock occurs. To solve this situation, the mechanism will discard some packets in order to remove the deadlock. Notice that the EBR is proposed as a reconfiguration mechanism for tolerating faults and reducing the traffic lost due to faults during the process, and although EBR may drop some packets it reduces the total number of lost packets in fault scenarios. Notice also that retransmission mechanisms are still needed in networks with fault scenarios, thus dropped packets by EBR (if any) will be retransmitted.

Packets to be discarded must be carefully selected in order to minimize the dropping rate. Provided that virtual cut-through is used, it can easily be demonstrated that in a cycle forming a deadlock configuration there is at least one packet at the head of one input queue introducing a dependence not defined in the new routing function. Figure 6.1 shows the case. Therefore, the mechanism must discard packets introducing dependences not defined in the new routing only when they are at the head of a

queue and can not advance (no credits are available). In any other case, the mechanism does not drop any packet (the mechanism potentially may drop packets only during a reconfiguration process).

To accurately select the packets that need to be dropped, an *epoch* marking system is used. With this system every switch discards only packets that could be causing a deadlock.

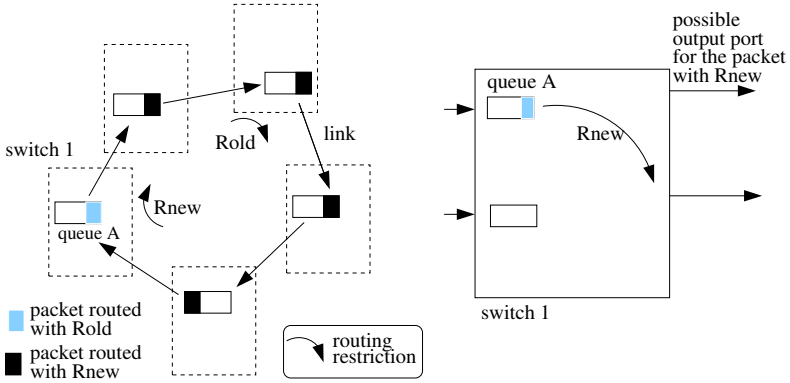


Figure 6.1: Cycle in a dynamic reconfiguration not introducing a deadlock.

For the sake of explanation, EBR is described as a fully deployed mechanism following the entire reconfiguration process, from the occurrence of the topological change (i.e., a fault) to the normal and final functioning of the network with the new routing algorithm. Figure 6.2 summarizes all the steps followed in the EBR reconfiguration process at the different network components. The following sections describe each step.

6.3 Triggering the EBR Mechanism

EBR is applied whenever a new routing algorithm is required for the network. In some situations the change in the topology does not necessarily lead to a change in the routing algorithm. This is the case of adding/removing end nodes to/from the system. As the routing algorithm is not changed there is no probability of deadlock, thus not requiring a global reconfiguration process. On the other hand, it may happen that a new switch or link is added, thus changing the topology. In that case, a change of the routing algorithm could lead to achieve higher performance thus triggering a reconfiguration process. Also, it may happen that the routing algorithm needs to be changed even if no topological change occurs. For instance, in those cases in which changing the routing algorithm leads to achieve higher performance.

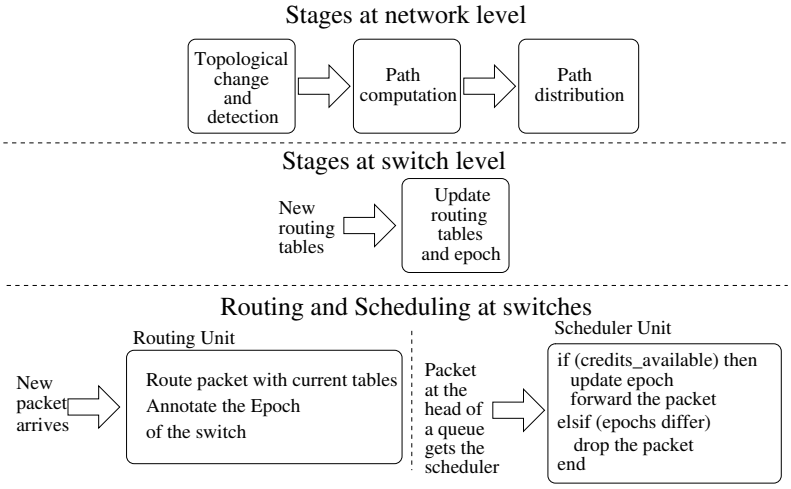


Figure 6.2: Stages of the EBR mechanism.

However, the most interesting topological change is when a switch or link fails (or a group of them). In this case it is required to change the routing algorithm, as some parts of the network may be disconnected and packets will be dropped through the failed component. For the sake of explanation, we will describe the mechanism only in the situation that a fail occurs.

In all these cases the EBR reconfiguration mechanism is triggered. To do so, a selected end node runs an instance of the Network Manager (NM) in charge of detecting any topological change. To do this, NM sends control packets periodically to all the switches. Switches reply with the current status of their links and neighbor components (switches and end nodes). With this provided information, NM builds the current topology. If a change is detected, the EBR reconfiguration process is triggered.

To send the new control information the *Directed-Route* mechanism provided in IBA is used. How control packets are sent is implementation specific.

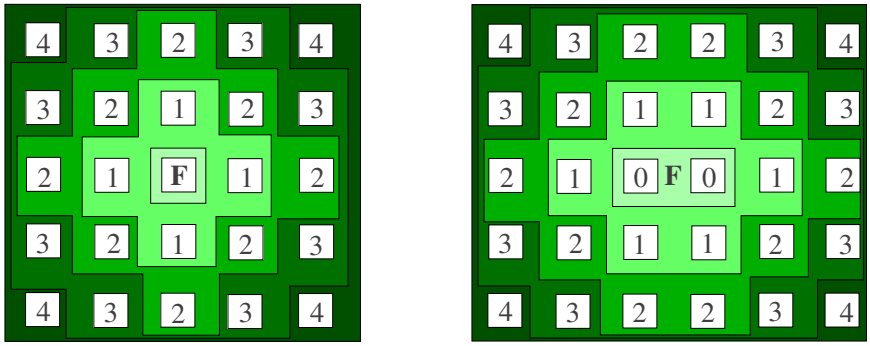
6.4 Routing Path Computation

Once the topological change has been detected new routing tables need to be computed. Usually this stage is the dominant of the entire process (from fault occurrence to normal operation). The computation time required depends on many factors like the complexity of the routing algorithm, the processing capacity of the computer being used, and so on. During the computation phase it is reasonable to think that packets are continuously being injected and therefore some of them are dropped through the failed component at a constant rate. Indeed, the number of dropped packets may be proportional to the computation time.

Most of the proposed reconfiguration mechanisms have focused on the table update process, thus they do not consider the computation phase. However, different techniques may be applied to alleviate the negative effects of this phase. As previously proposed in this thesis, a solution is to provide several disjoint paths per each source-destination pair. Therefore, once a topological change is detected, the end nodes are notified (by the NM) and from that moment they use the alternate paths that avoid the failed link. Therefore, during the long computation phase packets are not dropped. However, notice that using alternate paths is an additional mechanism not related with the reconfiguration process. We focus, thus, only in the reconfiguration process (table updates).

6.5 Path Distribution

Once the paths are computed, they must be distributed. To do this, the NM sends all the new routing tables to the switches in the network. In order to reduce the traffic lost through the failed components, the new routing info is distributed to the switches in an ordered way, starting at switches surrounding the failed components. Figure 6.3 shows two examples of the sequence used to update the switches.



(a) Failure at Switch F (b) Failure at Link F

Figure 6.3: Sequence of routing table updates. The numbers indicate the sequence and the distance to the failed component.

Additionally, in order to reduce the overhead control traffic, the NM may send only the differences between the old and the new routing tables to every switch. Depending on the similarity of the routing algorithms the percentage of control traffic reduction may be significant. Indeed, we will see in the evaluation that this improvement will affect the effectiveness of the mechanism.

Once a switch receives the new paths, it updates its routing table, removing the old one. Therefore, from that moment packets will not be routed with the old routing

function anymore. Notice that packets already routed and mapped into queues at the switch will not be routed again.

Figure 6.4 shows an example of the updating process in a switch. Figure 6.4.a shows the status of the switch before updating the routing table. Some packets have been routed with the R_{old} routing algorithm (red packets) where as other packets have not yet been routed (yellow packets). Figure 6.4.b shows the same switch once the routing info has been updated. As can be noticed, previous packets routed with R_{old} are not routed again. New packets being routed, however, use the new routing table, thus they are routed with R_{new} (green packets).

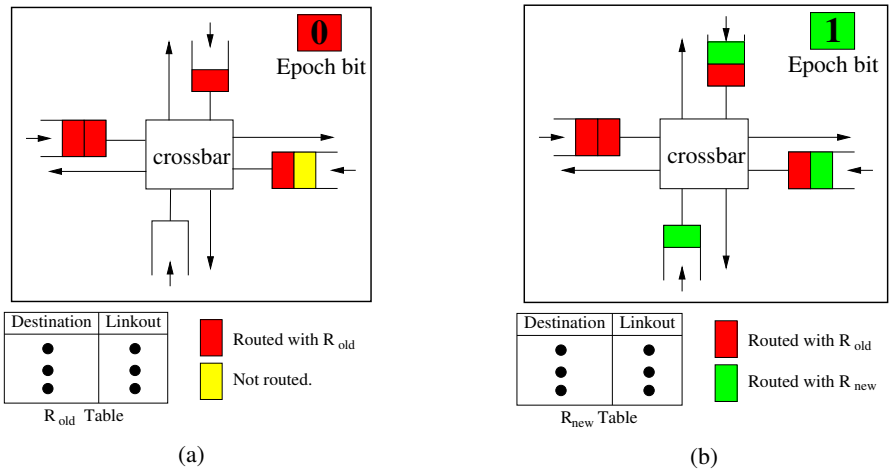


Figure 6.4: Example of a switch status: (a) before and (b) after updating the routing info.

6.6 Deadlock Recovery Process

As may be noticed, updating switches asynchronously may lead to introduce cycles in the network. These cycles are made of *old* packets and *new* packets.

In order to keep track of packets that potentially may introduce cycles, EBR uses a novel *epoch* marking system. An epoch consists of an ID (typically one bit) that represents the routing algorithm used to route a packet (e.g., a zero for the *old* routing function and a one for the *new* routing function). In this sense, every switch is labelled with the epoch bit corresponding to the routing algorithm applied by its routing tables (E_{SCHD}). Figure 6.4 shows the epoch bit. Upon reception of a new routing table the epoch bit of the switch is changed accordingly.

Packets are labeled also with the epoch bit (E_{PCK}). Each end node has an epoch bit, which is used for labeling each packet injected into the network. On every visited

switch, once successfully forwarded to the next switch, the epoch bit of the packet is updated with the epoch of the routing algorithm used for forwarding the packet (the epoch bit of the switch; E_{SCHD}). This means that a packet's epoch may be changed several times as it crosses different switches with different epochs.

One of the problems EBR must face is that the process of routing and packet forwarding are not usually performed at the same time, thus when routing a packet it is unknown if the packet will have credits at the next switch or end node. To solve this problem, EBR routes packets regardless of whether they later have credits or not. Later, when the packet is being scheduled, EBR drops the packet only if there are no credits available and the packet could lead to deadlock.

Whenever a new packet arrives to a switch the packet is routed and an internal epoch is associated with the packet (E_{RTD}). The internal epoch (E_{RTD}) is set with the same value of the epoch of the switch at that moment (at routing time). Once the packet reaches the head of its queue and is scheduled, the EBR deadlock recovery mechanism comes into action. Table 6.1 shows all the possible combinations that may occur when scheduling a packet. Table shows the epoch of the packet (E_{PCK}), the epoch of the switch at the time the packet was routed (E_{RTD}) and the epoch of the switch at the current time the packet is being scheduled (E_{SCHD}). The table shows whether the deadlock recovery mechanism discards or not a packet when it can not be forwarded (e.g., has not credits available). If, on the contrary, the packet has credits it is simply forwarded.

Case	E_{PCK}	E_{RTD}	E_{SCHD}	Discard packet
#1	0	0	0	No
#2	**0**	**0**	**1**	**Yes**
#3	0	1	0	Impossible case
#4	**0**	**1**	**1**	**Yes**
#5	1	0	0	No, transient
#6	**1**	**0**	**1**	**Yes**
#7	1	1	0	Impossible case
#8	1	1	1	No

Table 6.1: Actions performed by EBR (when no credits are available) based on the packet (E_{PCK}) and the switch (E_{RTD} and E_{SCHD}) epoch bits. (No overlapping is supported).

The first combination $E_{PCK} = E_{RTD} = E_{SCHD} = 0$ corresponds to an *old* packet that was routed with the old routing function and is being scheduled. In that situation the packet is not discarded. Simply it will be scheduled the next arbitration cycle. Notice that this packet in a reconfiguration process could introduce a deadlock along a cycle with packets routed with the new routing function. However, if this is the case, the packet will finally be discarded as the switch will receive the new

routing function and the epoch of the switch will change (E_{SCHD}), thus migrating to the second entry on Table 6.1. In that situation, the packet is discarded (if no credits are available).

Entries #3 and #7 have no meaning since they represent the case when the switch downgrades from a new routing function to an old routing function. In cases #4 and #6 packets will be discarded (if no credits are available). In case #4 it may happen that an *old* packet is using a forbidden transition in the new routing algorithm (it has been routed with R_{new} but could cross the switch through a transition between ports forbidden by R_{new}) and therefore it could introduce a deadlock.

In case #6 a special case arises: a *new* packet is routed with the old routing function but when being scheduled the routing table has been updated ($R_{SCHD} = 1$). In that situation the packet must be discarded (if no credits are available). Figure 6.5.a shows four switches in the network during the reconfiguration process. Switch D has its routing table and epoch label updated. Switch C routes packets with R_{old}, although they have been routed with R_{new} at the previous switch. Figure 6.5.b shows the case once table at switch C has been updated. Packets mapped at switch C are routed with R_{old}, although the switch has just updated its routing table to R_{new}. Notice, that the packet which was routed at the previous switch with R_{new}, keeps routed at this switch with R_{old} even if the switch has been updated (packets are not rerouted after updating the routing tables). In this case, if no credits are available, the packet will be discard at scheduling time.

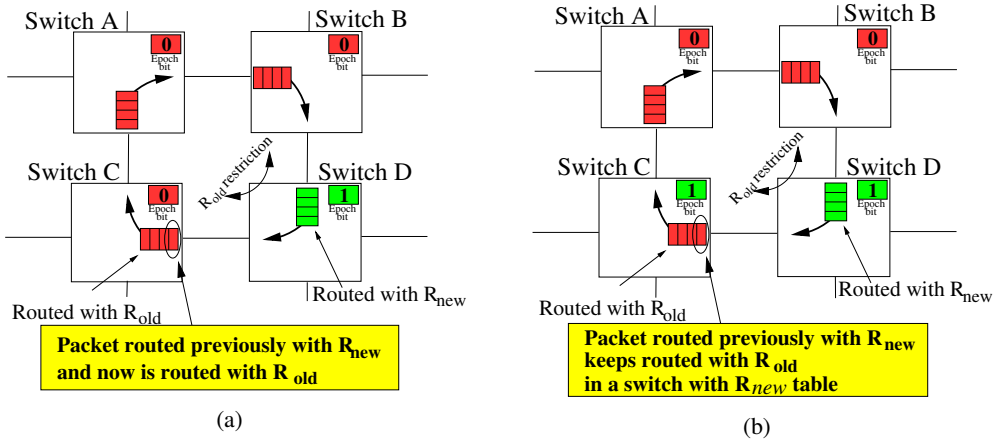


Figure 6.5: Example of routing and scheduling during the reconfiguration process at switch C (a) before being updated (*old* and (b) after being updated (*new*).

Case #5 is a transient state that will evolve to case #6, thus there is no need to drop any packet in that situation. This is because we assume that control packets will be reinjected and safely source routed with the new routing info. Finally the last case (#8) corresponds to a new packet being routed with the new routing function. Thus, the packet is not dropped.

Notice that only packets routed and scheduled with different epochs can be discarded (provided no credits are available), thus only during a reconfiguration process packets may be discarded leading to remove the deadlock.

The routing unit at switches is not modified. Simply, when the new paths (routing tables) are available to the switch they are used for the incoming packets. Once a packet is routed, the obtained output port and the current epoch of the switch (E_{RTD}) are attached to the packet until the packet reaches the head of its queue, thus contending for the requested output port.

However, the scheduler needs to behave differently. In particular, the packet will be arbitrated and on success (credits are available) the packet will be forwarded, even if the packet is using a forbidden transition defined in the new routing algorithm. On the contrary, if there are no credits for the packet at the requested output port, then the switch potentially may drop the packet. To do this, the epoch of the packet is compared with the current epoch of the switch and the epoch of the switch when the packet was routed (E_{RTD}). In some cases the packet is simply discarded (as depicted in Table 6.1). If not, the packet will be arbitrated the next arbitration cycle. A simple circuit implementing all the possible cases is shown in Figure 6.6.a.

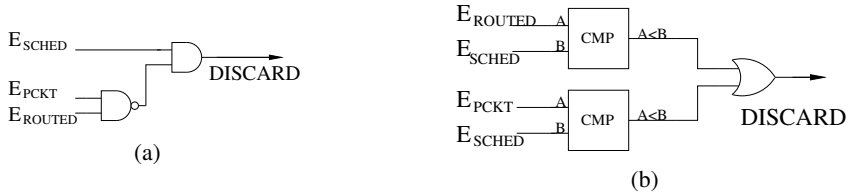


Figure 6.6: Logic implementing the EBR deadlock recovery mechanism, (a) simple circuit, (b) extended version.

6.7 Overlapping Multiple Reconfiguration Processes

The mechanism can be easily extended to support different overlapping reconfiguration processes. Simply the epoch field at packets, switches, and end nodes, must be extended. With two bits up to four reconfiguration processes can be overlapped. In this situation, EBR allows the overlapping of different reconfiguration processes.

That is, a new reconfiguration process may start before the previous once is still underway. Even an intermediate reconfiguration process can be aborted with no harm of deadlocks or inconsistencies.

When overlapping multiple reconfiguration processes with an extended epoch field new scenarios are possible. Therefore, Table 6.1 must be extended to consider the new possible cases. Similarly to the non overlapping case, a packet is dropped only if the packet at the header of the queue (being scheduled) has no credits, and the epoch of the packet (E_{PCK}) or the epoch used when it was routed is lower than the current epoch of the switch (E_{SCHD}). Figure 6.6.b shows a possible implementation with two comparators and an OR gate. Table 6.2 shows all the possible situations when using up to 3 different routing algorithms (epochs 0, 1, and 2). The actions described in the table are performed only if the packet being scheduled has no credits available.

Case	E_{PCK}	E_{RTD}	E_{SCHD}	Discard packet
#1	0	0	0	No
#2	**0**	**0**	**1**	**Yes**
#3	**0**	**0**	**2**	**Yes**
#4	0	1	0	Impossible case
#5	**0**	**1**	**1**	**Yes**
#6	**0**	**1**	**2**	**Yes**
#7	0	2	0	Impossible case
#8	0	2	1	Impossible case
#9	**0**	**2**	**2**	**Yes**
#10	1	0	0	No, transient
#11	**1**	**0**	**1**	**Yes**
#12	**1**	**0**	**2**	**Yes**
#13	1	1	0	Impossible case
#14	1	1	1	No
#15	**1**	**1**	**2**	**Yes**
#16	1	2	0	Impossible case
#17	1	2	1	Impossible case
#18	**1**	**2**	**2**	**Yes**
#19	2	0	0	No, transient
#20	2	0	1	No, transient
#21	**2**	**0**	**2**	**Yes**
#22	2	1	0	Impossible case
#23	2	1	1	No, transient
#24	**2**	**1**	**2**	**Yes**
#25	2	2	0	Impossible case
#26	2	2	1	Impossible case
#27	2	2	2	No

Table 6.2: Actions performed by EBR (when no credits are available) based on the packet (E_{PCK}) and the switch (E_{RTD} and E_{SCHD}) epoch bits. Overlapping is supported.

Notice that cases #19 and #20 are transients situations as it is expected that the new outing table (epoch 2) reaches the switch. Thus, both cases evolve to case #21, eve if reconfiguration associated to epoch 1 is aborted (case #20 does not happen).

Chapter 7

Evaluation Model

In this chapter we provide all the details of the evaluation model we have chosen. In particular, we describe the network simulation model, the traffic patterns, and the topologies used for evaluating the proposed mechanisms.

First, we introduce the concept of modeling a system with simulation tools. Later, we describe the simulator and the parameters used in this thesis. Once the simulation tool is presented, we describe the scenarios modeled. Finally, at the end of the chapter we describe how results are collected and the main performance metrics used.

7.1 Introduction to Simulation Modeling

The performance of a system can be evaluated in different ways [91]:

- **Evaluating a real system:** With this method the measures are obtained from a real system. Thus, it requires the real system to be available, which in most cases is expensive or even not feasible.
- **Evaluating a simulated system:** This is the most popular way for many researchers. With a simulation tool it is possible to accurately evaluate any system. However, the main problem of this method is that the required time is larger than in real systems.
- **Evaluating with analytical models:** This way becomes the cheapest method. It can provide results in a short period of time, although the evaluated model must be simplified. Therefore, this way provides the less accurate results. In this dissertation we don't use it since we are interested in a high detail model.

The evaluation of the contributions of this thesis is done with an ad-hoc simulator. The main reason for this choice is because we evaluate different methodologies

applied to different scenarios. A proper simulation tool allows many configurations to be evaluated. Also, given the current available processing power in the research group, many evaluations can be searched in parallel thus speeding up the research.

7.1.1 Simulation Tool

As a starting point we have used one of the in-house simulators developed by the research group. In particular, the simulator models an IBA network at clock cycle level, following most of the specifications of IBA [89]. In order to evaluate the different proposals, we have upgraded the simulator with new functionalities.

We define the *state* of a system as the collection of all the variables necessary to describe the system at a particular time, relative to the parameters and objectives of a study. Depending on the *state variables*, systems to be simulated can be categorized into two types: discrete and continuous.

The simulator models the network as a *discrete system*, because the network state variables change only at a countable number of points in time. The system is defined as a finite number of components, like end nodes, switches, links, and buffers. For example, a switch is a discrete system since its state variables, e.g., the free buffers in their queues change only when a flit is received or when a flit is sent. Instead, in a *continuous system* the state variables change continuously with respect to time. Examples of variables in a continuous system are temperature, position or speed.

Discrete event-driven simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change only at a countable number of points along the time. These points are the ones at which an *event* occurs, where an *event* is defined as an instantaneous occurrence which may change the state of the system. Although a discrete-event simulation could conceptually be done by hand, the amount of data that must be stored and manipulated for most real-world systems dictates those discrete-event simulations to be done on a computer.

Although simulation has been applied to a great diversity of systems, all of them share a number of common components and there is a common logical organization for these components. In particular, the following components will be found in most discrete event-driven simulation models:

- *System state*: Represents the collection of state variables necessary to describe the system at a particular time.
- *Initialization routine*: Initialize the simulation model and the initial events at time zero.
- *Simulation clock*: It provides the current value of simulated time.
- *Event list*: It contains the events pending for processing.

- *Statistics*: Stores statistical information about system performance.
- *Event routine*: Updates the system state when a particular type of event occurs (typically there is one event routine for each event type).
- *Timing routine*: Determines the next event from the event list and advances the simulation clock to the time when that event should happen.
- *Main program*: Calls the timing routine to determine the next event and transfers control to the corresponding routine to run the current event and update the system state appropriately.
- *Report generator*: Computes estimates (from the statistical counters) of the desired measures of performance and prints a report when the simulation ends.

Figure 7.1 shows the basic structure of a discrete event-driven simulation model. The simulation begins with the initialization routine, where the simulation clock is set to zero, the system sate and the statistical counters are reset, and the event list is initialized. Then, the timing routine is called to determine the next event to process. As an example, we consider the next one is event i. The simulation clock advances to the time that event i will occur. Then, the event routine i is called, where typically three types of activities occur: (1) updating the system state to account for the fact that an event of type i has occurred, (2) gathering information about system performance by updating the statistical counters, and (3) computing the future events due to this event (this information is added to the event list). After processing the event, the simulator checks if the simulation process has finished (reaching a time stamp or a certain number of processed messages. If so, the statistics are dumped and the process finishes.

7.1.2 Advantages and Disadvantages

Using simulation tools may exhibit benefits but also inconveniences. A list of benefits follows:

- Most complex, real-world systems with stochastic elements cannot be accurately described by a mathematical model which could be evaluated *analytically*. Thus, simulation is often the only way of investigation possible.
- Simulation allows to estimate the performance of an existing system under same projected set of operating conditions.
- Alternative proposed system designs (or alternative operating policies for a single system) can be compared via simulation to see which one better meets a specified requirement.

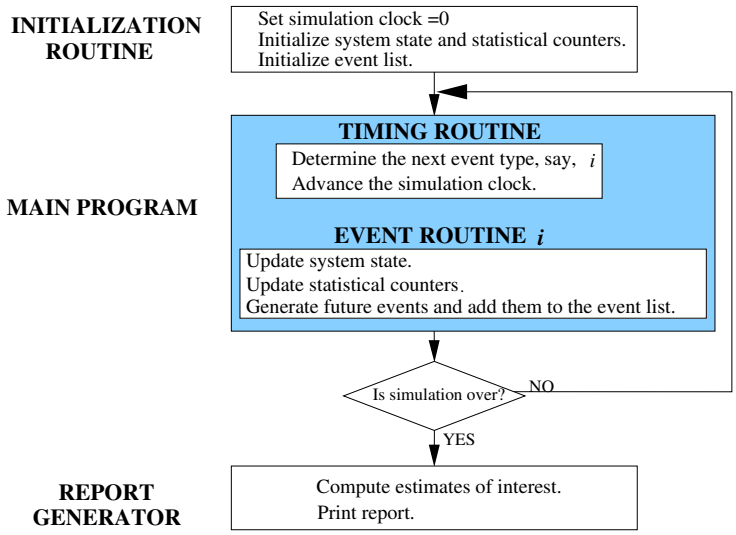


Figure 7.1: Structure of the Discrete-Event Simulation Model.

- In a simulation we can maintain much better control over experimental conditions than would generally not be possible when experimenting with the real system.
- Simulation allows to study in a short time the system running for a long time (e.g., an economic system), or alternatively a detailed study a system running in a short period of time.

The main drawback is the large required time to develop the simulation model, which must be a "valid" representation of the system under study for providing confidence simulation results.

7.2 Network Model

In this section we describe different aspects of the networks evaluated, as topologies, end-to-end flow control, traffic patterns and simulation parameters.

7.2.1 Topologies

Often, clusters and networks of workstations are arranged on regular network topologies when the performance is the primary concern. Low dimensional tori (2D and 3D) are one of the most widely used topologies in commercial parallel computers. Furthermore, recent proposals, such as Alpha 21364 [119] and BlueGene/L [44], use

2D and 3D tori, respectively. For this reason we have performed the evaluation of the proposed methodologies on tori networks. Notice that TFTR, RFTR and EBR proposals can be applied to any topology, even irregular topologies. However, the SPFTR and A-SPFTR are only suitable for tori.

7.2.2 End to End Flow Control

When considering faults in the network, some packets can be lost and some processing nodes might be blocked waiting for them, possibly blocking an application until the packet is received. Thus, an end-to-end flow control protocol is required (even if using a lossless network).

The flow control we have modelled consists of each end node keeps a copy of each packet it sends. Each destination notifies the received packets by sending an acknowledgment to the corresponding source end node. Thus, when the source receives the ACK, it discards the corresponding copy of the packet. A timer is used at the source end node to detect failed transmissions, thus triggering a retransmission of the packet. If the source end node does not receive the ACK of a packet before the timer expires, it re-sends the packet again.

Notice that it is possible that a packet correctly received be retransmitted because the ACK packet is lost or it arrives late at the source end node. In those cases the destination receives multiple copies of the same packet. All the redundant copies will be discarded, by using sequence numbers per flow (all these copies must be also notified to the source end node for stopping new retransmissions).

Finally, we also consider that many applications require that packets and messages are delivered in order. Thus, we modelled into the simulator a reorder buffer as a solution for such cases when packets are delivered out of order (due to packets being lost and retransmissions).

7.2.3 Traffic Patterns

We consider different traffic patterns when evaluating the network behaviour: synthetic patterns and traces.

Synthetic patterns are widely used because they allow to evaluate the network in the most generic way. When we use them, every end node has the same traffic injection rate. We evaluate the complete range of traffic injection rate, from low levels up to saturation point. The different synthetic traffic patterns used are: uniform, bitreversal, and hotspot. Each one has a different distribution of packet's destination. Next we describe each of them:

- For uniform traffic, each source end node sends packets to all the destinations with the same probability.

- For bitreversal, each source end node sends traffic only to one end node. The end node is computed by reversing all the bits of the source ID. Some numerical algorithms exhibit this type of distribution of destinations [42, 95].

- For hotspot, 10% of the sources (selected randomly) inject traffic to the same destination (selected randomly), the rest of end nodes inject traffic to random destinations. This traffic pattern allows to model the situation when one or more end nodes are frequently accessed by the remaining end nodes (a disk server, for instance).

On the other hand, traces are based on capturing the traffic when running real applications. Traces contain the source, destination, injection time, and the size of each packet sent. They allow to obtain results in more realistic scenarios and let compare with the results obtained when using synthetic patterns. In this thesis, some results obtained with this type of traffic pattern are shown.

The traces used were extracted under the execution of the FFT [145] and MP3D applications from SPLASH-2 [35] suite in a shared-memory multiprocessors. This type of applications are widely used when simulating multiprocessor systems, on engineering and scientific computations, and graphics processing:

- The FFT application processes a set of $n \times n$ real numbers. Data is organised into $\sqrt{n/p} \times \sqrt{n/p}$ matrixes to be distributed between p processors. The computation process requires three stages, all of them requiring all-to-all communication.

- MP3D: This application paralelizes the computation tasks by dividing the simulated system into different cells. For instance, this application is used on evaluating air flow dynamics in aeronautics.

7.2.4 Simulation Parameters

Next, we describe the main simulator parameters used in all the evaluations. Parameter values have been fixed following the IBA specs [89].

In IBA, packets are routed at each switch by accessing the forwarding table. This table contains the output port to be used at the switch for each possible destination. The routing time at each switch is set to 100 ns. This time includes the time to access the forwarding tables, the crossbar arbiter time, and the time to set up the crossbar connections.

We model a non-multiplexed crossbar on each switch. This crossbar supplies separate ports for each VL. IBA switches may support up to 16 Virtual Lanes (VLs). Buffers are used both at the input and the output side of the crossbar. Buffer size is fixed in both cases to 1 KB.

Links in InfiniBand are serial. In the simulator, the link injection rate is fixed to the 1X configuration [89]. 1X cables have a link speed of 2.5 Gbps. Therefore, a bit can be injected every 0.4 ns. With 8/10 coding [89] a new byte can be injected into the link every 4 ns. We also model the fly time (time required by a bit to reach the opposite link side). We model 20 m copper cables with a propagation delay of 5 ns/m. Therefore, the fly time is set to 100 ns.

The IBA specification defines a credit-based flow control scheme for each virtual lane with independent buffer resources. A packet is transmitted over the link if there is enough buffer space (credits of 64 bytes) to store the entire packet. IBA defines different packet sizes. In particular, MTU (Maximum Transfer Unit) is defined between 256 and 4096 bytes. Additionally, the *virtual cut-through* switching technique is used.

Packet size (both data and control packets) is fixed to 58 bytes, 32 bytes of payload and 26 bytes for the header. Flow control packets are 6 bytes long. Also, when evaluating the different reconfiguration schemes control packets are also considered. In particular, tokens carrying routing information are sized accordingly.

The values of the parameters for the end-to-end flow control have been optimized to achieve a good trade-off among a reduced retransmission time, the required memory resources for the pending packets at source and destination, and a reduced network overhead.

7.3 Modelling and Analyzing Faults

We consider link failures between switches[1]. Notice that to cope with failed links connecting switches to end nodes, it would be necessary to use either CAs with several ports or more than one CA per end node. Indeed, the failure of a link connecting an end node to a switch does not change the topology.

The fault tolerance degree of a mechanism is obtained by analyzing, for a certain number of faults, whether all the fault combinations keep the network logically connected. The method is n-fault tolerant if it provides for any combination of n failures a valid path for each source-destination pair. To this end, we should analyze all the possible fault combinations for every number of faults considered. However, as the number of faults increases, the number of possible fault combinations increases exponentially. Thus, from a particular number of faults upwards, it is impossible to explore all the fault combinations in a reasonable amount of time, specially in medium- and large-sized networks. To overcome this problem, we evaluate all the combinations on small network sizes[2], as performed in [86]. When the number of

[1] Note that a switch failure can be viewed as if all its links had failed.

[2] Notice that if a certain number of faults is tolerated by a small network, we can reasonably admit that it is also tolerated in larger networks.

fault combinations to be analyzed is too large from a computational point of view, we perform a statistical analysis (as followed in [86]), in which a representative subset of the total number of fault combinations is analyzed.

In Section 3.4.1 the mechanism to detect failures in InfiniBand is described. We have implemented the detection mechanism into the simulator. In particular, the SM sweeps the network periodically. The frequency of these sweeps is not defined by the IBA standard, thus it can be adjusted accordingly to parameters like the size of the network or the desired detection time for changes. We consider the sweep time fixed to 200,000 cycles (0.07ms).

Depending on the sweep time, the traffic lost due to the link failure will vary. With small sweep times the lost traffic will be low. However, more control packets (sweep packets) will be needed. The simulation considers also the required control traffic. As described in section 5.4, a special routing mechanism (*Directed-Route*) is used for sending the control packets.

We also modeled how the new routing info is distributed with control packets, and the confirmation of its reception since the control packets can also be discarded.

In order to carry out the performance evaluation under the occurrence of faults, we evaluate the network before any link failure happens, then we inject the failure and keep the simulation process until all the transitory effects of the applied fault-tolerant method disappear. To do this, we monitorize the network throughput and average packet latency.

7.4 Compared Reconfiguration Mechanisms

In the next chapter, the EVER reconfiguration mechanism will be evaluated and compared with recently proposed mechanisms.

In this section, we detail how the reconfiguration mechanisms, have been modeled in the simulator. In particular, we have implemented SR, DS and a static reconfiguration mechanism.

In all cases there exists a subnet manager (SM) located on an arbitrary end node (the same for all cases) that monitorizes the network for changes. This is achieved by sending periodically control packets to all the switches through the control virtual channel (VL15 in IBA).

For comparison purposes, two data virtual channels are modelled (VL0 and VL1) along with one control virtual channel for all the reconfiguration mechanisms, as DS requires at least two virtual channels. Notifications and routing table updates are sent through the control virtual channel in the same way as it is specified in InfiniBand (VL15). In particular, in all the reconfiguration mechanisms, once a topology change is detected, SM sends the new routing tables along with a control packet through the control virtual channel.

In DS, once a topology change is detected, SM computes the new routing tables and, afterwards, it distributes them to all end nodes and switches. At the same time, SM sends a "virtual channel drain" control packet to every switch and end node. Upon reception of the control packet the switch drains one of the two data virtual channels (i.e., VL1). Packets mapped in the data virtual channel are moved to the other data virtual channel (VL0) at each switch. Drainage thus occurs in parallel across all the network switches. Once VL1 is empty[3] and the new routing table is already updated[4], the SM is notified. SM then signals the end nodes and switches (via broadcasting a control packet) to start using both data virtual channels with the new routing function. The drained virtual channel (VL1) is used as the escape path for any packet in the other data virtual channel (VL0). Reconfiguration completes once all end nodes and switches are able to use both data virtual channels again.

In the case of SR two versions are considered. The first one, referred to as SR-PDA (SR Packet Dropping Aware), issues the tokens that separate the old and new traffic at the same time tables are sent to switches. Thus, tables are sent in parallel with a "reconfiguration" control packet that is broadcasted by the SM to all end nodes and switches, signaling nodes to generate reconfiguration tokens. This mechanism tends to reduce the number of dropped packets as it minimizes the time that R_{old} is in use, thus minimizing the number of packets that could be dropped due to routing across failed links/switches (at the expense of possibly longer average packet latencies). The second one, referred to as SR-LA (SR Latency Aware), first distributes the new routing tables and then sends the tokens to separate the traffic. SM first distributes and stores the new routing tables into a secondary location in the end nodes and switches before it broadcasts a "reconfiguration" control packet. It potentially reduces average packet latency at the expense of possibly exhibiting longer reconfiguration times.

In static reconfiguration (ST), once the topology change is detected, SM broadcasts a "network drain" control packet to all end nodes, instructing them to halt packet injection. At the same time, SM starts computing the new routing tables. Once the tables are computed and distributed to all end nodes and switches, and after the network is completely drained of all data packets, the SM broadcasts a control packet instructing the end nodes to resume packet injection. Reconfiguration completes once all nodes are allowed to inject packets again.

Figure 7.2.a compares the actions taken by each reconfiguration scheme. As can be observed in Figure 7.2.a, all the methods have in common the computation and distribution of routing tables. While computing the new routing tables one data virtual channel (DS) or the entire network (ST) is drained. Another detail is that in

[3]A distributed protocol is used for detecting that the data virtual channel is empty on every switch [124]. Each switch sends a notification to the SM once the data virtual channel is empty and there are no data packets injected through the output links sent by the switch.

[4]It requires each switch to store two different routing tables at the same time.

SR-PDA routing table distribution and tokens distribution occur in parallel, whereas in SR-LA it occurs sequentially. In DS and SR, SM issues control packets in order to inform switches that new routing tables are ready to be used. Finally, ACK packets are issued by switches to the SM in DS and SR (not in EBR) in order to acknowledge that reconfiguration has finished. In EBR the only task requested is the routing table distribution. Figure 7.2.b shows the different events for each of the simulated reconfiguration schemes.

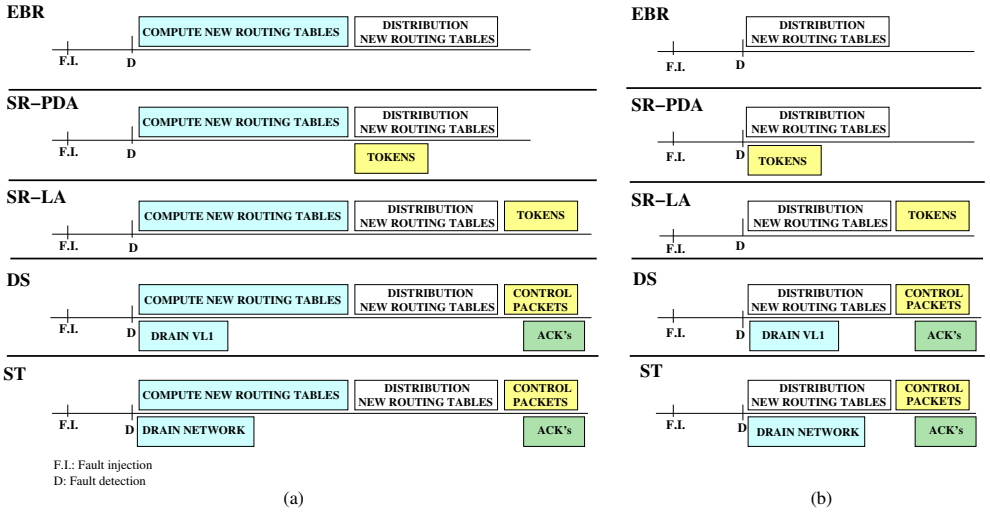


Figure 7.2: Reconfiguration schemes evaluated and events that distinguish them.

7.5 Collecting Results

For each simulation run (with synthetic traffic patterns), we assume that the packet generation rate is constant and the same for all the end nodes. In the simulations we have considered the delivery of the first 80,000 packets to be a transient state. Then, we inject the failure 0.02 secs (50,000,000 cycles) after the transient state. The simulation time varies depending on network load. For example, the reconfiguration process takes much more time when the network is congested. Therefore, we keep the simulation running until the process has finished and the network has reached a steady state after the failure.

Figure 7.3.a shows the performance achieved (accepted traffic is overage packet latency) for a 10×10 torus network with uniform traffic pattern. The second and third row of figures show the throughput evolution for the highest traffic injection rate before saturation (the simulated point just before the system is entering saturation corresponds to point 7 in Figure 7.3.a). The second row shows the throughput

evolution as a in function of simulation time. Whereas, the third row shows the evolution as a function of the number of delivered packets.

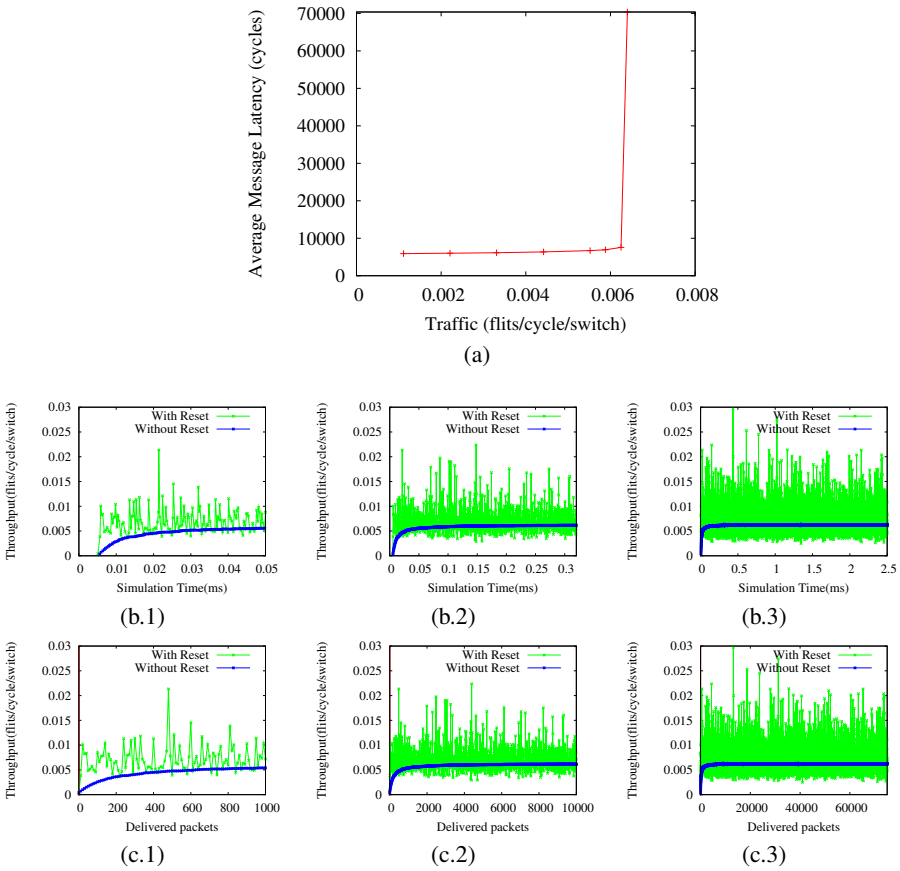


Figure 7.3: Determining transient state duration in the simulations.

In each figure we plot the throughput for each interval of 10 received packets, considering both *reset statistics* for each interval (green series) and *no reset statistics* (blue series). When reset statistic is activated, the throughput is measured as a function only of the 10 messages received for each given point (notice that there is high variability as the time period is very small, just 10 messages), whereas when *reset* is deactivated, the throughput is measured as a function of all the messages received until a given point. Each column of figures represents a different zoom degree, being the one with greater zoom the first one.

As can be noticed in Figure 7.3.c.2, the average network throughput (blue series) is well stable for 2,000 packets. Therefore, it seems quite reasonable that 80,000 packets is by far sufficient for the transient state (see Figures 7.3.b.3 and 7.3.c.3),

since results throughput curve is almost flat after a few thousands of packets are delivered. This way of selection the transient state period is advocated also in [67]. Notice that the x-axis in figures 7.3.c.1, 7.3.c.2, and 7.3.c.3 shows the absolute numbers of the total delivered packets from the start time, whereas y-axis shows relative numbers to an instant of time.

7.5.1 Performance Metrics

In all the evaluations, we represent the average latency, measured in nanoseconds, versus the accepted average traffic, measured in bytes/ns/switch. The latency corresponds to the elapsed time from the packet injection until the last byte of the packet arrives to its destination. The accepted traffic is the accepted amount of information in the network by time unit. Also, we represent the evolution of the average latency of the packets and the average latency from the generation time. The latency from the generation time is the elapsed time from the generation of the packet in the source node until it is delivered to the destination node.

Traditionally, the latency provided by the network has been plotted as the latency of packets at delivery time. Figure 7.4.a shows the traditional plot. At point x the plot indicates that the packets that **arrived** at time x exhibited an average latency of y cycles. In this document, however, the latency is plotted also at generation time. That is, latency of packets is accumulated at their generation time. For instance, in the new plot (Figure 7.4.b), packets **generated** at time x exhibited an average latency of y. By doing this, the impact of the reconfiguration on packets is better appreciated.

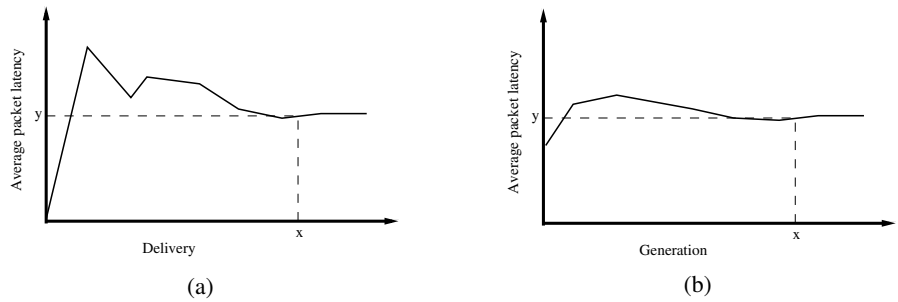


Figure 7.4: Plotting latency at (a) reception time or at (b) injection time.

Packet latency is broken down in the following components: queue latency, network latency, and token latency. Queue latency is the time the packets spend at the source end node waiting to be injected. The network latency is the time the packets spend within the network. Finally, token latency is the time packets are blocked at the head of a queue waiting for a token (only when SR is applied). Results for average latencies are presented.

Some results, presented in this thesis, require an confidence interval in order to quantitatively evaluate how representative they are. In such cases, the confidence interval we provide corresponds to a range that contains the true mean in terms of a confidence coefficient of 95% (it is the most commonly used).

This page intentionally left blank

Chapter 8

Evaluation

In this Chapter we evaluate all the contributions made in this thesis. As the main metrics, we are interested in the fault tolerance degree achieved by each mechanism. Moreover, we want to analyze the resources required by each contribution to achieve a certain fault tolerance degree. Therefore, as the main metric we analyze the trade-off between fault tolerance and resources needed. We also, however, analyze other metrics like network throughput, latency, and path quality. Evaluation is performed assuming the network parameters defined by IBA specs (described in the previous chapter).

This chapter is organized as follows. In Section 8.1 we evaluate the TFTR methodology. In Section 8.2 we evaluate the SPFTR and the A-SPFTR methods. In Section 8.3 the RFTR mechanism is evaluated. Finally, in Section 8.4 we evaluate the EBR mechanism.

8.1 TFTR Methodology

The TFTR methodology computes disjoint paths to tolerate multiple faults without disconnecting any pair of end nodes. However, the number of disjoint paths that can be obtained depends on the minimum degree of any switch in the network. As is a maximum of 4 disjoint paths in a 2D torus (6 in a 3D torus), 3 faults can be tolerated at most in 2D tori (5 in a 3D tori). Therefore, the main goal of the evaluation is to know the number of resources required to achieve such a fault tolerance degree.

In particular, different number of SLs is granted in order to evaluate the fault tolerance degree achieved by the methodology. Regarding virtual channels, TFTR obtains the complete set of disjoint paths in the evaluated 2D and 3D tori using only two virtual channels. However, we have also evaluated TFTR when using four virtual channels in 2D tori. Also, for comparison purposes, we analyze the fault tolerance achieved by the *up*/down** routing algorithm.

Different scenarios are evaluated in order to obtain different conclusions from the TFTR mechanism. Table 8.1 summarizes the different scenarios considered, showing for each one the routing algorithms, topologies, resources, traffic patterns, injection rates, and failure combinations evaluated.

Routing algorithm or mechanism	Max VLs	Max SLs	Faults	Topologies	Traffic pattern	Injection rate
UD	1	1	$0 \ldots 4$	$4 \times 4 \ldots 10 \times 10$ torus	not required	
TFTR	2	unbounded	$0 \ldots 4$	$4 \times 4 \ldots 10 \times 10$ torus	not required	
TFTR	4	unbounded	$0 \ldots 4$	$4 \times 4 \ldots 7 \times 7$ torus	not required	
TFTR	2	2,3,4	0	$4 \times 4 \ldots 8 \times 8$ torus	not required	
TFTR	2	unbounded	$0 \ldots 4$	$4 \times 4 \ldots 8 \times 8$ torus	uniform	Full range*
TFTR	2	unbounded	$0 \ldots 6$	$4 \times 4 \times 4$ torus	not required	

* means from low up to high injection rate when the network reaches the saturation point.

Table 8.1: Evaluated scenarios for TFTR.

8.1.1 Number of Paths

Table 8.2 shows the number of alternative disjoint paths obtained by the *up*/down** and TFTR routing algorithms. One virtual channel is used for *up*/down** and two virtual channels for TFTR[1]. The table shows the percentage of source-destination pairs for which one, two, three or four disjoint paths could be computed. Results for any combination of routing algorithm and topology are shown.

As we can observe, TFTR obtains the maximum number of disjoint paths for all the pairs of end nodes, i.e., four disjoint paths for all the evaluated 2D tori and six paths for the evaluated 3D tori. So, TFTR tolerates up to 3 and 5 failures in 2D and 3D tori, respectively, using only two virtual channels. On the other hand, *up*/down** is not able to find a sufficient number of disjoint paths for every source-destination pair. Indeed, for all network sizes there is a significant number of source-destination pairs with a unique *up*/down** path. So, *up*/down** is not able to tolerate even 1-link failure.

8.1.2 Singular Cases

Figure 8.1 shows the percentage of singular cases[2] when using *up*/down** and TFTR for different number of faults (in 2D tori). All fault combinations have been examined except for 5 and 6 faults, where a random set of fault combinations has been analyzed. In situations where the number of combinations become too large, only

[1]Notice that an additional virtual channel with *up*/down** does not allow new paths to be used.

[2]Definition of singular case appears on page 40.

Torus	SLs	Percentage of end node pairs with n disjoint paths			
		n=1	n=2	n=3	n=4
TFTR (2 VLs)					
$4 \times 4, 5 \times 5$	3	0.00%	0.00%	0.00%	**100.00%**
$6 \times 6, 7 \times 7$	4	0.00%	0.00%	0.00%	**100.00%**
8×8	5	0.00%	0.00%	0.00%	**100.00%**
9×9	6	0.00%	0.00%	0.00%	**100.00%**
10×10	7	0.00%	0.00%	0.00%	**100.00%**
up/down** (1 VL)					
4×4	1	19.17%	30.00%	33.33%	17.50%
5×5	1	22.00%	38.00%	30.00%	10.00%
6×6	1	25.87%	42.54%	25.08%	6.51%
7×7	1	26.53%	47.11%	21.77%	4.59%
8×8	1	28.13%	49.80%	18.65%	3.42%
9×9	1	28.09%	52.78%	16.48%	2.65%
10×10	1	28.83%	54.55%	14.51%	2.12%
Torus	SLs	n=1	n=2	n=5	n=6
TFTR (2 VLs)					
$4 \times 4 \times 4$	3	0.00%	0.00%	0.00%	**100.00%**

Table 8.2: Percentage of end node pairs with n disjoint paths, when using an unbounded number of SLs.

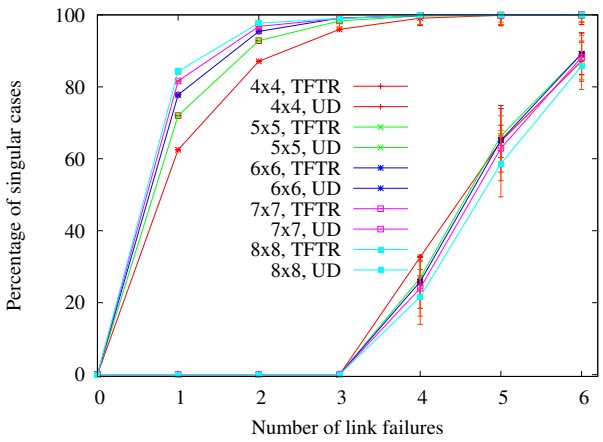


Figure 8.1: Percentage of singular cases in 2D tori that could not be handled by the methods.

100,000 randomly generated cases are evaluated. Confidence intervals are used for those cases.

We can observe that TFTR does not present any singular case for one, two, and three fault cases, as it is able to obtain four disjoint paths. Therefore, the methodology is 3-fault tolerant with 2 VLs for 2D tori. However, for more than 3 faults, the proposed methodology is not able to tolerate all combinations. For example, up to 30% of 4-faults combinations are not tolerated in a 4×4 torus. This percentage decreases slightly as network size increases, and increases as the number of faults in the network increases. For six faults, almost none of the failure combinations is tolerated by the methodology when it is applied to 2D tori. However, it is to notice that the *up*/down** routing algorithm has a pretty worse behavior. Nearly, all fault combinations, even for the one fault case, are not tolerated by this algorithm. Notice that, when a singular case arises, the only solution is to launch a (local or global) network reconfiguration process to compute new routing tables.

8.1.3 Resources Required

Once confirmed that two virtual channels are enough to tolerate three faults for 2D tori and five faults for 3D tori, we analyze the number of service levels that are required in order to correctly use these virtual channels in InfiniBand. Table 8.2 shows also the number of required service levels (in an unbounded scenario). The need of SLs increases as network size increases. In particular, 3 SLs are required for a 4×4 torus network and 7 SLs for a 10×10 torus network. As paths get longer, the probability of mapping conflicts increases, and thus, additional SLs are required. It is important to note also that for the 3D torus network, only 3 SLs are required. This is an important observation since it may be used to select the final topology. As an example, a network with 64 end nodes can be designed by using a $4 \times 4 \times 4$ torus in order to obtain a fault-tolerant system with only 3 SLs, rather than using a 8×8 torus requiring 5 SLs.

Taking into account that IBA allows up to 16 SLs, the number of SLs required by TFTR for some topologies limits the applicability of SLs to other purposes, like QoS. Therefore, a bounded scenario where the number of available SLs is limited should be also evaluated. Table 8.3 shows the number of disjoint paths obtained by TFTR when the number of SLs is bounded. Notice that results for just one SL are not shown as it does not allow to use any VL transition, and therefore results would be the same as in the case of *up*/down**.

When reducing the number of SLs from 5 down to 4, the TFTR strategy can only guarantee to tolerate two faults in large networks. Using 3 SLs it is possible to tolerate only one single link failure, from a 6×6 network and upwards. However, in 4×4 and 5×5 tori, it is still possible to tolerate up to 3 link failures. Finally, when using only 2 SLs one link failure is tolerated for all network sizes.

When limiting the number of SLs, the number of available disjoint paths decreases for some source-destination pairs. In particular, Table 8.3 shows that from

Torus	SLs	Percentage of end node pairs with n disjoint paths			
		n=1	n=2	n=3	n=4
4×4	2	0.00%	0.83%	4.58%	94.58%
5×5	2	0.00%	1.17%	15.17%	83.67%
6×6	2	0.00%	1.75%	23.02%	75.24%
	3	0.00%	0.08%	4.52%	95.40%
7×7	2	0.00%	2.93%	29.97%	67.09%
	3	0.00%	0.40%	20.87%	88.61%
8×8	2	0.00%	4.32%	30.53%	65.13%
	3	0.00%	0.77%	14.26%	84.97%
	4	0.00%	0.00%	5.03%	94.97%

Table 8.3: Percentage of pairs of end nodes with n disjoint paths when using 2 VLs and a bounded number of SLs.

4×4 to 7×7 torus networks, some source-destination pairs were granted only two disjoint paths. Although the percentage of these cases is low, it limits the fault tolerance degree of TFTR to one fault, no matter how low is this percentage. When limiting the number of SLs to 4 for the 8×8 torus, the methodology only tolerates 2 faults. Therefore, the methodology effectiveness is highly sensitive to the number of SLs available.

Table 8.4 shows the percentage of paths that use each SL identifier (all the packets using the path are labeled with the same SL). As can be seen, with two SLs (SL0 and SL1) roughly 90% of paths can be routed without any mapping conflict. However, a small fraction of the paths requires additional SLs in order to avoid mapping conflicts, which increases with network size.

Torus	Max SLs	Percentage of end node pairs using each SL			
		SL=0	SL=1	SL=2	SL=3
4×4	2	58.10%	41.90%	-	-
5×5	2	55.21%	44.79%	-	-
6×6	2	53.55%	46.45%	-	-
	3	49.81%	43.40%	6.79%	-
7×7	2	52.45%	47.55%	-	-
	3	48.29%	44.60%	7.12%	-
8×8	2	51.74%	48.26%	-	-
	3	47.83%	45.37%	6.80%	-
	4	45.82%	43.75%	6.79%	3.6%

Table 8.4: Percentage of paths using a particular SL with a bounded number of SLs and 2 VLs.

8.1.4 Quality of Routing Paths

Table 8.5 shows some metrics of the paths computed by TFTR and *up*/down** when the number of SLs is unbounded. TFTR obtains, on average, longer paths than those obtained by *up*/down**. Note that to achieve disjoint paths, some non-minimal paths

are required, thus, increasing the average path length. However, when taking into account only the shortest path for every source-destination pair we notice that TFTR obtains shorter paths than *up*/down**. Remember that TFTR is able to perform some down → up link transitions, which allows TFTR to provide minimal paths not achieved by *up*/down**. Indeed, the average path length when considering only the shortest paths for every source-destination pair is close to the average topological distance (almost every source-destination pair has at least one minimal path).

Routing algorithm	Torus	VLs	SLs	Percentage of paths with t transitions		Average path length	Average path length of shortest paths	Average topological distance
				t=0	t=1			
up/down**	4×4	1	1	100.00%	0.00%	2.8495	2.1333	2.1333
TFTR		2	3	56.77%	43.23%	3.2145	2.1333	
up/down**	5×5	1	1	100.00%	0.00%	3.5278	2.6933	2.5000
TFTR		2	3	51.92%	48.08%	3.9666	2.5067	
up/down**	6×6	1	1	100.00%	0.00%	4.1773	3.3063	3.0857
TFTR		2	4	49.03%	50.97%	4.6885	3.0873	
up/down**	7×7	1	1	100.00%	0.00%	4.8673	3.9435	3.5000
TFTR		2	4	46.25%	53.75%	5.3958	3.5230	
up/down**	8×8	1	1	100.00%	0.00%	5.5248	4.5878	4.0000
TFTR		2	5	45.03%	54.97%	6.0781	4.0804	
up/down**	9×9	1	1	100.00%	0.00%	5.8371	5.1722	4.5000
TFTR		2	6	41.54%	58.46%	6.8347	4.5930	
up/down**	10×10	1	1	100.00%	0.00%	6.8459	5.8921	5.0000
TFTR		2	7	40.77%	59.23%	7.5212	5.1493	
TFTR	$4 \times 4 \times 4$	2	3	46.28%	53.72%	4.5627	3.0480	3.0476

Table 8.5: Routing path metrics for TFTR and *up*/down**.

8.1.5 TFTR with Additional Virtual Channels

If more resources are granted to the methodology, TFTR is able to obtain even better sets of paths. Table 8.6 shows the metrics for paths computed with four VLs and an unbounded number of SLs. In this case, maximum priority to minimizing the path length is given in order to select the final set of paths. We can observe that for all the topologies, the average path length is reduced, including the average shortest path length, which becomes even closer to the average topological distance. However, it is not worth using more VLs because they do not contribute to increase the fault tolerance degree. Moreover, using more VLs generates more mapping conflicts which increases the required number of SLs.

8.1.6 Network Throughput

Figure 8.2 shows the performance degradation suffered by the network in the presence of faults when TFTR is used. Every source end node uses the shortest disjoint

Torus	VLs	SLs	Average path length	Average shortest path length	Average topological distance
4×4	4	5	3.0917	2.1333	2.1333
5×5	4	8	3.7363	2.5033	2.5000
6×6	4	12	4.4698	3.0889	3.0857
7×7	4	14	5.0112	3.5000	3.5000

Table 8.6: Routing path metrics for TFTR when using 4VLs and an unbounded number of SLs.

path that does not traverse any faulty link. For a particular number of faults, different random combinations of faults are injected. For every combination of faults the network is simulated (using the simulation tool and simulation parameters presented in Chapter 7). Uniform traffic pattern is used and the packet size is fixed to 58 bytes (this includes the IBA packet header, the packet payload of 32 bytes and the IBA packet tail). Confidence intervals are shown for every number of failures, however, they are so small that they are hardly visible.

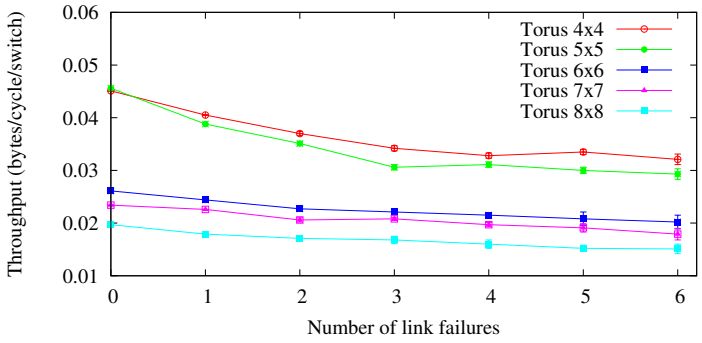


Figure 8.2: Network throughput degradation when using TFTR for different torus networks.

As can be observed, throughput decreases as the number of faults increases. However, it can be noticed that performance degradation is relatively low in larger networks. This is because the same number of faulty links affects a lower percentage of paths in larger networks. For instance with 6 faulty links, throughput decreases up to 28% for the 4×4 torus and only 18% for the 8×8 torus.

It has to be noted that the performance degradation is due to the change in the topology rather than to the TFTR mechanism. Anyway, a different set of paths may

lead to better performance. However, the real goal of TFTR is to keep connectivity rather than to keep performance to its maximum.

8.1.7 Computational Cost

Figure 8.3 shows the time required by TFTR to compute all the paths for different 2D tori. As can be observed the computational time grows exponentially with the radix of the network (bear in mind the logarithmic scale). For instance, to compute all the disjoint paths in a 10×10 torus network, TFTR takes one hour on a Xeon 3.06 GHz processor.

Although these are bad news for TFTR, it is fair to say that the computation process is performed offline. Indeed, disjoint paths are already computed when the failure is detected. Anyway, the computational time is excessive and claims for further improvements.

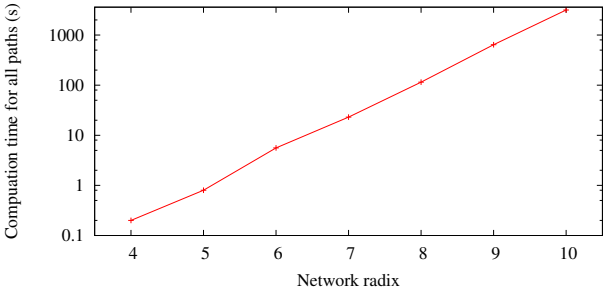


Figure 8.3: Computational cost of TFTR for computing the set of disjoint paths on different 2D torus networks.

8.1.8 Summary

We can obtain some interesting conclusions from the analysis of the obtained results. First, alternative disjoint paths are required to obtain a certain degree of fault tolerance. The *up*/down** is not able to tolerate even a single fault. TFTR requires some resources. In particular, in IBA networks, an increasing number of SLs is needed as network size increases. However, two virtual channels suffice regardless of networks size. Second, neither the path quality nor the network throughput suffer much when computing paths with TFTR. Indeed, average shortest path length is very close to

the average topological distance. Third, when using additional virtual channels no benefits are provided in terms of fault tolerance.

However, the main drawback of TFTR is its computational cost. Although TFTR is performed offline it has an exponential cost. This fact motivates the convenience of using SPFTR which is evaluated next.

8.2 SPFTR and A-SPFTR Methodologies

In this section, we evaluate SPFTR and A-SPFTR. Notice that both models are scalable in size, but only SPFTR is scalable in the number of dimensions. For comparison purposes, we also evaluate the *up*/down** routing scheme.

As in TFTR, different scenarios are considered when evaluating SPFTR and A-SPFTR. Table 8.7 summarizes each scenario.

Routing algorithm or mechanism	Faults	Topologies	Traffic pattern	Injection rate
UD, SPFTR and A-SPFTR	$0 \ldots 6$	$5 \times 5 \ldots 20 \times 20$ torus	not required	
SPFTR	$0 \ldots 8$	$6 \times 6 \times 6 \ldots 8 \times 8 \times 8$ torus	not required	
SPFTR and A-SPFTR	$0 \ldots 6$	$5 \times 5 \ldots 8 \times 8$ torus	uniform	low, medium, high

Table 8.7: Evaluated scenarios for SPFTR and A-SPFTR.

8.2.1 Singular Cases

Figure 8.4 shows the percentages of singular cases for link failures when using *up*/down** and *SPFTR* (results for A-SPFTR are similar) for different number of faults. As can be observed, *SPFTR* does not present any singular case until four links have failed for 2D tori, and six links have failed for 3D tori. Therefore, the methodology is $(2n-1)$-fault tolerant with 2 VLs and 4 SLs for n-dimensional tori (SPFTR) or with 2 VLs and 3 SLs for 2D tori (A-SPFTR). On the other hand, as previously seen *up*/down** does not tolerate even a single link failure.

We can see that none of the routing methods is able to tolerate all the failure combinations in 2D tori in presence of more than 3 faults. Indeed, for four failures in the network, 30% of combinations are not tolerated in the worst case (the worst case was the smallest evaluated torus, 5×5 torus network, decreasing this percentage for larger torus, see Figure 8.4.b). When compared with TFTR (Figure 8.1) we can observe that roughly the same fault tolerance levels are achieved.

Figure 8.5 shows that when considering switch failures, *SPFTR* does not have any singular case until four switches fail for 2D tori. The percentage of singular cases, in case of more than 3 switches fail, however, is higher than when considering the same number of link faults.

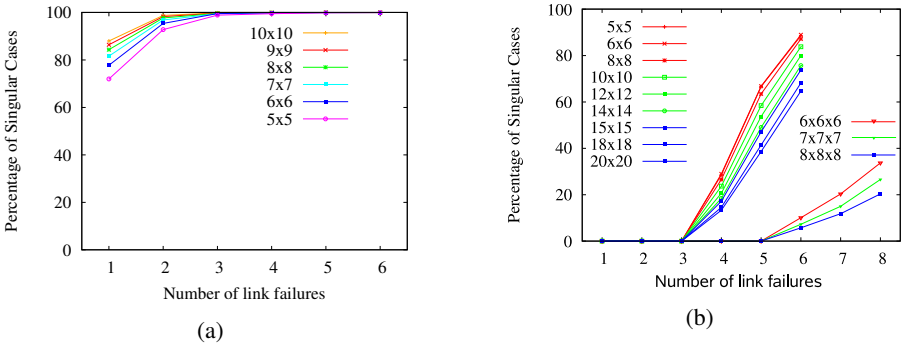


Figure 8.4: Singular cases for link failures for (a) *up*/down** and (b) SPFTR in 2D and in 3D tori (100,000 combinations evaluated at each point when the number of fault combinations is higher than 100,000).

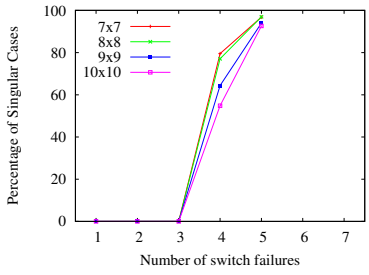


Figure 8.5: Singular cases for switch failures for SPFTR in 2D tori (100,000 combinations evaluated at each point when the number of fault combinations is higher than 100,000).

8.2.2 Quality of Routing Paths

Figures 8.6.a and 8.6.b show the average path length of the shortest paths computed by SPFTR, A-SPFTR (only for 2D tori), and the *up*/down** routing algorithm. Also, the figures show the average topological distance. When considering only the shortest path for every pair of end nodes (the common case in the fault-free scenario), SPFTR and A-SPFTR achieve, on average, shorter paths than *up*/down**. Additionally, as network size increases, the difference between the average path length and the average topological distance slightly increases. This is due to the fact that only one forbidden transition is allowed to be crossed (2 VLs are used), causing some route patterns to need longer paths to reach destination.

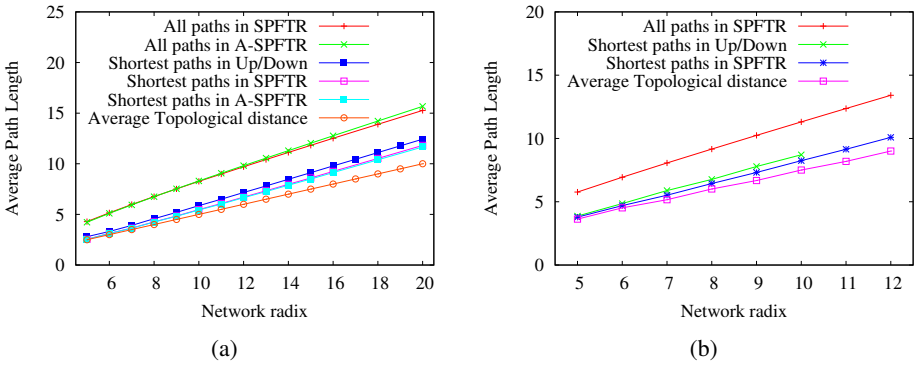


Figure 8.6: Average path length in (a) 2D torus and (b) in 3D torus.

8.2.3 Network Throughput

Figure 8.7 shows the performance degradation suffered as the number of faults increases when applying the SPFTR methodology (similar results are obtained when the A-SPFTR is evaluated).

In the presence of faults every source end node uses the shortest disjoint path that does not traverse any faulty link. For a particular number of faults, different random combinations of faults are injected. For every combination of faults the network is simulated, its throughput is obtained and the average results are displayed.

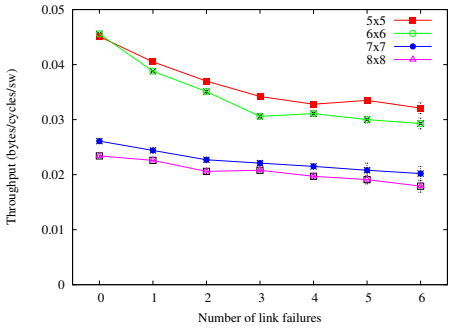


Figure 8.7: Degradation of performance with faults for SPFTR and A-SPFTR.

As can be observed, results are similar to those obtained by TFTR. Indeed, performance is mainly affected by the topology modification due to failures.

8.2.4 Computational Cost and Resources Needed

All the previous results show the same trend achieved by TFTR. Indeed, computed paths are similar and thus they obtain similar performance and fault tolerance. However, two important differences arise. The first one is that SPFTR bounds the number of SLs to four (SPFTR) or three (A-SPFTR). Figure 8.8 shows the comparison between TFTR and SPFTR of the required number of SLs in 2D tori.

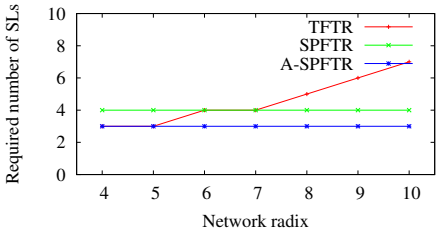


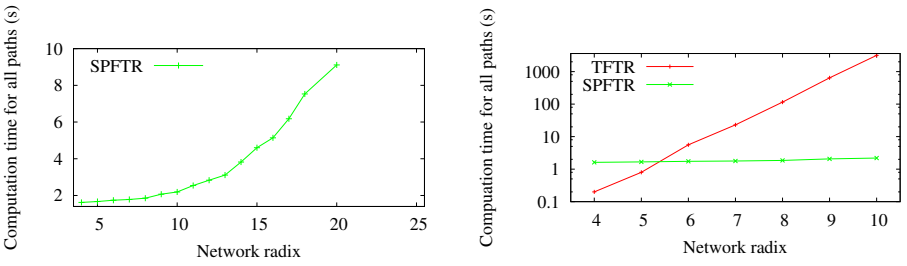
Figure 8.8: Comparison between TFTR and SPFTR of the required number of SLs in 2D tori.

The second one, is the computational cost. Figure 8.9.a shows the computational cost of SPFTR (for A-SPFTR same results are obtained) for different 2D tori. As can be observed, the computational time grows quadratically with the number of end nodes in the network. Notice that only the computation of forwarding tables is required. This is because SLtoVL tables and route patterns are already computed. Indeed, when comparing TFTR and SPFTR in terms of computational cost (Figure 8.9.b) we can observe the large differences between both methods. As an example, all the routing information for a 20×20 torus network when using SPFTR was computed in less than 10 seconds using an Intel Xeon 3.06 GHz. For TFTR this would take more than a week and would consume many resources (SLs) on the same processor.

8.2.5 Summary

Although TFTR and SPFTR provide an average path length close to the topological distance and achieve similar results in terms of throughput, SPFTR provides the complete set of disjoint paths with only 2VLs and 3SLs, whereas TFTR requires an increasing number of SLs as network size increases. SPFTR also minimizes the time needed to compute the required set of disjoint paths.

Therefore, SPFTR should be preferred to provide disjoint paths when using a torus network. However, TFTR should be used when using other network topologies.



(a) (b)

Figure 8.9: Comparison of computational cost of TFTR and SPFTR in a Xeon 3.06 GHz (a) linear scale and (b) logarithmic scale.

Regardless of the computational cost and the number of resources needed, there is another major limitation that may affect the choice of the fault tolerance method. This is the number of link failures to be tolerated. Indeed, both methods evaluated so far have the same fault tolerance degree. Both tolerate up to three link failures in 2D tori and up to five link failures in 3D tori. The next methods will overcome this issue and will tolerate much more faults. Indeed, they are dynamic methods that will adapt to topology changes due to link failures.

8.3 RFTR Methodology

Now, we focus our attention on RFTR. As previously done, we analyze its fault tolerance degree, the required resources, computational time, and exhibited performance. Also, we compare RFTR with the previous methods (TFTR and SPFTR), and with respect to the alternative of applying a reconfiguration process, such as the Simple Reconfiguration method. Contrary to TFTR and SPFTR, RFTR is able to dynamically provide new paths as long as faults appear, thus, potentially achieving higher fault tolerance levels. Simple reconfiguration is a dynamic network reconfiguration method which has proved to be a fast mechanism that works for any topology and between any pair of old and new routing functions.

Following the previous analysis, we consider torus network topologies. In particular, we analyze 2D tori with different sizes, ranging from 16 switches (4×4) up to 196 switches (14×14). Also 3D tori with 64 switches ($4 \times 4 \times 4$) and 216 switches ($6 \times 6 \times 6$) are analyzed. In all these cases, two end nodes are attached to each switch. Table 8.8 shows for each scenario the routing algorithms, topologies, and fault combinations evaluated.

RFTR is evaluated using *up*/down** routing as the underlying routing algorithm.

Mechanisms or routing algorithm	Failures	Topologies
SPFTR	$0 \ldots 8$	4×4 and $4 \times 4 \times 4$ torus
TFTR	$0 \ldots 8$	4×4 and $3 \times 3 \times 3$ torus
RFTR	$0 \ldots 8$	$4 \times 4 \ldots 14 \times 14$ torus
RFTR	$0 \ldots 8$	$3 \times 3 \times 3 \ldots 6 \times 6 \times 6$ torus
up/down**	0	$5 \times 5 \ldots 14 \times 14$ torus
up/down**	0	$3 \times 3 \times 3 \ldots 6 \times 6 \times 6$ torus
up/down**	$0 \ldots 8$	5×5 torus
up/down**	$0 \ldots 8$	$8 \times 8 \times 8$ torus

Table 8.8: Evaluated scenarios for RFTR.

We have selected this routing scheme because, unlike DOR, it is able to provide some alternative paths between every source-destination pair, thus providing greater flexibility. In most cases, two VLs are used by RFTR, thus, at most, only one transition is allowed to each path. In order to maximize the benefits from the two available VLs, packets not requiring virtual channel transitions are randomly injected through any of the two available virtual channels. For paths with a VL transition, packets are injected through VL0.

8.3.1 Singular Cases

Table 8.9 shows the results obtained when considering up to eight link failures in 4×4 and $3 \times 3 \times 3$ tori. For each number of faults, the number of combinations analyzed can be observed. As can be seen, RFTR is able to tolerate all the link failure combinations. RFTR is 8-fault tolerant in the 4×4 torus, and 4-fault tolerant in the $3 \times 3 \times 3$ torus (100% of fault combinations analyzed), whereas from 5 faults upward we only can state it in statistical terms in the $3 \times 3 \times 3$ torus. This is a reasonable fault tolerance degree taking into account the network sizes considered (it represents more than 5% of failed links).

When comparing RFTR with respect to TFTR and SPFTR, we can observe that RFTR is able to tolerate a larger number of faults, whereas TFTR and SPFTR only can tolerate up to 3 faults in 2D tori and 5 failures in 3D tori.

8.3.2 Quality of Routing Paths

Figures 8.10.a and 8.10.b show for a 2D and a 3D tori the average path length provided by RFTR. Also, for comparison purposes, the average path length provided by the reconfiguration process is plotted. As a reference, the average topological distance between switches is also shown. We can observe that the difference between the average path length and the average topological distance slightly increases as the

Number of failures	RFTR				TFTR			SPFTR		
	Evaluated cases	Percentage of non tolerated cases	Max VLs	Max SLs	Percentage of non tolerated cases	Max VLs	Max SLs	Percentage of non tolerated cases	Max VLs	Max SLs
	Torus $3 \times 3 \times 3$				Torus $3 \times 3 \times 3$			**Torus** $4 \times 4 \times 4$		
1	81*	**0.00%**	2	3	**0.00%**	2	3	**0.00%**	2	4
2	3,240*	**0.00%**	2	3	**0.00%**	2	3	**0.00%**	2	4
3	85,320*	**0.00%**	2	3	**0.00%**	2	3	**0.00%**	2	4
4	1,663,740*	**0.00%**	2	5	**0.00%**	2	3	**0.00%**	2	4
5	25,621,596	**0.00%**	2	5	**0.00%**	2	3	**0.00%**	2	4
6	6,490,804	**0.00%**	2	5	0.27%	2	3	0.06%	2	4
7	17,386,083	**0.00%**	2	5	1.63%	2	3	0.40%	2	4
8	16,082,127	**0.00%**	2	5	5.34%	2	3	5.34%	2	4
	Torus 4×4				Torus 4×4			Torus 4×4		
1	54*	**0.00%**	2	3	**0.00%**	2	3	**0.00%**	2	4
2	1,431*	**0.00%**	2	5	**0.00%**	2	3	**0.00%**	2	4
3	24,804*	**0.00%**	2	6	**0.00%**	2	3	**0.00%**	2	4
4	316,251*	**0.00%**	2	6	27.01%	2	3	31.25%	2	4
5	3,162,510*	**0.00%**	3	7	58.92%	2	3	74.96%	2	4
6	25,827,165*	**0.00%**	3	7	83.76%	2	3	95.96%	2	4
7	177,100,560*	**0.00%**	3	8	94.72%	2	3	99.72%	2	4
8	1,040,465,790*	**0.00%**	3	8	99.87%	2	3	99.99%	2	4

*means that all the fault combinations have been analyzed.

Table 8.9: Fault tolerance achieved and resources required by RFTR, TFTR, and SPFTR. For SPFTR the minimum network size for 3D torus topology is $4 \times 4 \times 4$.

number of faults increases. Moreover, the average path length values obtained by reconfiguration are slightly lower than the ones obtained by RFTR. This is due to the fact that the methodology tries to minimize the number of VLs used (usually 2 VLs, and in some cases up to 3 VLs). This leads to select, in some cases, intermediate switches that are not in the minimal path from source to destination, thus increasing the average path length.

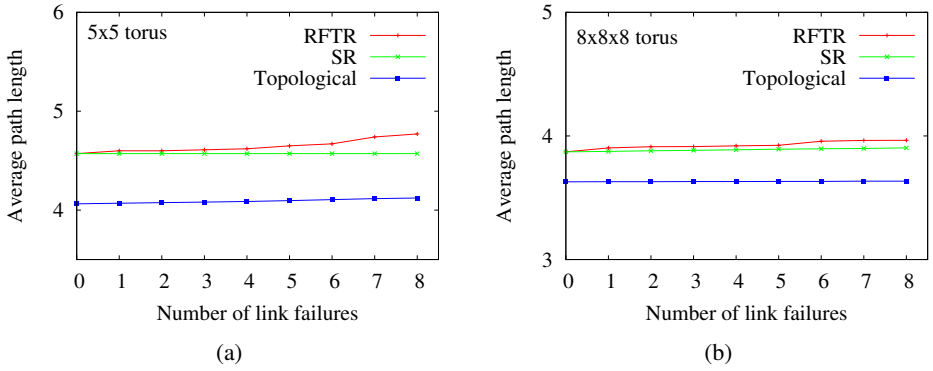


Figure 8.10: Average path length when using RFTR and SR.

8.3.3 Resources Required

Table 8.9 also shows the number of resources required by RFTR, TFTR, and SPFTR. As can be seen, only 2 VLs and up to 6 SLs are required by RFTR to deal with 4 faults in the 2D torus network. However, a third VL is required from 5 faults upward, also increasing the number of SLs up to 8 (dealing with 8 faults). On the other hand, only 2 VLs are required in the 3D torus network, regardless of the considered number of link failures. Further, the maximum number of SLs required is 5. We have corroborated by statistical analysis that resource requirements in larger 2D and 3D tori are the same. From the comparison with other methodologies we can conclude that RFTR is able to achieve a much higher fault-tolerance degree, at the expense of requiring a number of resources slightly greater than the required by TFTR and SPFTR.

We have also evaluated the memory resources required by the Subnet Manager to support the IRT and DRT tables, as they are the most important data structures used by RFTR. Memory requirements directly depend on the number of table entries. As commented in Section 5.2.1, each table provides an entry for every visited switch along each routing path. Therefore, the total amount of entries on each table is defined by the average path length times the number of paths. Table size is, thus, similar to the size required to store all the routing paths at switches. As shown in Figures 8.10.a and 8.10.b, average path length depends on the number of link failures. However, its increment with the number of link failures is not significant. Hence, in practical terms, it would reasonably allow us to define table size independently of the number of link failures. Total amount of memory required by RFTR to support both reachability tables is shown in Table 8.10.

Topology	Radix	Total Memory for DRT and IRT tables	Topology	Radix	Total Memory for DRT and IRT tables
2D Torus	7	38 KB	3D Torus	3	6350 Bytes
	8	75 KB		4	50 KB
	10	232 KB		6	890 KB
	12	584 KB		7	2661 KB

Table 8.10: Memory requirements.

8.3.4 Computational Cost

Figures 8.11.a and 8.11.b show the computational cost of RFTR (based on an AMD Opteron at 1.4 GHz) for different consecutive link failures, for 2D and 3D torus networks with different sizes, respectively. Plotted values correspond to the average computational time obtained from evaluating 1,000 fault combinations for every number of faults (when the number of combinations is greater than 1,000). Confi-

dence intervals are plotted. As can be observed, the time required by RFTR is very low (lower than one second), and linearly increases with the number of link failures. This is because as the number of faults increases, the number of affected paths increases too. The low computational time exhibited by RFTR contrasts with the one exhibited by the reconfiguration technique. In particular, assuming *up*/down** routing (as the algorithm used every time a fault is detected), the time required to compute the new routing tables dramatically increases with network size, as can be seen in Figures 8.11.c and 8.11.d for 2D and 3D tori, respectively. These figures show the average computational time for all the fault combinations of one link.

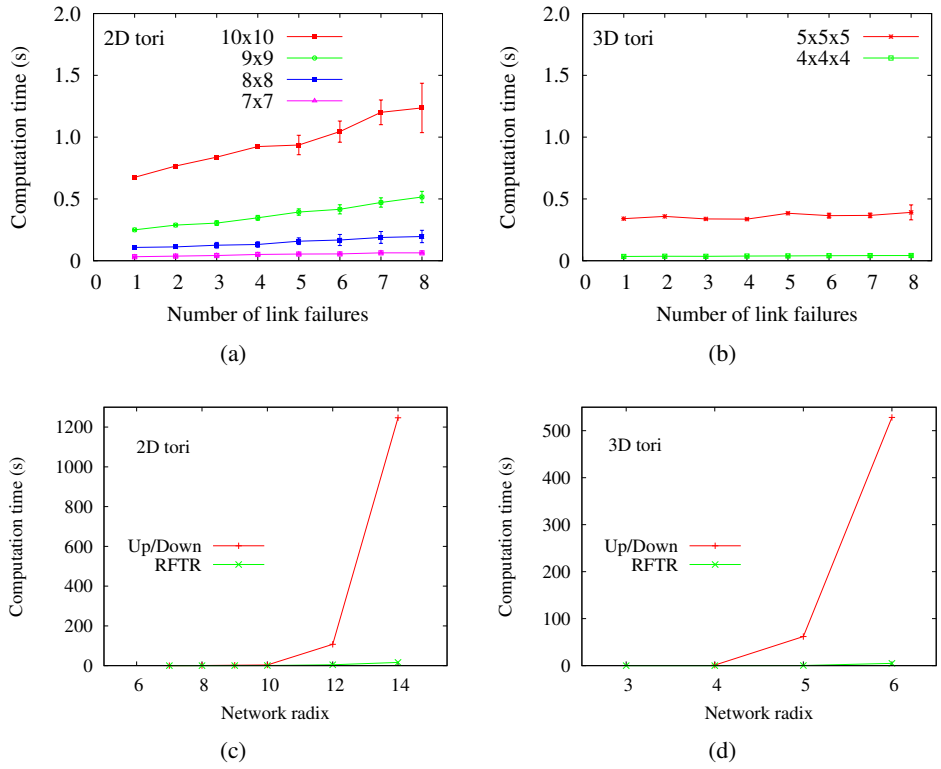


Figure 8.11: Average computational time (a,b) for RFTR and for different number of failures in 2D and 3D tori, (c,d) for *up*/down** and RFTR in 2D and 3D tori with one failed link.

8.3.5 Network Throughput

In this section we analyze how RFTR influences network performance for different number of link failures. In particular, we evaluate the performance degradation after

the occurrence of a certain sequence of faults. For comparison purposes, we also evaluate the performance achieved by the SR reconfiguration process launched after every fault. In order to maximize performance, we assume that the reconfiguration process computes a new spanning tree (required by *up*/down** routing) by selecting the switch with the lowest average distance to the rest of switches as root.

We have run 100 simulations for each number of faults and for each analyzed strategy (RFTR and reconfiguration). In each of them, faulty links were selected randomly. In order to carry out the evaluation of the fault tolerance process, we evaluate the network during all the reconfiguration process: before any link failure occurs, when the fault is injected and detected, during the computation and distribution of the new routing paths, and until all the transient effects of the fault-tolerant method disappear. Figure 8.12 shows the scheme of the different events evaluated into the simulator.

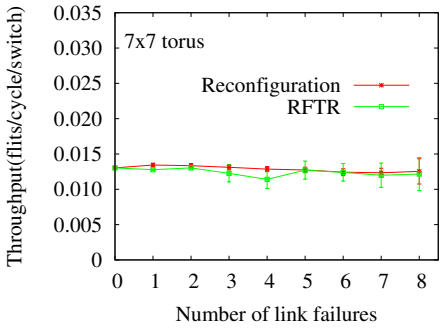
F.I.: Fault injection
D: Fault detection

Figure 8.12: Events of fault detection, computation and distribution of the new routing tables evaluated.

The simulation also considers the required control traffic. As previously described in Section 5.4, a special routing mechanism (Directed-Route) is used to forward control packets. In particular, we modeled the new routing information distributed with control packets, and with ACK packets (since control packets can be discarded).

Average throughput and confidence intervals can be seen in Figure 8.13. As can be observed, throughput decreases in both cases as the number of faults increases. However, it is to note that the difference is negligible. This is a good result because, applying our methodology (RFTR), we are achieving similar network performance to the one provided when applying reconfiguration, but without needing to completely compute all the routing tables (indeed, modifying the routing algorithm by updating only some routing table entries).

Notice that the *up*/down** routing algorithm has been used to compute the initial paths of RFTR. This algorithm exhibits a high computational time because it searches for a set of paths that provides the best traffic balance in the network, thus achieving high throughput. By using RFTR in combination with *up*/down** we can still guarantee high network throughput in the presence of faults with a very low computational time.



(a)

Figure 8.13: Network throughput degradation.

Finally, we are interested in analyzing RFTR performance during the entire process, starting when the fault is injected until routing tables are updated. In particular, we analyze how the network traffic is temporarily affected by the process of detecting the fault, computing new paths, updating forwarding and SLtoVL tables at switches, and notifying the new LIDs to all the end nodes. For comparison purposes, we also analyze the transient network behavior when applying SR.

Figure 8.14.a shows the accepted and lost traffic over time for an injection rate near saturation when a fault is injected and RFTR is applied. On the other hand, Figure 8.14.b shows the traffic evolution for the same injection rate when the same fault is injected and the SR reconfiguration process is applied. Figure 8.14.c shows the accumulated number of lost bytes along the process.

The meaning of the vertical lines is the following: the first one represents the time when the fault is injected and the time when the SM receives the fault detection notification (only one line for both times, because notification happens 18.25 μs after the injection of the fault, and the difference is too small to be represented with two lines in the plot). The second line represents the time when the SM finishes the computation of the new routing information and the time when all switches and end nodes have received all the new routing information (only one line because the information is received 14.60 μs for RFTR, and 66.46 μs for Simple Reconfiguration, after computation process finishes, and this is also a small difference to be represented with two lines in the plot). These computational times required by RFTR and Simple Reconfiguration have been estimated from real experiments (computed on a AMD Opteron at 1.4 GHz).

The times between injection and detection of the fault are the same in RFTR and Simple Reconfiguration, because both simulations assume the same traffic rate, the same frequency of sweeps for fault detection, and the fault is injected at the same time in both simulations.

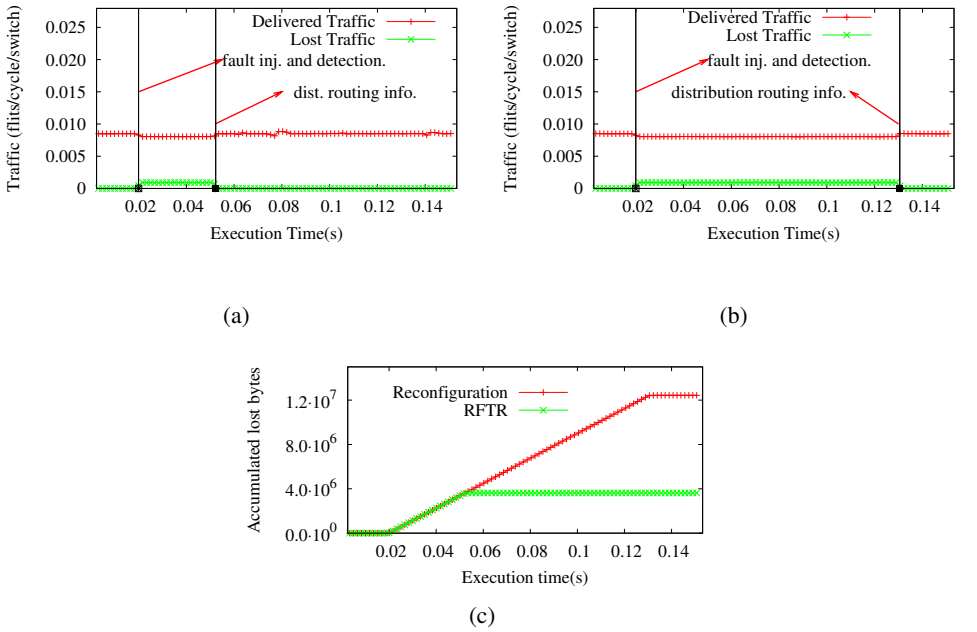


Figure 8.14: (a,b) Accepted traffic when using RFTR and Simple Reconfiguration algorithms, respectively. (c) Accumulated number of lost bytes using RFTR and Simple Reconfiguration algorithms.

We have observed that the time between the end of the computation of the new routing information and the end of the global process remains constant for RFTR, independently of the traffic rate. However, for Simple Reconfiguration it increases with the traffic rate because it requires to solve all dependencies with old traffic. For the evaluation we have considered that control packets are sent in an independent virtual channel (VL15) with the highest routing priority.

Also, it is to note that the useful traffic is not significantly affected by the process. It has to be noted that the shown case corresponds to a worst case for RFTR, that is, the failed link is near the root switch of the spanning tree used by *up*/down**, thus affecting a large percentage of paths.

Figure 8.15.b shows a 7×7 torus network with the routing restrictions defined by *up*/down**. Also, in this figure, a link is marked as failed. For such cases, using RFTR, the amount of routing information sent depends on the number of paths affected by the fault. In this sense, Figure 8.15.a shows the percentage of routing information sent to every switch after the occurrence of a link failure in a 7×7 torus. As before, we are considering a worst case with regard to the link failure selection. Also, Figure 8.15.a shows the percentage of updated entries in forwarding tables to every switch for a total reconfiguration. We can observe that RFTR sends a very small fraction of routing information compared to the routing information sent by a

full reconfiguration. On average it represents the 19.83 % of the total size of the forwarding tables. Figure 8.15.c shows the percentage of updated entries in forwarding tables to each switch in the network depending on its coordinates in the torus.

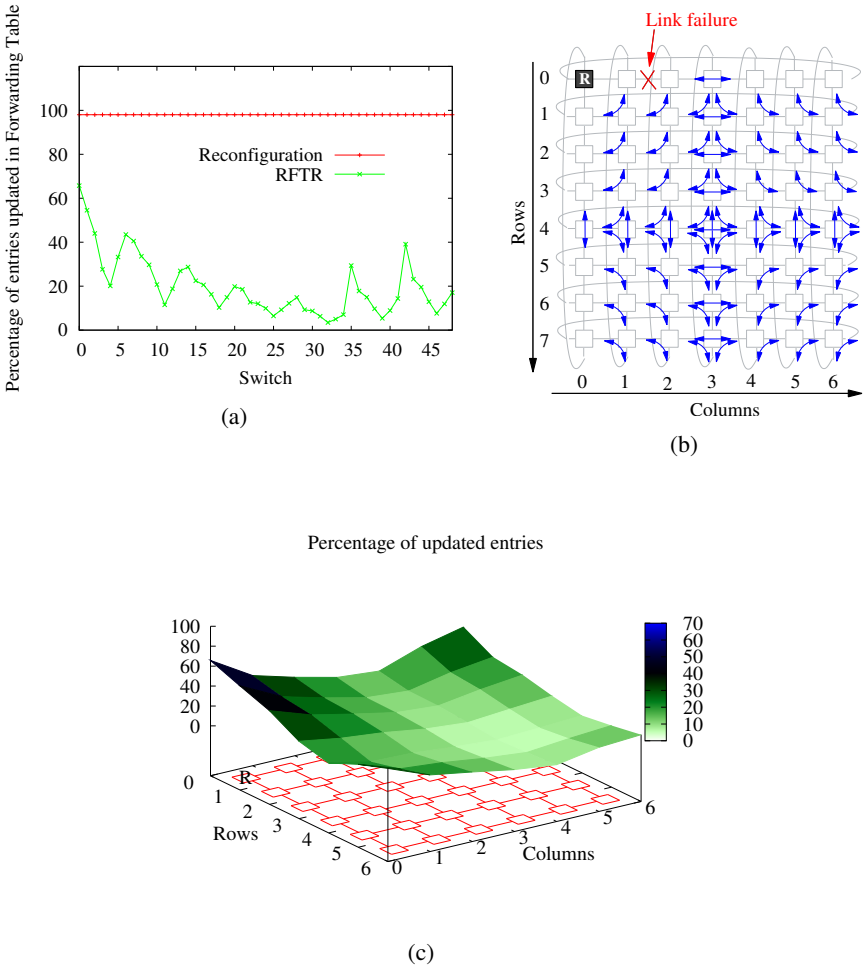


Figure 8.15: Average routing information sent after the first failure.

8.3.6 Summary

TFTR and SPFTR provide a set of alternative disjoint paths for each pair of source-destination pair. Both mechanisms are able to tolerate failures by migrating the affected paths to an alternative disjoint path. All the sets of alternative paths are computed at an initial stage before any fault happens, therefore the mechanisms need to provide all the sets solving all the mapping conflicts and requiring a low number of resources. This evolves to a complex problem when computing all the alternative paths, although only a low percentage of them are used when a fault occurs.

Routing tables computed when applying the RFTR method are less complex, because they only provide one path for each source-destination pair. RFTR only computes the new paths to substitute the affected paths (after the fault occurs). Thus, the strong point of TFTR and SPFTR is their short time for providing a solution. The strong point of the RFTR is that it can tolerate a large number of faults.

Figure 8.16 shows the computation required time for the TFTR, *up*/down**, RFTR, and SPFTR for different tori. Figure also shows the correlated exponential function for the computational cost of each mechanism. The correlation between exponential functions and the measured costs of the mechanisms is almost one. (0.9996, 0.994, 0.996, and 0.995 for TFTR, *up*/down**, RFTR and SPFTR (for radix>9) respectively).

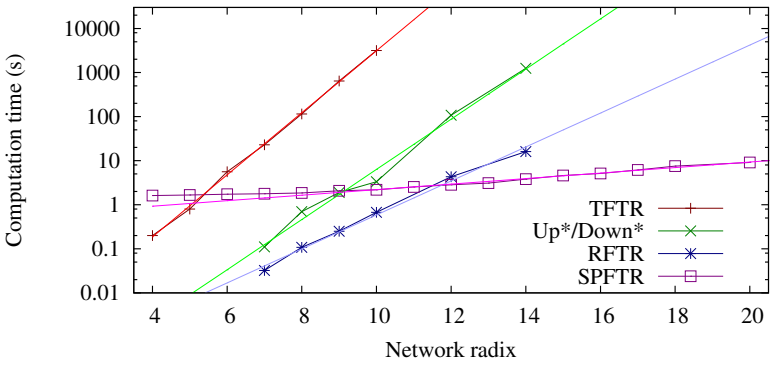


Figure 8.16: Average computational time for TFTR,SPFTR, *up*/down** and RFTR on 2D tori. Figure also shows the correlated exponential function for the computational cost of each mechanism.

8.4 EBR Methodology

To conclude this section, the final method presented in this thesis, EBR, is evaluated.

The EBR mechanism is evaluated in two ways. First, it is evaluated when sending all the routing tables. Second, it is evaluated when sending only the differences between the old and new routing tables. This version is referred to as EBRD (EBR with differences).

We compare the EBR mechanism with the Simple Reconfiguration (SR) and the Double Scheme (DS) mechanisms (for description of both mechanisms, please refer to Section 7.4).

8.4.1 Traffic Patterns and Topologies

For each simulation run, we assume that the packet generation rate is constant and the same for all the end nodes.

Reconfiguration mechanisms are evaluated on different torus networks ranging from 6×6 up to 14×14. Two end nodes are attached to each switch. The different reconfiguration mechanisms are evaluated upon the occurrence of a link failure, under different kind of scenarios, synthetic traffic and real traces.

The different synthetic traffic patterns used are: uniform, bit-reversal, and hotspot. For uniform traffic, each source sends packets to all the destinations with the same probability. For bit-reversal, each source sends traffic only to one end node. The end node is computed by reversing all the bits of the source ID. For hotspot, 10% of the sources (selected randomly) inject traffic to the same destination (selected randomly). The rest of end nodes inject traffic to random destinations.

The traces used are from shared-memory multiprocessors (SPLASH-2). In particular FFT and MP3D applications have been evaluated.

Table 8.11 shows for each scenario the routing algorithms, topologies, and traffic patterns evaluated.

Reconfiguration mechanisms	Topologies	Traffic	Injection rate
SR-PDA, SR-LA, EBR, EBRD, and DS	$6 \times 6 \ldots 14 \times 14$ torus	uniform	low, medium, high
SR-PDA, SR-LA, EBR, EBRD, and DS	10×10 torus	hot-spot and bit-reversal	low, medium, high
SR-PDA, SR-LA, EBR, EBRD, and DS	8×8 torus	FFT, MP3D	scaled to low, medium, high

Table 8.11: Evaluated scenarios for EBR.

In all the scenarios evaluated the detection/notification of faults is accounted for in the simulations. Moreover, routing table computational time is assumed to be zero, which would be the case if tables were precomputed before the reconfiguration event (i.e., for user-directed or planned reconfiguration). This is because we focus only on the reconfiguration process.

As a first scenario, we evaluate all the reconfiguration mechanisms for a random link failure in a 6×6 torus network. In this case, the old and new routing algorithms are the *up*/down**. In both cases the root switch is located at the same switch (0,0). Uniform traffic and three different injection rates are used (LOW, MEDIUM, and HIGH). Figure 8.17 shows the performance achieved by the network for increasing injection rates. The figure also shows the three traffic injection rates analyzed (LOW, MEDIUM, and HIGH).

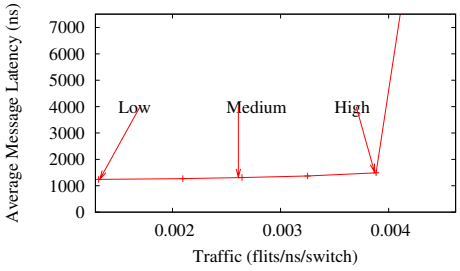


Figure 8.17: Performance and reference traffic injection rates for the 6×6 torus network.

Figure 8.18 shows the reconfiguration time for the 6×6 torus for each reconfiguration mechanism and for different injection rates. In the figure we can see that each mechanism requires roughly the same reconfiguration time regardless of the traffic rate. This is because in all the cases the distribution of routing tables is the process that takes most of the time (tables are sent sequentially) and the amount of information to distribute is the same regardless of the traffic injection rate. Also, it is to note that control packets use the reserved control virtual channel and they have higher priority than data packets.

As can be noticed EBR gets more time to reconfigure the network than DS and SR-PDA. This is due to the sequence used to reconfigure the switches. To explain this, Figure 8.19 shows the time required for distributing four routing tables for two different cases. SM is sending sequentially the routing tables, each table requiring the same injection time (assuming that all the tables have the same size), but each table requires different amount of time to arrive to its destination. In the Figure, the total

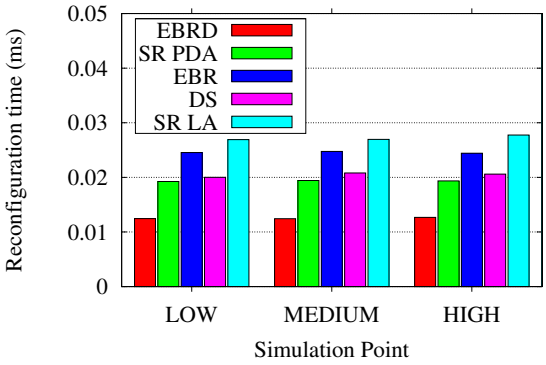


Figure 8.18: Reconfiguration time for 6×6 torus network and uniform traffic pattern.

required time in the first case is lower than the one in the second case shown. This is because in the former case the distribution of tables is better overlapped when SM sends the routing tables (sequentially). Each table requires the same injection time (assuming that all the tables have the same size), but each table requires different amount of time to arrive to its destination. Therefore, total required time can be reduced when overlapping the distribution of tables (sending first tables that take more time to reach their destination). Thus, the order in which tables are distributed affects total reconfiguration time. EBR sends the routing tables firstly to the switches surrounding the fault, and then to the remaining switches ordered by increasing distance to the failed component (see Figure 6.3). Thus, total required time for the reconfiguration process might increase compared with the other reconfiguration mechanisms.

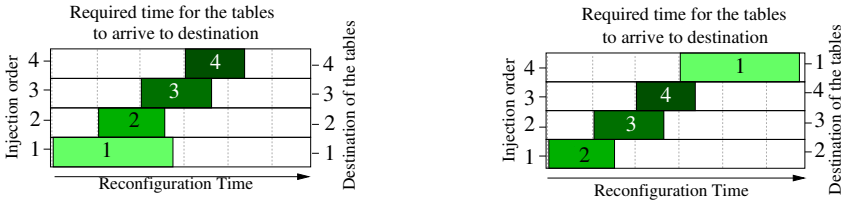


Figure 8.19: Sequence in the distribution of routing tables (may affect the reconfiguration time).

However, two important things must be considered. First, updating only routing tables with the differences between the old and the new ones, may greatly reduce the reconfiguration time. In particular EBRD achieves a reduction factor of 1.5 when compared with DS and SR-PDA.

8.4.2 Dropped Packets

However, the second and most important observation is that in EBR and EBRD switches reconfigure as soon as they get the new routing table. Instead, in the rest of mechanisms all the switches must be updated before using the new routing paths. This has a great impact on the number of dropped packets. As Figure 8.20 shows EBR achieves a lower number of dropped packets when compared with SR-LA and DS. Indeed, differences are much noticeable as traffic rate increases. Also, when using differences (EBRD) results are much better, (similar to SR-PDA).

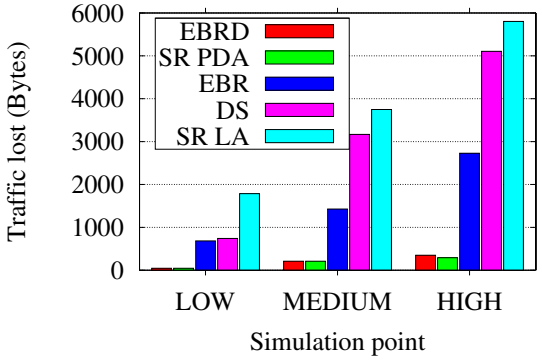


Figure 8.20: Dropped and discarded packets for 6×6 torus network and uniform traffic pattern.

Therefore EBRD achieves the lowest reconfiguration time (a reduction factor of 1.5 with respect to DS and SR-LA) and the lowest number of dropped packets (a reduction factor of 14 with respect to DS and SR-LA for MEDIUM traffic).

We also have evaluated the mechanism with other traffic patterns. Figure 8.21.a shows the traffic lost for a 10×10 torus network for HIGH traffic rate and different synthetic traffic patterns. This figure shows only results for HIGH traffic injection rate, but similar ratio of traffic lost between the different routing mechanisms are obtained when using different traffic injection rates (as shown in Figure 8.20 for uniform traffic pattern). Figure 8.21.b shows the traffic lost for a 8×8 torus network when using traces from shared-memory multiprocessors. It has to be noted that no packet is discarded as a consequence of the application of the deadlock recovery mechanism at any of the evaluated scenarios EBR (and EBRD).

8.4.3 Average Packet Latency

Figure 8.22 shows the evolution of packet latency for HIGH traffic rate. In all plots there is a close up which shows the latency during the reconfiguration, two vertical

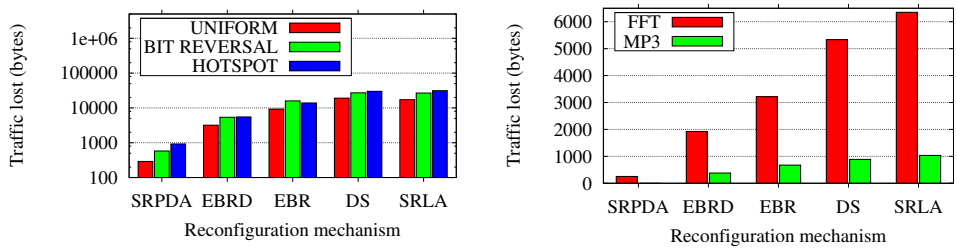


Figure 8.21: Traffic lost when using (a) Synthetic traffic patterns and (b) and traces.

lines show when the fault is detected and when the reconfiguration is finished. As can be noticed, latency exhibited by EBR (also by SR-LA) is not significantly affected during the reconfiguration processes (notice that each plot has a different scale). However, SR-PDA latency increases significantly (is designed to reduce packet dropping and not packet blocking). The reason for this is that SR experiences higher latency due to the blocking introduced by the tokens, and packets blocked by tokens block the packets coming behind, thus introducing contention. This effect is rapidly spreaded, and thus, packets in the network experience higher latencies (this increase in network latency keeps even when the reconfiguration process ends).

It is to note that in all the evaluated scenarios some packets exhibit higher queue latency values. The reason is that these packets are dropped (due to the component failure) and thus, need to be retransmitted from their source. In particular, DS shows high queue latency during the reconfiguration. This is due to the large number of packets lost.

8.4.4 Network Throughput

Figure 8.23 shows the traffic delivered at destinations when using the different reconfiguration mechanisms. As can be noticed for EBR (also by SR-LA) delivered traffic does not suffer any disturbance during the reconfiguration. Also, we can see that during reconfiguration in DS, traffic is accumulated on a virtual channel, while the other is being reconfigured. However, we can notice that in SR-PDA delivered traffic drops to zero during most of the reconfiguration time. This is due to the fact that tokens have many dependencies that block the advance of new packets through the network. Also, we can see at the end of the reconfiguration that the delivered traffic sharply increases, motivated by the fact that traffic has been accumulated at the network and tokens have disappeared.

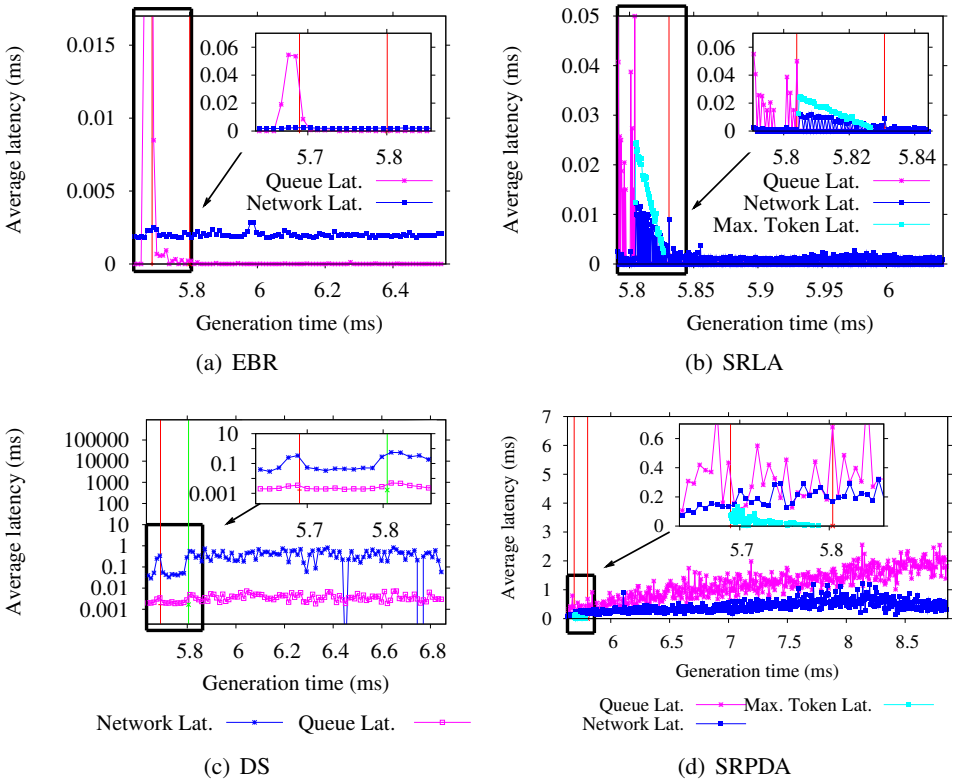


Figure 8.22: Average latency on 10x10 torus network during the reconfiguration and the transient time after it, each plot has a close up which shows the latency during the reconfiguration.

8.4.5 Scalability

Now, we evaluate the mechanisms for different network sizes. Uniform and HIGH traffic rate are considered. Figure 8.24.a shows the reconfiguration time for each mechanism. Notice that as network size increases all the mechanisms tend to exhibit much higher reconfiguration times and even tend to be similar. However, the EBRD mechanism keeps a much lower reconfiguration time, with a reduction factor of 10 for 18×18 networks (notice the logarithmic scale of the X axis). Notice also the very low dropping rate at EBR (and EBRD) (see Figure 8.24.b) with respect to DS and SR-LA. The EBRD mechanism achieves dropping numbers very similar to SR-PDA. In any case the deadlock recovery mechanism has not discarded a single packet. For network throughput and packet latency (not shown) we have obtained the same conclusions as for the first scenario (6×6 torus network).

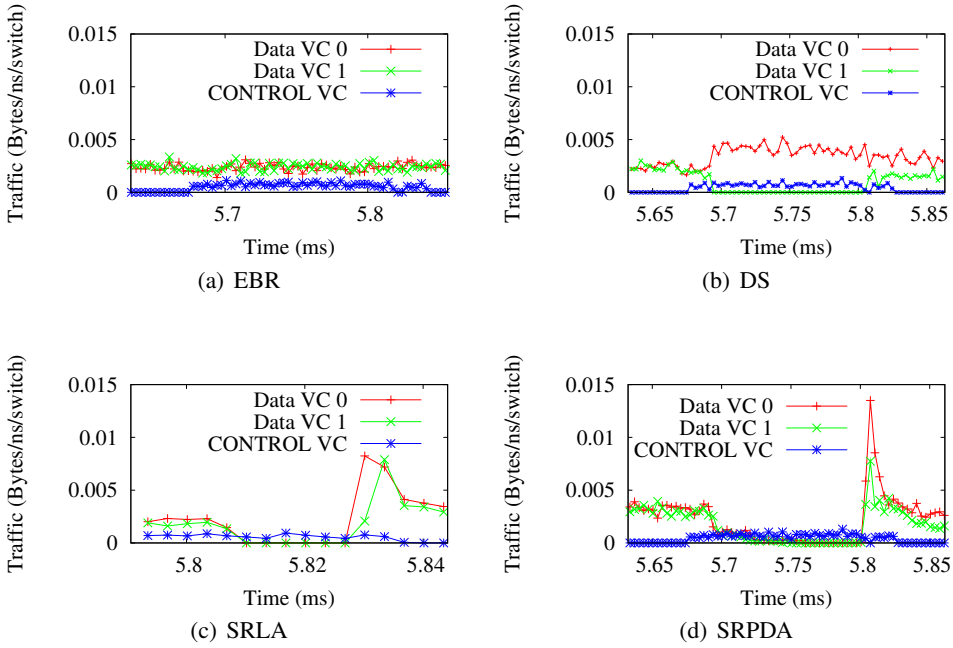


Figure 8.23: Delivered traffic for 10×10 torus network and uniform traffic pattern.

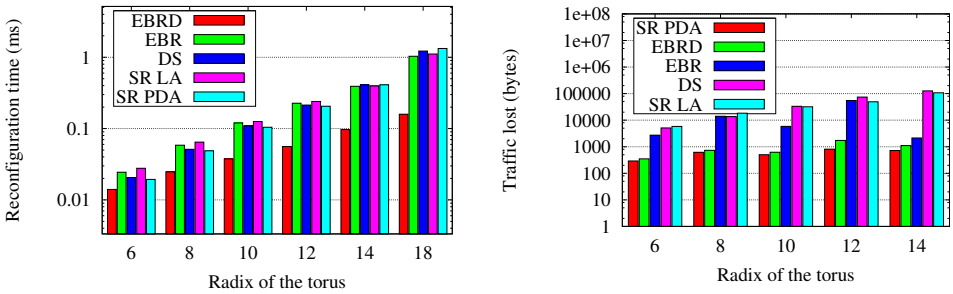


Figure 8.24: Performance results for different tori and uniform traffic pattern, (a) Reconfiguration time and (b) Dropped and discarded packets.

8.4.6 Summary

As an initial conclusion it can be seen that in terms of reconfiguration time and number of dropped packets, EBR (and EBRD) outperforms DS and SR-LA (SR-PDA is beaten also in terms of reconfiguration time). Only SR-PDA drops a lower number of bytes than EBR and EBRD. But this is because SR-PDA blocks the traffic in order to avoid packets to be routed through the failed link. As we have seen, this has a great impact on packet latency and delivered traffic.

As a summary, Table 8.12 shows the different conclusions we can extract from the evaluations performed. As shown, EBRD mechanism achieves the best numbers in terms of traffic and latency performance, reconfiguration time and number of packets dropped. Also, required resources are very modest.

Reconfiguration mechanism	Resources needed	Reconfiguration time	Traffic perturbance	Latency perturbance	Packets dropped
DS	2 VCs	MEDIUM	NO	NO	HIGH
SR-PDA	tokens	HIGH	YES	YES	LOW
SR-LA	tokens	MEDIUM	NO	NO	HIGH
EBR	epoch bits and recovery mechanism	MEDIUM (LOW EBRD)	NO	NO	LOW

Table 8.12: Summary of the characteristics of the different dynamic reconfiguration mechanisms.

Chapter 9

Conclusions

In this chapter we describe the conclusions of the work done, the publications and the future work.

9.1 Conclusions

There exists a great interest in developing fault tolerance mechanisms, since the failure rate in PC clusters is high. In particular, the time required by fault tolerance mechanisms for tolerating the faults is critical, because as long as it increases, the traffic lost and the perturbance of the application running on the cluster also increases. Additionally, the mechanisms should be transparent for the running applications and require as small number of resources as possible. However, to the best of our knowledge there are not solutions for fault tolerance in PC clusters that can be applied regardless of the network topologies and/or the network technology used.

The aim of this dissertation has been the development of fault tolerance mechanisms for interconnection networks used in PC cluster. Indeed, the main challenges have been to propose novel fault tolerance mechanisms for PC clusters able to provide the maximum fault tolerance capabilities and reduce the traffic lost due to the failures, while still maintaining low the required number of resources (e.g., virtual channels or service levels) and the elapsed time required to handle the fault (e.g., the computation time or the time for distributing new routing information through the network).

In this thesis we aim at providing fault tolerance mechanisms which achieve the previous requirements. Moreover, the proposed mechanisms accomplish two additional requirements. Firstly, the network traffic is never stopped by the mechanisms under any circumstances, and secondly, the proposed strategies try to be transparent to the running applications from the point of view of network performance (e.g.,

throughput, latency, and QoS requirements) as much as possible. That is, their application should hardly affect network performance.

A part of this thesis is dedicated to describe the four fault tolerance mechanisms proposed for PC cluster interconnection networks (i.e., TFTR, SPFTR, RFTR, and EBR), and their application environments. Additionally, we have evaluated all of them under simulation. In particular, without loss of generality, the proposed mechanisms are applied to InfiniBand networks (a part of this dissertation has been dedicated to describe the environment where the mechanisms are evaluated). Therefore we consider this thesis achieves the previously presented objectives.

9.2 Contributions

This dissertation has made the following contributions:

- Firstly, this thesis proposes two new fault tolerant routing strategies with selection of path in the source end node[1]. The mechanisms are referred to as *TFTR* and *SPFTR*. Both mechanisms are based on providing a set of alternative disjoint paths for each pair of end nodes, in such a way that it is guaranteed that the system is deadlock free. The sets of disjoint paths provide a fast solution when a fault is detected, because they are ready to be used without needing to update any routing table (they are already computed and the routing information is already distributed). Despite the fact that TFTR minimizes the number of required resources its number increases as network size increases, thus not being scalable. To overcome this drawback, we propose the SPFTR mechanism, which is scalable because it uses the same number of resources regardless of the network size. The fault tolerance degree of both mechanisms corresponds to the number of provided alternative paths minus one.

- Secondly, this thesis proposes a new fault tolerance mechanism based on routing in stages that is referred to as *RFTR*. The main objective consists of obtaining alternative paths for all the faulty ones in a time-efficient manner and using a bounded number of network resources. The idea consists of computing, in presence of faults, alternative paths joining fragments of existing paths, where the first fragment starts at the source end node, and the last fragment ends at the destination end node. This approach is able to provide, in probabilistic terms, a fault tolerance degree higher than that provided by TFTR/SPFTR. Indeed, the proposed approach can be considered as a network reconfiguration process limited to those switches/routing paths affected by the fault. When compared

[1] End nodes are the source and the destination of messages.

to a full network reconfiguration process, RFTR reduces significantly the number of dropped packets and the time required to manage the fault occurrence situation.

- Lastly, this thesis proposes a new reconfiguration mechanism that is referred to as *EBR*. It can be applied for providing fault tolerance Reconfiguration. With reconfiguration it can be tolerated any number of faults as the network can tolerate, i.e. as long as the network remains connected. It is achieved starting a reconfiguration process once a fault is detected. Evaluation results show that EBR works efficiently in different topologies and with different routing algorithms. When compared with current reconfiguration proposals, EBR always gets the best numbers in all the analyzed parameters (e.g., dropped packets, latency, throughput, reconfiguration time, resources required), thus achieving the good properties of all the reconfiguration mechanisms.

Each of these proposals has advantages and drawbacks. We can conclude when comparing between them that when the number of faults is small, TFTR and SPFTR can provide a fast solution as new paths are already computed an distributed once the fault is detected. TFTR supports any topology. However, these methodologies have limited the number of faults that can be tolerated, which depends on the number of disjoint paths they can provide. Moreover, the TFTR requires an increasing number of resources as the network size increases. However, SPFTR has not this drawback, but, it is limited to be applied only to torus networks.

When the number of faults increases, TFTR and SPFTR can not guarantee to tolerate them, although TFTR could try to dynamically compute new partially disjoint paths at expense of high computational cost[2], which could affect the system performance. However, RFTR can tolerate larger number of faults and its computational cost is small compared with that of TFTR when the number of faults increases, and also smaller than computing a new routing algorithm. Moreover, RFTR requires less number of resources that TFTR when the number of faults increases. On the other hand, the main drawback of RFTR is that the number of required resources increases as the number of faults increases Therefore the maximum number of faults to be tolerated is bounded by the maximum number of resources available.

Finally, when the number of faults requires a larger number of resources than that available for RFTR, EBR appears as the last chance to tolerate them. In particular, EBR tolerates any number of failures without requiring any additional resource except to those required by the deadlock recovery mechanism used.

Table 9.1 shows a summary of the advantages and drawbacks of each proposed methodology.

[2]We have assumed that the set of disjoint routing paths is statically computed by TFTR at the start of the system. This computational costs does not affect the system performance when faults occur.

<table>
<tr><th></th><th colspan="4">Fault tolerance mechanism</th></tr>
<tr><th></th><th>TFTR</th><th>SPFTR</th><th>RFTR</th><th>EBR</th></tr>
<tr><td rowspan="2">Advantages</td><td rowspan="2">Fast solution.
Supports
any topology.</td><td rowspan="2">Fast solution.
Constant and
small number of
required resources.</td><td colspan="2">Supports any topology.</td></tr>
<tr><td>Large number of
tolerated
faults.</td><td>Tolerates any number
of faults.
Not additional
resources*</td></tr>
<tr><td rowspan="2">Drawbacks</td><td colspan="2">Limited to tolerate
as many faults as disjoint paths.</td><td rowspan="2">Increasing number
of faults requires
an increasing
number of resources.
Requires to compute
some new paths.</td><td rowspan="2">Requires to compute
new routing function.</td></tr>
<tr><td>Resources required
increase with
network size and
number of faults.</td><td>Only for
torus networks.</td></tr>
</table>

*Requires a deadlock recovery mechanism that require a certain support on switches.

Table 9.1: Summary of the advantages and drawbacks of the proposals.

We consider that the proposals can be used alternatively since they provide a different balance between fault tolerance capabilities and amount of time required for handling the fault situation. In particular, the mechanism which tolerates a higher amount of faults requires also a higher amount of time. In this sense, the Figure 9.1 shows an orientation about the time required by each methodology (from the detection of the fault until an fault-free alternative routing path is provided, taking into account all the time required to have the new routing path available, such as the computation time of the new routing paths and the routing information updating time). The required time is important because the traffic lost due to the fault depends on it. Therefore, when the number of faults is small it is recommended to use the faster solution, that are the methodologies based on disjoint paths (i.e., TFTR/SPFTR). The methodology based on reachability, RFTR, is recommended as the best option when the number of faults is larger than the number of disjoint paths provided by TFTR/SPFTR. Finally, EBR should be used when the number of faults or the number of resources required by the previous methodologies are too large for tolerating them. Notice, that there is a trade off between the time required for handling the fault and the number of tolerated faults.

The Figure 9.2 shows a different view about the selection process of the most suitable fault tolerant approach depending on the MTBF/MTTR ratio. When the MTBF is equal or smaller than MTTR, that means the number of faults in the network at any time should be, in probabilistic terms, smaller than the n disjoint paths provided by the TFTR or SPFTR (depending on the topology and the available resources). Therefore, such a case is the best scenario for the use of such methodologies. On the other hand, the RFTR methodology is recommended to be used on scenarios where the number of faults is larger than that of the previous scenario. That is, when the number of faults is larger than the number of disjoint paths provided by TFTR or SPFTR. Finally, the EBR can be applied to any scenario independently of the number

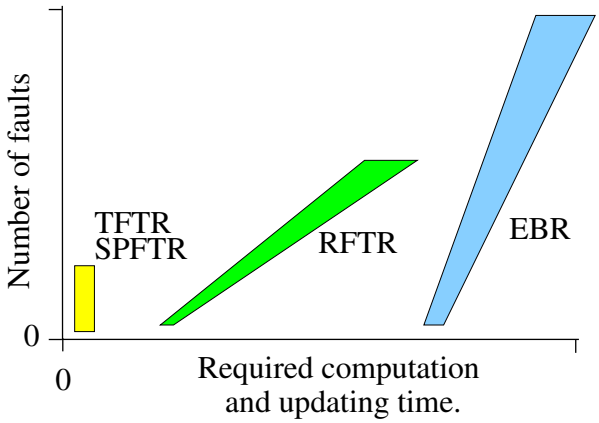


Figure 9.1: Schema for selecting the most suitable fault tolerance mechanism depending on the number of faults and the time required for tolerating them.

of faults. It should be recommended to be used when the number of faults becomes larger than that of the previous scenarios.

However, we also consider that the different proposals can be used complementarily, that is, first the faster mechanisms, which tolerate the smaller number of faults, and later the mechanisms which may require larger amount of time but tolerate a larger number of faults.

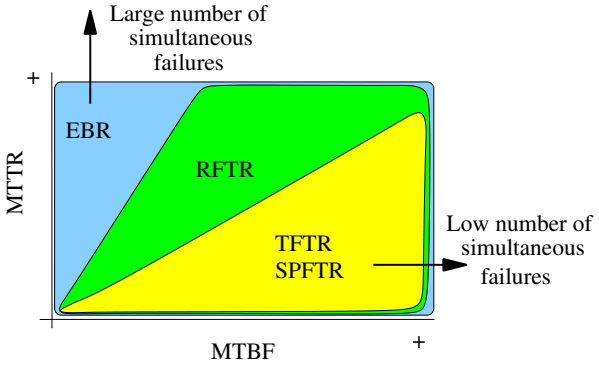


Figure 9.2: Schema for selecting fault tolerance methodology, depending on MTBF and MTTR.

Additionally, all these methodologies could be combined. For example, the RFTR can provide better new routing paths if it starts working with a larger set of paths, such as the set of paths provided by SPFTR. This could provide solutions

which would have less mapping conflicts and require a smaller number of required resources. All the methods can combined, the selection of one of them depends on the total number of faults in the network. Notice, after applying a reconfiguration, (may be using the EBR mechanism), TFTR and RFTR could be applied again, requiring small number of resources, because the underlying routing algorithm should be new.

9.3 Publications

Next, a series of articles published in relation to the work presented in this memory appears:

The publication for the overview before starting the thesis is [115]:

- Líneas de investigación en Tolerancia a Fallos (Research Lines on Fault Tolerance). J. M. Montañana, M. E. Gómez, A. Robles, J. Flich, P López, and J. Duato. In *Proceedings of the Jornadas de Paralelismo XIV*, pages: 425-430 Leganés, Madrid, (Spain). September. 2003, Publisher: ARCOS de la UCIIIM. ISBN:84-89315-34-5.

The publications for the mechanisms based on disjoint paths are [116, 111, 112]:

- Transition-Based Fault-Tolerant Routing Methodology For InfiniBand Networks. J. M. Montañana, J. Flich, A. Robles, P. López, and J. Duato. Workshop on Communication Architecture for Clusters (CAC 2004), In *Proceedings of the* 18^{th} *International Parallel and Distributed Processing Symposium (IPDPS 2004)*, page 186, New México, USA. April 2004, Publisher: IEEE Computer Society Press. ISBN 0-7695-2132-0.

- Providing Fault Tolerance To InfiniBand Networks. J. M. Montañana, A. Robles, J. Flich, P. López, and J. Duato. In *Proceedings of the Jornadas de Paralelismo XV.*, pages 265-270, Almería (Andalucía). September. 2004, Publisher: SP-Universidad de Almería. ISBN: 84-8240-714-7.

- A Scalable Methodology for Computing Fault-Free Paths in InfiniBand Torus Networks.J. M. Montañana, J. Flich, A. Robles, and J. Duato. In *Proceedings of the International Symposium on High Performance Computing (ISHPC-VI 2005)*, pages 79-92, Nara (Japan). September. 2005, Publisher: Lecture Notes in Computer Science, Springer- Verlag. ISBN: 978-3-540-77703-8.

The publications for the mechanisms based on reachability are [113, 114]:

- Substitution-Path-Based Fault-Tolerant Routing Methodology. J. M. Montañana, J. Flich, A. Robles, and J. Duato. In *Proceedings of the CEDI 2005-I Congreso Español de Informática, XVI Jornadas de Paralelismo Actas de las XVI Jornadas de Paralelismo*, pages 173-180, Granada (Spain), 2005, Publisher: Ed. Thomson. ISBN: 84-9732-430-7.

- Reachability-Based Fault-Tolerant Routing. J. M. Montañana, J. Flich, A. Robles, and J. Duato. In *Proceedings of the The Twelfth International Conference on Parallel and Distributed Systems (ICPADS. July 12-15, 2006)*, pages 515-524, Minneapolis (USA) 2006, Publisher: IEEE Computer Society Press. ISBN: 0-7695-2612-8.

The publications for the mechanisms based on reconfiguration are [106, 109, 110, 105]:

Contributions to Conferences:

- Simple Deadlock-Free Dynamic Reconfiguration. O. Lysne, J. M. Montañana, T. M. Pinkston, J. Duato, T. Skeie, and J. Flich. In *Proceedings of the 11^{th} International Conference on High Performance Computing (HiPC)*, pages 504-515, Bangalore (India). December 2004, Publisher: Lecture Notes in Computer Science No 3296, Springer-Verlag 2004. ISBN: 3-540-24129-9.

- EBR: A Deadlock Recovery-Based Reconfiguration Mechanism. J. M. Montañana, J. Flich, and J. Duato. In *Proceedings of the CEDI 2007-II Congreso Español de Informática. XVIII Jornadas de Paralelismo*, pages 229-236, Zaragoza (Spain), 2007. ISBN: 978-84-9732-593-6.

- EBR: A Deadlock Recovery-Based Reconfiguration Network Mechanism. J. M. Montañana, J. Flich, and J. Duato. In *Proceedings of the 22^{nd} IEEE International Parallel and Distributed Processing Symposium (IPDPS. April 14-18, 2008)*, Miami, Florida (USA), 2008. Publisher: IEEE Computer Society Press. ISBN 978-1-4244-1694-3.

Publication on international journals:

- An Efficient and Deadlock-Free Network Reconfiguration Protocol. O. Lysne, J. M. Montañana, J. Flich, J. Duato, T. M. Pinkston, and T. Skeie, *IEEE Transactions on Computers*, vol. 57:(6), pages 762-779, June, 2008

9.4 Future Work

Taking into account that TFTR and SPFTR can provide a fast fault tolerance, but they are limited to provide n disjoint paths, which prevents their application when

the number of faults is equal to or greater than n, we consider that these mechanisms could be combined with RFTR. This mechanism can tolerate larger number of faults, but its response at the occurrence of a fault is not so fast as TFTR/SPFTR because the new alternative routing paths must be dynamically computed. While the alternative paths are being computed, the packets being routed through the faulty link will be dropped, affecting overall system performance. To mitigate this drawback, we suggest, as future work, to try combining both mechanisms, TFTR and RFTR. The idea would be to provide at any time at least two disjoint routing paths for every source-destination pair (or as much disjoint as possible). When a fault occurs, we would immediately switch to an alternative path, so providing a fast response. Meanwhile, new alternative paths could be computed in background, without interfering with the normal system's behaviour. In this way, when the next fault occurs, alternative paths will be already computed, proceeding to quickly replace the old routing paths by the new ones. Therefore, we would combine the advantages of both mechanisms, that is, fast response (TFTR) and high fault tolerance degree (RFTR).

Additionally, we plan to improve the RFTR mechanism by using a set of disjoint paths (the same used by TFTR) instead of only one path for each source-destination pair as the basis to compute additional alternative paths. This could help compose shorter new paths by using a smaller number of resources when a fault is detected. However, the time required by RFTR to handle much more paths as source may increase the computation time of new paths, which should be evaluated. Therefore, we consider we could propose an efficient fault tolerance mechanism which could tolerate a larger number of faults with a small number of required resources. To sum up, the combination of the RFTR and TFTR mechanisms could combine their advantages and reduce their limitations at the same time.

Also, we are considering the development of a version of the EBR for networks with wormhole switching, in order to provide a new reconfiguration mechanism for NoC (Network on Chip), when considering the same goals as we consider for EBR.

Appendix A

Route Patterns for SPFTR

In this appendix we list the patterns obtained in SPFTR. Table A.1 shows the nomenclature used to define movements. For instance, U means movement in the up direction along Y axis, until the Y coordinate of the destination is reached. *1HU* means going one hop in the up direction. *(DST-1)U* means going up until the previous row to the destination.

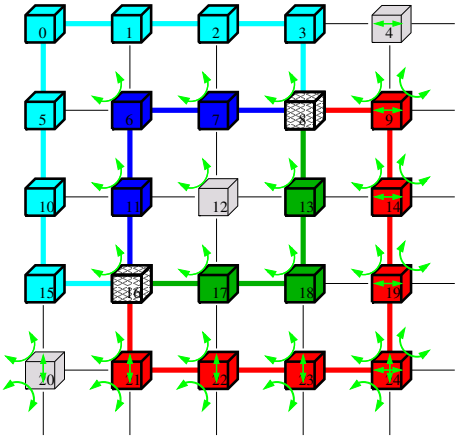
1HU: 1 hop UP	(DST-1)U: UP until Y_{DST-1}	MAX U: UP until the maximum value of y
1HD: 1 hop DOWN	(DST-1)D: DOWN until Y_{DST-1}	MAX D: DOWN until the maximum value of y
1HR: 1 hop RIGHT	(DST-1)R: RIGHT until X_{DST-1}	MAX R: RIGHT until the maximum value of x
1HL: 1 hop LEFT	(DST-1)L: LEFT until X_{DST-1}	MAX L: LEFT until the maximum value of x
U: UP until destination	(DST-2)U: UP until Y_{DST-1}	∅U: UP until y-coordinate equals to 0
D: DOWN until destination	(DST-2)D: DOWN until Y_{DST-1}	∅D: DOWN until y-coordinate equals to 0
R: RIGHT until destination	(DST-2)R: RIGHT until X_{DST-1}	∅R: RIGHT until x-coordinate equals to 0
L: LEFT until destination	(DST-2)L: LEFT until X_{DST-1}	∅L: LEFT until x-coordinate equals to 0
	(DST-3)U: UP until Y_{DST-1}	1U: UP until y-coordinate equals to 1
	(DST-3)D: DOWN until Y_{DST-1}	1D: DOWN until y-coordinate equals to 1
	(DST-3)R: RIGHT until X_{DST-1}	1R: RIGHT until x-coordinate equals to 1
	(DST-3)L: LEFT until X_{DST-1}	1L: LEFT until x-coordinate equals to 1

Table A.1: Nomenclature used to define the sequences of movements.

Figure A.1 shows an example of a route pattern (four disjoint paths are obtained), and also the sequence of movements to compose each path.

Table A.2 list the sequences of movements. Each route pattern is composed by a list four and six sequences of movements for 2D and 3D tori, respectively. Table A.3 shows the complete set of route patterns, where the each route pattern is enumerated.

At the end of this appendix, the route patterns used (identified with a number) from each source to each possible destination for a 7×7 torus.




```
Pattern:  U  R                  Path n:   0 SL:   1 switches:  16 11  6  7  8   .  .  .  .  .  . Forbidden transitions crossed: 0
Pattern:  R U                   Path n:   1 SL:   0 switches:  16 17 18X 13  8   .  .  .  .  .  . Forbidden transitions crossed: 1
Pattern:  1HL  (DST−1)U  R D    Path n:   2 SL:   1 switches:  16 15 10  5  0  1  2  3  8   .  . Forbidden transitions crossed: 0
Pattern:  1HD  (DST+1)R  U L    Path n:   3 SL:   0 switches:  16 21 22 23 24X 19 14  9  8 . Forbidden transitions crossed: 1
```

Figure A.1: Example of route pattern applied.

0	1HU R D	52	1HR D R	104	1HL D
1	R	53	1HU 1HU (DST-1)L U R	105	1HL (DST+1)U R U
2	1HD R U	54	1HR U R	106	1HD ØL U L
3	L	55	ØR D R	107	(DST-1)U L D
4	U	56	1HU (DST+1)U L U	108	ØL U L
5	1HR D L	57	(DST+1)L D L	109	1HR 1HU 1HU (DST-1)L D R
6	D	58	1HR (DST+1)D R U	110	1HL (DST-2)U (DST+1)R D L
7	1HL D R	59	(DST+1)D L U	111	1HL ØU L U
8	1HU R U	60	(DST+1)L U L	112	(DST-1)L U R
9	R D	61	1R D R	113	ØL U R
10	D R	62	1HR U L	114	1HU ØL U R
11	1HL D L	63	MAXD R D	115	1HL ØU R U
12	(DST-1)R D R	64	1HL U L	116	1HR 1HU 1HU L D
13	D L	65	(DST-1)R U R	117	1HD 1HL 1HL (DST-1)U R D
14	L D	66	1HU 1HR 1HR (DST+1)D L U	118	1HR ØU L U
15	U R	67	1HU (DST+1)R D L	119	1HR 1U R D
16	(DST+1)R D L	68	1HU (DST-1)R D R	120	1HD MAXR D R
17	1HD R D	69	(DST-2)R (DST+1)D R U	121	1HU (DST+1)L U L
18	1HL MAXD L U	70	(DST+1)U R U	122	1HD (DST+1)L U L
19	1HD (DST-1)R D R	71	MAXD L D	123	1HU (DST+1)L D L
20	1HU (DST+1)R U L	72	1HD (DST+1)R U L	124	(DST+1)R U L
21	(DST+1)D R U	73	1HL 1HU 1HU R D	125	1HD 1HR 1HR 1HR (DST-1)U L D
22	1HL (DST+1)D L U	74	1HL MAXD R D	126	MAXL MAXD L U
23	1HL U R	75	1HU MAXR D R	127	1HD MAXR (DST-1)U L D
24	R U	76	1HR MAXD L D	128	1HL 1U R D
25	U L	77	1HD (DST-1)L U R	129	1HD 1HL 1HL U L
26	(DST-1)R 1D R D	78	1HL 1HD 1HD R D	130	1HD (DST+1)L 1HU (DST-1)L U R
27	(DST-1)D (DST-1)R D R	79	1HU (DST-1)L D R	131	1HD 1HL (DST-1)D R D
28	L U	80	1HU ØL D L	132	1HD ØL (DST-1)D R D
29	1HU L D	81	1HL (DST-1)D R D	133	ØL (DST-1)U R D
30	1HD L U	82	1HR MAXD (DST-1)L U R	134	1HD MAXR U L
31	1HL 1HD 1HD (DST+1)R U L	83	1HR 1HU ØR D R	135	ØL 1HU 1HU (DST+1)R D L
32	(DST+1)U L U	84	1HU 1HL 1HL (DST+1)D R U	136	1HR 1HU 1HU L U
33	MAXR D R	85	1HU (DST-2)L (DST+1)D R U	137	ØL ØU R U
34	1HL (DST+1)D R U	86	1HR MAXD R D	138	1HR MAXD (DST-1)R U R
35	(DST-1)D R D	87	1HL (DST+1)U L U	139	1HR 1HD (DST-1)R U R
36	1HU L U	88	1HU (DST-2)R (DST+1)D R U	140	1HD (DST-3)L U L
37	1HD L D	89	1HR D	141	1HD 1HL 1HL 1HU L U
38	MAXL D R	90	1HD 1HR 1HR (DST-1)U L D	142	1HL 1HU (DST+1)L U L
39	MAXR U R	91	1HD 1HR 1HR U L	143	1HD (DST-2)L (DST-1)U R D
40	(DST-1)D L D	92	1HL (DST-1)U L D	144	1HL 1HL U L
41	1HD (DST+1)R D L	93	(DST-1)U R D	145	(DST+3)D R D
42	1HL (DST+3)D R D	94	1HR (DST-1)U R D	146	MAXR U L
43	1HL (DST-1)D (DST-1)R D R	95	1HD (DST-1)R U R	147	1HD 1HR (DST-1)D L D
44	1HR 1HD 1HD (DST-1)L U R	96	ØU R U	148	1HR (DST+1)U L U
45	1HU (DST-1)L U R	97	1HL MAXU R D	149	(DST-2)U (DST-1)L U R
46	1HR (DST+1)D L U	98	1HD MAXR U R	150	1HL 1D R D
47	(DST-1)L D R	99	1HR (DST-1)U L D	151	1HL 1HD 1HD (DST+1)R D L
48	1HR (DST-1)D R D	100	1HL (DST-1)U R D	152	(DST+1)U 1HL U L
49	1HR (DST-1)D L D	101	1HD 1HD MAXR U L	153	(DST+1)L 1HU L U
50	1HD (DST-1)L D R	102	1HL 1HU 1HU (DST+1)R D L	154	(DST+1)U 1HL (DST-1)U L D
51	MAXR 1HD 1HD (DST-1)L U R	103	ØU L U		

Figure A.2: Complete set of sequences of movements for obtaining all the patterns for 2D direct networks.

0	0 1 2 3	48	25 65 17 28	96	29 12 59 57	144	123 9 10 22	192	25 54 37 28
1	4 5 6 7	49	29 1 2 3	97	79 58 13 14	145	103 9 10 11	193	15 24 100 106
2	8 9 10 11	50	66 5 6 7	98	88 5 6 7	146	103 52 13 14	194	15 24 132 108
3	8 12 13 14	51	67 9 10 22	99	29 61 59 57	147	70 124 125 23	195	15 24 37 64
4	15 16 17 18	52	0 52 21 11	100	79 69 13 14	148	67 9 10 126	196	25 138 37 28
5	8 9 19 11	53	0 12 21 11	101	8 89 10 11	149	96 9 10 11	197	25 139 37 28
6	20 12 21 14	54	68 69 13 14	102	36 9 19 11	150	96 12 13 14	198	15 24 140 92
7	15 16 17 22	55	68 58 13 14	103	36 61 13 14	151	25 99 98 28	199	107 54 141 142
8	4 5 6 23	56	70 9 10 11	104	15 24 64 17	152	4 62 127 23	200	25 65 37 28
9	15 24 17 11	57	25 39 71 28	105	4 62 90 23	153	93 65 2 64	201	25 65 132 28
10	25 26 27 28	58	15 24 72 73	106	15 24 91 92	154	6 5 4 7	202	4 62 143 23
11	29 1 30 3	59	15 24 72 74	107	93 54 2 60	155	103 33 13 14	203	103 12 13 14
12	0 1 2 31	60	46 75 13 14	108	15 24 72 92	156	67 9 10 115	204	4 62 6 7
13	32 33 13 14	61	67 9 10 34	109	93 54 2 64	157	0 12 13 115	205	70 9 10 60
14	20 9 10 34	62	67 34 10 9	110	25 94 95 28	158	36 62 71 112	206	32 65 13 14
15	8 16 17 7	63	0 52 21 57	111	96 16 17 23	159	8 124 17 23	207	70 9 10 144
16	8 16 35 7	64	66 5 40 38	112	15 24 72 97	160	107 65 2 64	208	93 65 41 28
17	36 5 37 38	65	66 5 37 38	113	25 98 99 28	161	29 3 30 109	209	15 24 145 64
18	25 39 40 28	66	15 76 30 64	114	15 24 72 100	162	68 58 13 128	210	15 24 35 64
19	15 24 41 42	67	25 76 77 28	115	15 24 101 100	163	25 77 76 28	211	25 65 35 28
20	25 24 17 43	68	15 24 72 78	116	0 1 2 102	164	8 124 117 23	212	93 54 41 28
21	29 44 30 3	69	25 65 2 78	117	0 12 21 102	165	107 39 30 60	213	32 39 13 14
22	45 46 14 13	70	29 33 59 11	118	103 52 13 104	166	15 24 106 100	214	20 9 10 105
23	36 5 37 47	71	79 46 13 14	119	67 9 10 105	167	103 5 37 47	215	70 9 10 64
24	45 46 13 14	72	66 5 40 47	120	0 12 13 105	168	8 9 19 57	216	70 146 17 23
25	8 48 10 11	73	80 9 10 34	121	8 9 106 23	169	107 54 30 60	217	25 39 147 28
26	36 33 13 14	74	75 46 13 14	122	25 99 77 28	170	15 24 129 100	218	15 24 41 100
27	36 5 40 47	75	66 5 37 47	123	107 65 2 108	171	25 86 130 87	219	45 148 13 14
28	25 39 37 28	76	36 39 13 14	124	29 109 30 3	172	15 24 35 60	220	8 9 10 64
29	25 49 50 28	77	15 24 41 81	125	0 1 2 110	173	25 54 35 28	221	36 62 37 112
30	29 51 30 3	78	29 82 30 3	126	75 5 37 111	174	67 9 10 111	222	8 146 17 23
31	36 52 13 14	79	29 33 59 57	127	103 5 37 112	175	8 124 35 23	223	25 99 50 28
32	53 46 13 14	80	36 14 13 52	128	8 16 17 113	176	70 124 17 23	224	149 148 13 14
33	32 52 13 14	81	15 24 41 78	129	114 24 41 78	177	93 65 2 110	225	17 24 15 64
34	36 12 13 14	82	15 24 17 64	130	67 9 19 115	178	15 24 131 64	226	25 99 120 28
35	25 54 40 28	83	29 83 30 3	131	25 39 28 71	179	103 39 13 14	227	8 124 17 7
36	25 55 37 28	84	29 52 59 11	132	107 39 30 64	180	32 62 37 112	228	15 24 41 150
37	56 9 10 57	85	84 5 6 7	133	25 116 77 28	181	93 54 2 110	229	0 1 2 151
38	25 58 37 47	86	85 5 6 7	134	4 62 117 23	182	8 124 17 113	230	123 1 2 150
39	45 9 59 57	87	14 46 13 79	135	79 118 13 14	183	8 124 23 17	231	32 62 40 112
40	36 9 10 57	88	85 5 37 47	136	107 54 30 64	184	32 62 132 112	232	32 54 13 14
41	15 24 40 60	89	36 54 13 14	137	117 62 4 23	185	15 24 72 133	233	152 9 10 153
42	25 52 37 28	90	15 24 71 64	138	15 24 106 92	186	15 24 134 100	234	154 24 10 60
43	25 12 37 28	91	25 54 71 28	139	29 109 59 11	187	93 65 2 108	235	15 24 40 64
44	25 12 40 28	92	25 86 77 28	140	103 119 120 11	188	0 1 2 135		
45	25 61 37 28	93	25 86 77 87	141	121 86 77 28	189	29 118 10 11		
46	4 62 6 23	94	0 1 30 3	142	15 24 122 92	190	79 136 13 14		
47	15 24 63 64	95	29 52 59 57	143	25 94 77 28	191	67 9 10 137		

Figure A.3: Complete set of route patterns for 2D direct networks. Each patter contains 4 sequences of movement for composing 4 disjoint paths.

From source 0:

---	0	0	0	0	0	0
1	2	2	2	2	2	3
1	4	4	4	4	5	3
1	4	4	4	4	6	3
1	4	4	4	4	6	3
1	7	7	7	7	5	3
8	9	9	9	9	9	10

From source 1:

11	---	12	12	12	0	0
13	1	14	14	14	14	2
13	1	15	15	15	15	5
13	1	16	16	16	16	2
13	1	16	16	16	16	2
17	1	16	16	16	16	2
18	1	19	19	19	19	20

From source 2:

11	21	---	12	12	0	0
13	22	1	14	14	14	2
13	23	1	15	15	15	2
13	24	1	16	16	16	25
13	24	1	16	16	16	25
26	27	1	16	16	16	2
28	29	1	19	19	19	20

From source 3:

11	21	30	---	12	0	0
31	22	22	1	14	14	2
13	23	23	1	15	15	2
13	23	23	1	16	16	25
13	23	23	1	16	16	25
26	23	23	1	16	16	25
28	29	29	1	19	19	20

From source 4:

11	21	30	30	---	0	0
31	22	32	32	1	14	2
13	23	23	23	1	15	2
13	23	23	23	1	16	25
13	23	23	23	1	16	25
26	23	23	23	1	16	25
28	29	29	29	1	19	20

From source 5:

11	11	21	21	21	---	0
33	34	32	24	24	1	2
33	23	23	23	23	1	2
33	23	23	24	24	1	2
33	23	23	24	24	1	2
31	23	23	23	23	1	2
35	35	36	36	36	1	9

From source 6:

11	11	11	11	11	11	---
37	31	34	34	34	34	1
37	31	38	34	34	34	1
37	31	39	39	39	34	1
37	31	39	39	39	34	1
40	31	31	31	31	34	1
41	42	43	44	44	45	1

From source 7:

46	47	47	47	47	47	48
---	0	0	0	0	0	49
50	51	52	53	53	53	54
50	51	53	53	53	53	55
50	51	53	53	53	53	55
1	56	53	53	53	53	54
8	2	9	9	9	9	34

From source 8:

57	46	58	59	59	58	48
11	---	12	12	12	12	0
60	50	61	62	62	63	53
64	50	61	61	61	61	53
64	50	61	61	61	61	53
65	1	61	61	61	61	53
18	1	15	15	15	15	2

From source 9:

66	67	46	58	68	68	69
11	21	---	12	12	12	0
70	71	50	61	61	61	52
70	72	50	61	61	61	73
70	72	50	61	61	61	73
74	75	50	61	61	61	52
76	23	1	77	77	77	2

From source 10:

66	67	67	46	68	68	69
11	21	78	---	12	12	0
79	71	71	50	61	61	52
70	75	75	50	61	61	53
70	75	75	50	61	61	53
74	75	75	1	61	61	53
80	23	23	1	81	81	2

From source 11:

66	67	67	67	46	68	69
11	21	78	21	---	12	0
79	71	71	71	50	61	52
70	75	75	75	50	61	53
70	75	75	75	50	61	53
74	75	75	75	1	61	53
80	23	23	23	1	81	2

From source 12:

66	67	67	67	67	46	82
11	11	78	83	83	---	0
84	71	71	71	71	85	73
84	71	71	71	71	86	73
84	71	71	71	71	86	73
84	87	87	88	88	1	56
89	23	23	23	23	1	9

From source 13:

90	91	92	93	93	93	46
94	11	11	11	11	11	---
51	95	96	96	96	97	98
51	95	96	99	99	97	98
51	95	96	99	99	97	98
51	95	96	96	96	100	98
101	102	31	34	34	103	1

From source 14:

46	104	47	47	47	47	48
105	106	107	108	108	109	110
---	0	0	0	0	0	49
50	51	52	53	53	53	55
50	51	52	53	53	53	55
50	51	52	53	53	53	54
8	111	111	111	111	9	44

From source 15:

57	46	112	112	112	112	82
113	105	114	115	115	114	109
11	---	116	116	116	0	0
74	50	61	117	117	117	53
65	50	61	117	117	117	53
74	50	61	117	117	117	53
118	8	119	119	119	119	120

From source 16:

57	67	46	59	59	82	121
113	122	105	114	114	114	123
11	124	---	125	116	116	0
70	71	50	61	61	117	53
74	75	50	61	61	117	53
74	71	50	61	61	61	53
126	127	1	128	129	130	2

From source 17:

131	67	67	46	68	68	82
132	133	133	134	58	58	109
11	124	78	---	12	0	0
70	71	71	50	61	61	53
70	71	75	50	61	61	53
74	71	75	50	61	61	53
126	127	135	1	16	16	2

From source 18:

131	67	67	67	46	68	82
132	133	133	133	134	58	109
11	124	78	124	---	0	0
70	71	71	71	50	61	53
70	71	71	71	50	61	53
74	71	71	71	50	61	53
126	127	135	135	1	16	2

From source 19:

91	67	67	67	67	46	82
136	136	122	122	122	137	138
11	11	11	11	11	---	0
84	71	71	71	71	85	73
95	139	71	71	71	85	73
95	71	71	71	71	85	73
140	127	135	135	135	1	2

From source 20:

90	91	141	93	93	93	46
142	136	136	136	136	143	134
94	11	11	11	11	11	---
144	84	97	97	97	97	85
144	84	97	97	97	97	85
144	84	97	97	97	97	85
145	146	146	146	146	146	1

From source 21:

46	47	47	47	47	47	48
105	147	107	108	108	109	110
105	108	107	107	107	109	110
---	107	107	107	107	109	107
50	51	51	52	52	52	55
50	148	52	53	53	53	55
8	111	111	111	111	149	150

From source 22:

57	46	59	59	59	59	82
151	105	115	115	115	114	109
151	152	114	114	114	114	153
11	---	116	116	116	0	0
74	50	61	117	117	117	53
74	154	61	61	61	61	53
155	1	156	130	130	130	157

From source 23:

57	158	46	159	159	159	82
132	122	105	114	114	114	160
132	122	134	114	114	114	153
11	161	---	116	116	116	0
70	71	50	61	61	61	162
74	71	50	61	61	61	53
31	135	1	156	130	130	3

From source 24:

57	158	163	46	68	68	82
132	122	122	134	114	164	138
165	122	122	134	114	114	166
11	21	78	---	12	0	0
70	71	71	50	61	61	53
74	71	71	50	61	61	53
155	167	167	1	16	168	2

From source 25:

57	158	67	67	46	68	82
132	122	122	122	134	164	138
165	122	122	122	134	114	166
11	21	78	124	---	0	0
70	71	71	71	50	61	53
74	71	71	71	50	61	73
155	167	167	167	1	168	2

From source 26:

57	158	67	67	67	46	82
132	122	122	122	122	164	138
169	122	122	122	122	134	170
11	11	124	124	124	---	0
84	71	71	71	71	85	73
84	71	71	71	71	85	73
140	167	167	135	135	1	2

From source 27:

90	91	171	93	93	93	46
142	136	136	136	136	143	134
142	136	136	136	136	143	134
94	11	11	11	11	11	---
144	84	97	97	97	97	85
144	84	97	97	97	97	85
145	146	146	146	146	146	1

From source 28:

46	47	47	47	47	47	48
105	147	107	108	108	109	110
105	108	107	107	107	109	110
105	108	107	107	107	109	110
---	107	107	107	107	109	107
50	148	52	53	53	53	55
8	111	111	111	111	149	150

From source 29:

57	46	59	59	59	59	82
151	105	115	115	115	114	109
151	152	114	114	114	114	153
151	152	114	114	114	114	153
11	---	116	116	116	0	0
74	154	61	61	61	61	53
155	1	156	130	130	130	157

From source 30:

57	158	46	159	159	159	82
132	122	105	114	114	114	160
132	122	134	115	115	114	153
132	122	134	115	115	114	153
11	161	---	116	116	116	0
74	71	50	61	61	61	53
31	135	1	156	130	130	3

From source 31:

57	158	163	46	68	68	82
132	122	122	134	114	164	138
132	122	122	134	114	114	138
132	122	122	134	114	114	166
11	21	78	---	12	0	0
74	71	71	50	61	61	73
155	167	167	1	16	168	2

From source 32:

57		158		67		67		46		68		82
132		122		122		122		134		164		138
132		122		122		122		134		114		138
132		122		122		122		134		114		166
11|	21|	78|	124|	---|	0|	0
74		71		71		71		50		61		53
155		167		167		167		1		168		2

From source 33:

57		158		67		67		67		46		82
132		122		122		122		122		164		138
169		122		122		122		122		134		166
169		122		122		122		122		134		170
11|	11|	124|	124|	124|	---|	0
84		71		71		71		71		85		73
140		167		167		135		135		1		2

From source 34:

90		91		171		93		93		93		46
142		136		136		136		136		143		134
142		136		136		136		136		143		134
142		136		136		136		136		143		134
94|	11|	11|	11|	11|	11|	---
144		84		97		97		97		97		85
145		146		146		146		146		146		1

From source 35:

46		82		82		82		82		82		48
46		108		108		108		108		109		110
46		172		107		107		107		109		110
46		172		107		107		107		109		173
46		172		107		107		107		109		173
---|	0|	0|	0|	0|	0|	49
1		174		174		174		174		149		150

From source 36:

28		46		175		176		176		176		82
28		46		114		114		114		114		109
28		105		114		177		177		114		109
28		105		114		177		177		114		178
28		105		114		177		177		114		178
11|	---|	116|	116|	116|	0|	0
179		1		156		156		156		156		157

From source 37:

28		180		46		159		159		159		82
151		122		46		114		114		114		160
132		122		134		114		181		114		153
132		122		134		114		114		177		153
132		122		134		114		181		177		153
11|	124|	---|	116|	116|	116|	0
179		135		1		156		156		156		157

From source 38:

57		180		180		46		182		183		82
132		180		122		46		164		164		123
132		184		122		134		185		185		138
132		122		122		134		186		186		187
132		122		122		134		186		186		187
11|	124|	124|	---|	188|	0|	0
189		135		190		1		191		191		2

From source 39:

57		180		180		180		46		183		82
132		180		180		122		46		164		123
132		184		122		122		134		185		138
132		122		122		122		134		186		187
132		122		122		122		134		186		187
11|	124|	124|	124|	---|	0|	0
189		135		190		135		1		191		2

From source 40:

192		180		180		180		180		46		82
192		180		180		180		180		46		193
192		122		122		122		122		134		166
192		122		122		122		122		134		194
192		122		122		122		122		134		194
11|	11|	124|	124|	124|	---|	0
145		135		135		135		135		1		2

From source 41:

195		192		196		197		197		197		46
198		199		196		200		200		192		46
198		169		143		143		143		201		202
198		136		136		169		169		201		134
198		136		136		169		169		201		134
94|	11|	11|	11|	11|	11|	---
145		146		146		203		203		203		1

From source 42:

204		205		205		205		205		56		206
204		207		82		82		82		82		192
46		82		82		82		82		208		48
46		209		210		210		210		208		211
46		209		210		210		210		208		211
46		82		212		208		208		208		211
---|	0|	0|	0|	0|	0|	49

From source 43:

213		46		214		214		214		214		215
28		46		176		159		159		176		82
18		46		216		216		216		176		82
217		46		216		216		216		176		82
217		46		216		216		216		176		82
28		46		218		218		218		218		82
11|	---|	116|	116|	116|	116|	0

From source 44:

213		219		204		214		214		220		220
28		221		204		159		159		159		82
18		221		46		222		222		159		82
217		221		46		222		222		159		82
217		221		46		222		222		159		82
217		223		46		218		218		218		82
11|	124|	---|	116|	116|	116|	0

From source 45:

213		224		219		204		14		14		220
28		219		180		204		159		159		225
226		219		223		46		222		77		225
226		221		223		46		222		227		225
226		221		223		46		222		227		225
217		223		223		46		218		228		82
11|	124|	124|	---|	229|	230|	0

From source 46:

213		224		224		224		204		14		220
28		219		180		180		204		159		225
226		219		180		231		46		77		225
226		221		223		223		46		227		225
226		221		223		223		46		227		225
217		223		223		223		46		228		82
11|	124|	124|	124|	---|	230|	0

From source 47:

232		219		219		224		224		204		220
192		89		219		219		219		204		220
35		221		223		223		223		46		82
192		223		223		223		223		46		82
192		223		223		223		223		46		82
192		223		223		223		223		46		82
11|	11|	11|	83|	83|	---|	0

```
 From source 48:
| 233| 232| 206| 206| 206| 206| 204|
| 234| 195| 192| 192| 192| 192| 204|
| 234| 235| 192| 192| 192|  35|  46|
| 234| 235|  35|  35|  35|  35|  46|
| 234| 235|  35|  35|  35|  35|  46|
| 195| 235|  35|  35|  35|  35|  46|
|  94|  11|  11|  11|  11|  11| ---|
```

Appendix B

Acronyms

APM Automatic Path Migration

ASI Advance Switching Interconnect

ATM Asynchronous Transfer Mode.

BFS Breadth First Search

BSC Barcelona Supercomputing Center

CA Channel Adapter

CDG Channel Dependency Graph

CMU Carnegie Mellon University

COTS Commercial-Off-The-Shelf

CPU Central Process Unit

CRC Cyclic redundancy check

CSMA/CD Carrier Sense, Multiple Access / Collision Detect

DFS Depth-First-Spanning tree

DGID Destination port Global ID

DLID Destination Local Identifier

DOR Dimension Order Routing

DRT Direct Reachability Table

DS DoubleScheme

DSM Distributed Shared Memory

EBR Epoch-Based Reconfiguration

FDDI Fiber Distributed Data Interface

FLITS flow control units

GAP Grupo de Arquitecturas Paralelas

HCA Host Channel Adapter

HoL Head of Line

HPC High Performance Computing

ID Identifier

IBA InfiniBand

IBM Intl Business Machines

IBTA InfiniBand Trade Association

IPC Inter Processors Communication

IRT Indirect Reachability Table

ISV Independent Software Vendor

KSR Kendall Square Research

LAN Local Area Network

LANL Los Alamos National Laboratory

LID Local Identifier

LLNL Lawrence Livermore National Laboratory

LMC LID Mask Control

MINs Multistage Interconnection Networks

MIPS Million instructions per second

MTBF Mean Time Between Failures

MTBI Mean Time Between Interrupts

MTTF Mean Time to failures

MTTR Mean Time to restore

MPI Message Passing Interface

MPP Massively Parallel Processing

NIC Network Interface Card

NoC Network on Chip

NOW Network of Workstations

NM Network Manager

NUMA Non Uniform Memory Access

NIC Network Interface Card

O.S. Operating System

PSC Pittsburg Supercomputing Center

QoS Quality of Service

RAID Redundant Arrays of Inexpensive Disks

RFTR Reachability Based Fault Tolerant Routing

SAN System Area Network

SAF Store & Forward

SCSI Small Computers System Interface

SGID Source port Global ID

SISD Single Instruction; Single Data

SIMD Single Instruction; Multiple Data

SL Service Level

SM Subnet Manager

SMA SubNet Manager Agent

SMP Symmetric Multi-Processing

SPFTR Scalable Pattern-based Fault Tolerant Routing

SR Simple Reconfiguration

STP Spanning-Tree Protocol

TFTR Transition Fault Tolerant Routing

TMR Triple Modular Redundancy

UD *up*/down**

UMA Uniform Memory Access

VCT Virtual Cut-Through

VL Virtual Lane

WAN Wide Area Network

WH Wormhole

Bibliography

[1] Ethernet: local area network and devices defined under the standard IEEE 802.3 and the protocol CSMA/CD, 1985.

[2] FDDI standards published by ANSI (American National Standards Institute), the specifications are ANSI X3.139 and X3.148 `http://www.ansi.org/`.

[3] IEEE Standard, "802.3u Media Access Control (MAC) Parameters, Physical Layer, Medium Attachment Units, and Repeater for 100Mb/s Operation, Type 100BASE-T",1995.

[4] More information on BlueGene can be found at the webpage: `http://www.top500.org/wiki/index.php/Blue_Gene`.

[5] Achieving Mainframe-Class Performance on Intel Servers Using InfiniBand Building Blocks: `http://www.oracle.com/technology/deploy/availability/pdf/oracle_IB.pdf`.

[6] Advanced Switching Specifications can be found at the webpage: . www.asi-sig.org.

[7] Amazon webpage. `http://www.amazon.com`.

[8] Barcelona Supercomputing Center webpage. `http://www.bsc.es/`.

[9] Cray XD1 datasheet available at:
http://www.cray.com/downloads/Cray_XD1_Datasheet.pdf.

[10] Earth Simulator webpage. http://www.es.jamstec.go.jp/esc/eng/.

[11] Google webpage. `http://www.google.com`.

[12] Information on Himalaya server can be found at the webpage:. http://www.compaq.com.

[13] Message Passing Interface. Info and standard at: http://www.mcs.anl.gov/mpi.

[14] More information can be found at: http://www.topspin.qassociates.co.uk/solutions-high-performance-computing.htm.

[15] More information HP products for InfiniBand networks can be found at: http://www.hp.com/products1/serverconnectivity/adapters/infiniband/.

[16] More information on ABE Supercomputer can be found at:. http://www.ncsa.uiuc.edu/.

[17] More information on Agilent company can be found at: http://www.agilent.com/.

[18] More information on AOL company can be found at:. http://www.aol.com.

[19] More information on DELL company can be found at: http://www.dell.com/.

[20] More information on Gigabit Ethernet specifications at:. http://www.ieee802.org/3/.

[21] More information on IBM products for InfiniBand networks can be found at: http://www-03.ibm.com/systems/bladecenter/hardware/openfabric/infiniband.html.

[22] More information on InfiniBand at SUN can be found at: http://blogs.sun.com/aland/resource/ib-at-sun.22.02.07.odp.

[23] More information on Intel in InfiniBand Trade Association can be found at: http://www.intel.com/technology/infiniband/.

[24] More information on iWarp can be found at: http://www.cs.cmu.edu/~iwarp.

[25] More information on Mellanox company can be found at: http://www.mellanox.com/.

[26] More information on Network Appliance Joins InfiniBand Trade Association Steering Committee can be found at: http://www.netapp.com/us/company/news/15304311.html.

[27] More information on OpenSM can be found at: http://infiniband.sourceforge.net/SM/OpenSM/index.html.

[28] More information on Silverstorm company can be found at: http://www.silverstorm.com/.

[29] More information on Voltaire in InfiniBand Trade Association can be found at: http://www.voltaire.com/NewsAndEvents/Press_Releases/press2007/September_12_2007_01.

[30] Performance of MPICH-MX 1.2.6..0.9.2c over MX-2G Uniprocessor (UP) case (one process node). http://www.myri.com/scs/performance/Myrinet-2000/MPICH-MX/.

[31] POWER3: The next generation of PowerPC processors. http://www.research.ibm.com/journal/rd/446/oconnell.html.

[32] Resuls HPC Challenge Benchmark are available at the webpage: http://icl.cs.utk.edu/hpcc/hpcc_results.cgi.

[33] SGI Challenge webpage:. http://www.sgichallenge.com/.

[34] Simulation of Hafnium Gate Material http://www-03.ibm.com/servers/deep\-computing/pdf/Simulation_of_Hafnium_Gate_Material.pdf, february 2007.

[35] Stanford Parallel Applications for Shared Memory. http://www-flash.stanford.edu/apps/SPLASH/.

[36] Sun Fire 15000, detailed info can be found at http://sunstuff.org/hardware/systems/other/SunFire15000/.

[37] System availability statistics available at: http://www.nersc.gov/nusers/status/AvailStats/FY08/.

[38] The Cray T3E. detailed info at http://www.cray-cyber.org/systems/t3e.php.

[39] Top500 Supercomputer list can be found on the webpage:. http://www.top500.org.

[40] web page. http://www.cray.com/products/xd1/index.html#RapidArrayInterconnect.

[41] Yahoo webpage. http://www.yahoo.com.

[42] *Efficient Communications for Fine-Grain Distributed Computers. Ph.D. Dissertation*. Southampton University, 1991.

[43] Cluster Interconnects: The Whole Sheband. `http://www.clustermonkey.net//content/view/124/34/1/2/`, April 2006.

[44] N. Adiga, M. Blumrich, D. Chen, et al. Blue Gene/L torus interconnection network. *IBM Journal of Research and Development*, 49, March 2005.

[45] T. Anderson, D. Culler, and D. Patterson. A case for NOW (Networks of Workstations). *IEEE Micro*, 15:54–64, February 1995.

[46] K. V. Anjan and T. M. Pinkston. An Efficient, Fully Adaptive Deadlock Recovery Scheme: DISHA. pages 201–210, 1995.

[47] K. V. Anjan and T. M. Pinkston. DISHA: a deadlock recovery scheme for fully adaptive routing. In *Proceedings of the 9th International Symposium on Parallel Processing*, pages 537–543, April 25-28 1995.

[48] A. Bermúdez, R. Casado, F. J. Quiles, and J. Duato. Handling Topology Changes in InfiniBand. *IEEE Transactions on Parallel and Distributed Systems*, February 2007.

[49] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[50] R. V. Boppana and S. Chalasani. Fault-tolerant wormhole routing algorithms for mesh networks. *IEEE Transactions on Computers*, 44:848–864, July 1995.

[51] R. Casado, A. Bermúdez, J. Duato, F. J. Quiles, and J. L. Sáchez. A protocol for deadlock-free dynamic reconfiguration in high speed local area networks. *IEEE Transactions on Parallel and Distributed Systems*, 12:115–132, 2001.

[52] R. Casado, A. Bermúdez, J. Duato, F. J. Quiles, and J. L. Sánchez. A protocol for deadlock-free dynamic reconfiguration in high-speed local area networks. *IEEE Transactions on Parallel and Distributed Systems*, 12:115–132, February 2001.

[53] R. Casado, A. Bermúdez, F. J. Quiles, J. L. Sánchez, and J. Duato. Performance evaluation of dynamic reconfiguration in high-speed local area networks. In *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture*, 2000.

[54] S. Chalasani and R. V. Boppana. Fault-tolerant wormhole routing in tori. In *Proceedings of the* 8^{th} *International Conference on Supercomputing*, pages 146–155, July 1994.

[55] S. Chalasani and R. V. Boppana. Communication in multicomputers with nonconvex faults. In *Proceedings of Euro-Par'95*, pages 673–684, August 1995.

[56] C. L. Chen and G. M. Chiu. A fault-tolerant routing scheme for meshes with nonconvex faults. *IEEE Transactions on Parallel and Distributed Systems*, 12:467–475, May 2001.

[57] L. Cherkasova, V. Kotov, and T. Rokicki. Fibre channel fabrics: Evaluation and design. In *Proceedings of* 29^{th} *International Conference On System Sciences*, February 1995.

[58] A. A. Chien and J. H. Kim. Planar- adaptive routing: Low- cost adaptive networks for multiprocessors. In *Proceedings of the* 19^{th} *International Symposium on Computer Architecture*, pages 268–277, May 1992.

[59] R. Cole, B. M. Maggs, and R. K. Sitaraman. On the Benefit of Supporting Virtual Channels in Wormhole Routers. In *Proceedings of the Symposium on Parallel Algorithms and Architectures*, pages 131–141. ACM, 1996.

[60] J. Costanzo, L. Crowl, L. Sanchis, and M. Srinivas. Subgraph Isomorphism on the BBN Butterfly Multiprocessor. Butterfly Project Report 14, Department of Computer Science, University of Rochester, Rochester, New York, 14627-0226, October 1986.

[61] C. M. Cunningham and D. R. Avresky. Fault-Tolerant adaptive routing for two-dimensional meshes. In *Proceedings of the first Annual International Symposium on High Performance Computing Architecture*, 1995.

[62] R. Cypher, F. M. auf der Heide, C. Scheideler, and B. Vöcking. Universal algorithms for store-and-forward and wormhole routing. pages 356–365, 1996.

[63] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 4:466–475, April 1993.

[64] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36:547–553, 1987.

[65] W. J. Dally and C. L. Seitz. Deadlock-free Message Routing in Multiprocessors Interconnection Networks. *IEEE Transactions on Computers*, C-36:547–553, May 1987.

[66] W. J. Dally and C. L. SEITZ. Deadlock-free Message Routing in Multiprocessors Interconnection Networks. *IEEE Transactions on Computers*, C-36:547–553, May 1987.

[67] W. J. Dally and B. Towles. *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.

[68] B. V. Dao, J. Duato, and S. Yalamanchili. Configurable flow control mechanisms for fault tolerant routing. In *Proceedings of the* 22^{nd} *International Symposium on Computer Architecture*, pages 220–229, June 1995.

[69] B. V. Dao, J. Duato, and S. Yalamanchili. Dynamically configurable message flow control for faulty-tolerant routing. *IEEE Transactions on Parallel and Distributed Systems*, 10:7–22, January 1999.

[70] D.Reed. High-end computing: The challenge of scale. Director's Colloquium. May 2004.

[71] J. Duato. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, 1993.

[72] J. Duato. A theory of fault-tolerant routing in wormhole networks. In *Proceedings of the International Conference on Parallel and Distributed Systems*, pages 600–607, December 1994.

[73] J. Duato. A theory to increase the effective redundancy in wormhole networks. In *Proceedings of the Parallel Processing Letters*, volume 4, pages 125–138, June 1994.

[74] J. Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 6:1055–1067, October 1995.

[75] J. Duato, B. Dao, P. Gaughan, and S. Yalamanchili. Scouting: fully adaptive, deadlock-free routing in faulty pipelined networks. In *Proceedings of the International Conference on Parallel and Distributed Systems*, pages 608–613, 1994.

[76] J. Duato, O. Lysne, R. Pang, and T. M. Pinkston. Part I: A theory for deadlock-free dynamic network reconfiguration. *IEEE Transactions on Parallel Distributed Systems*, 16(5):412–427, 2005.

[77] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks. An Engineering Approach*. Morgan Kaufmann Publishers, 2003.

[78] T. H. Dunnigan. Early experiences and performance of the Intel Paragon. Technical Report ORNL/TM-12194, Oak Ridge National Laboratory, October 1994. `http://www.csm.ornl.gov/$\sim$dunigan/paragon.ps`.

[79] M. R. Fahey, S. Alam, T. H. Dunnigan, J. Vetter, and P. Worley. Early Evaluation of the Cray XD1. In *Proceedings of the* 47^{th} *Cray User Group Conference*, Knoxville, TN, May 2005.

[80] W. Frawley, G. Piatetsky-Shapiro, and C. Matheus. Knowledge Discovery in Databases: An Overview. pages 213–228, Fall 1992.

[81] J. C. French, T. W. Pratt, and M. Das. Performance Measurement of a Parallel Input/Output System for the Intel iPSC/2 Hypercube. 1991.

[82] P. T. Gaughana and S. Yalamanchili. A family of faulty-tolerant routing protocols for direct multiprocessor networks. *IEEE Computer Transactions on Parallel and Distributed Systems*, 6:482–497, May 1995.

[83] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *Proceedings of the* 19^{th} *International Symposium on Computer Architecture*, pages 278–287, May 1992.

[84] C. J. Glass and L. M. Ni. Fault-tolerant wormhole routing in meshes. In *Proceedings of the* 23^{nd} *International Symposium on Fault-Tolerant Computing*, pages 240–249, June 1993.

[85] M. E. Gómez, J. Duato, J. Flich, P. López, A. Robles, N. A. Nordbotten, O. Lysne, and T. Skeie. An efficient fault-tolerant routing methodology for meshes and tori. 3(1):10–13, July 2004.

[86] M. E. Gómez, N. A. Nordbotten, J. Flich, P. López, A. Robles, J. Duato, T. Skeie, and O. Lysne. A Routing Methodology for Achieving Fault-Tolerance in Direct Networks. *IEEE Transactions on Computers*, 55:400–415, 2006.

[87] R. L. Hadas and E. Brandt. Origin-based fault-tolerant routing in the mesh. In *Proceedings of the First International Symposium on High-Performance Computer Architecture*, pages 102–111, January 1995.

[88] C. T. Ho and L. Stockmeyer. A New Approach to Fault-Tolerant Wormhole Routing for Mesh-Connected Parallel Computers. *IEEE Transactions on Computers*, 53:427–439, April 2004.

[89] InfiniBand Trade Association™. *InfiniBand Architecture specification release 1.2*, October 2004. Available at `http://www.InfiniBandta.com`.

[90] W. H. Inmon. *Building the Data Warehouse*. Wiley, John & Sons, Incorporated.

[91] R. Jain. The art of computer systems performance analysis. *John Wiley & Sons*, 1991.

[92] P. Keleher, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proceedings of the 1994 Winter USENIX Conference*, 1994.

[93] D. J. Kerbyson, A. Hoisie, and H. J. Wasserman. Modeling the Performance of Large-Scale Systems. *IEE Proceedings: Software, Inst. Electrical Engineers*, 150(4):214–221, July 2003.

[94] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3:267–286, 1979.

[95] J. Kim and A.Chien. An evaluation of planar adaptive routing. In *Proceedings of the 4^{th} International Parallel and Distributed Processing Symposium (IPDPS-92)*, 1992.

[96] M. Koibuchi, A.Funahashi, A.Jouraku, and H.Amano. Lturn routing: An adaptive routing in irregular networks. In *Proceedings of the 2001 International Conference on Parallel Processing (ICPP'01)*, pages 374–383, September 2001.

[97] M. Koibuchi, A. Jouraku, K. Watanabe, and H. Amano. Descending Layers Routing: A Deadlock-Free Deterministic Routing using Virtual Channels in System Area Networks with Irregular Topologies. In *Proceedings of the International Conference on Parallel Processing (ICPP'03)*, 2003.

[98] S. Konstantinidou. Adaptive, minimal routing in hypercubes. In *Proceedings of the sixth MIT Conference on Advanced Research in VLSI*, pages 139–153, April 1990.

[99] J. Laprie. Dependable Computing and Fault Tolerance: Concepts and terminology. In *Proceedings of the 15^{th} IEEE International Symposium on Fault-Tolerant Computing*, 1985.

[100] J. Laudon and D. Lenoski. The SGI Origin: A ccNUMA Highly Scalable Server. In *The 24^{th} Annual International Symposium on Computer Architecture*, pages 241–251. ACM, 1997.

[101] T. Lee and J. P. Hayes. A fault tolerant communication scheme for hypercube computers. *IEEE Transactions on Computers*, C-41:1242–1256, October 1992.

[102] M. Levine. Proceedings of the SF's terascale computing system. In *Proceedings of the 7^{th} Workshop on Distributed Supercomputing (SOS7)*, The Lodge at Tamarron, Durango, Colorado, USA, March 2003. `http://www.cs.sandia.gov/SOS7/`.

[103] D. H. Linder and J. C. Harden. An Adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Transactions on Computers*, C-40:2–12, January 1991.

[104] O. Lysne and J. Duato. Fast dynamic reconfiguration in irregular networks. In IEEE Computer Society, editor, *Proceedings of the 2000 International Conference of Parallel Processing*, pages 449–458, Toronto (Canada), 2000.

[105] O. Lysne, J. M. Montañana, J. Flich, J. Duato, T. M. Pinkston, and T. Skeie. An Efficient and Deadlock-free Network Reconfiguration Protocol. *IEEE Transactions on Computers*, 57(6):762–779, June 2008.

[106] O. Lysne, J. M. Montañana, T. M. Pinkston, J. Duato, T. Skeie, and J. Flich. Simple Deadlock-Free Dynamic Network Reconfiguration. In *Proceedings of the 11^{th} International Conference on High Performance Computing (HiPC)*, Bangalore (India), 19-22 December 2004.

[107] O. Lysne and T. Skeie. Load balancing of irregular system area networks through multiple roots. In *Proceedings of 2^{nd} International Conference on Communications in Computing(CIC'01)*, 2001.

[108] D. E. McDysan and D. L. Spohn. *ATM Theory and Application*. McGraw-Hill, New York, 1998.

[109] J. M. Montañana, J. Flich, and J. Duato. EBR: A Deadlock Recovery-Based Reconfiguration Mechanism. In *Proceedings of the CEDI 2007-II Congreso Español de Informática. XVIII Jornadas de Paralelismo.*, pages 229–236, Zaragoza, (Spain), April 2007.

[110] J. M. Montañana, J. Flich, and J. Duato. EBR: A Deadlock Recovery-Based Reconfiguration Network Mechanism. In *Proceedings of the 22^{nd} International Parallel and Distributed Processing Symposium (IPDPS 2008),IEEE Computer Society Press*, Miami, Florida, (USA), April 2008.

[111] J. M. Montañana, J. Flich, A. Robles, and J. Duato. A Transition-Based Fault-Tolerant Routing Methodology for InfiniBand Networks. In *Proceedings of*

the 18^{th} *International Parallel and Distributed Processing Symposium (IPDPS 2004),IEEE Computer Society Press*, New Mexico, USA, April 2004.

[112] J. M. Montañana, J. Flich, A. Robles, and J. Duato. A Scalable Methodology for Computing Fault-Free Paths in InfiniBand Torus Networks. In *Sixth International Symposium on High Performance Computing (ISHPC VI)*, Nara (Japan), September 2005.

[113] J. M. Montañana, J. Flich, A. Robles, and J. Duato. Substitution-Path-Based Fault-Tolerant Routing Methodology. In *Proceedings of the CEDI 2005-I Congreso Español de Informática, Actas de las XVI Jornadas de Paralelismo*, pages 173–180, Granada, (Spain), 2005. Ed. Thomson.

[114] J. M. Montañana, J. Flich, A. Robles, and J. Duato. Reachability-Based Fault-Tolerant Routing. In *Proceedings of the twelfth International Conference on Parallel and Distributed Systems*, pages 515–524, Minneapolis (USA), July 2006. IEEE Computer Society Press.

[115] J. M. Montañana, M. E. Gómez, A. Robles, J. Flich, P. López, and J. Duato. Líneas de investigación en Tolerancia a Fallos (Research Lines on Fault Tolerance). In *Proceedings of the Jornadas de Paralelismo XIV*, pages 425–430, Leganés, Madrid, (Spain), September 2003. ARCOS de la UCIIIM.

[116] J. M. Montañana, A. Robles, J. Flich, P. López, and J. Duato. Providing Fault Tolerance To InfiniBand Networks. In *Proceedings of the Jornadas de Paralelismo XV*, pages 265–270. SP-Universidad de Almería, September 2004.

[117] J. Morrison. The ASCI Q System at Los Alamos. In *Proceedings of the* 7^{th} *Workshop on Distributed Supercomputing (SOS7)*, The Lodge at Tamarron, Durango, Colorado, USA, March 2003. `http://www.cs.sandia.gov/SOS7/`.

[118] T. Mudge, G. Buzzard, and T. Abdel-Rahman. A High Performance Operating System for the NCUBE. In *Proceedings of the Second Conference on Hypercube Multiprocessors*, pages 90–99, 1987.

[119] S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 Network Architecture. *IEEE MICRO*, January-February 2002.

[120] N. Natchev, D. Avresky, and V. Shurbanov. Dynamic reconfiguration in high-speed computer clusters. In *Proceedings of the International Conference on Cluster Computing*, pages 380–387, Los Alamitos (USA), 2001. IEEE Computer Society.

[121] J. M. Nick, B. B. Moore, J. Y. Chung, and N. S. Bowen. S/390 Cluster Technology: Parallel Sysplex. *IBM Systems Journal*, 36(2):172–201, 1997.

[122] M. Noakes, D. Wallach, and W. Dally. The J-machine multicomputer: an architectural evaluation. In *Proceedings of the* 20^{th} *International Symposium Computer Architecture*, pages 224–235, 1993.

[123] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network (QsNet): High-Performance Clustering Technology. In *Proceedings of the* 9^{th} *IEEE Hot Interconnects. (HotI'01), Palo Alto, California*, August 2001. (original version) IEEE Micro January-February 2002 (extended version).

[124] T. M. Pinkston, R. Pang, and J. Duato. Deadlock-free dynamic reconfiguration schemes for increased network dependability. *IEEE Transactions on Parallel and Distributed Systems*, 14(8):780–794, August 2003.

[125] T. M. Pinkston and S. Warnakulasuriya. On deadlocks in interconnection networks. In *Proceedings of the* 24^{th} *International Symposium on Computer Architecture*, June 1997.

[126] D. K. Pradhan. *Fault-Tolerant Computer System Design*. Prentice Hall, February 1996.

[127] V. Puente, R. Beivide, J. A. Gregorio, J. M. Prellezo, J. Duato, and C. Izu. Adaptive Bubble Router: A Design to Improve Performance in Torus Networks. In *Proceedings of the International Conference on Parallel Processing*, pages 58–67, 1999.

[128] V. Puente, J. A. Gregorio, R. Beivide, and F. Vallejo. A Low Cost Fault Tolerant Packet Routing for Parallel Computers. In *Proceedings of the* 17^{th} *International Symposium on Parallel and Distributed Processing (IPDPS)*, page 45.1, Washington, DC, USA, 2003. IEEE Computer Society.

[129] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide. Immunet: A Cheap and Robust Fault-Tolerant Packet Routing Mechanism. In *Proceedings of the* 31^{th} *Annual International Symposium on Computer Architecture*, 2004.

[130] W. Qiao, L. M. Ni, and T. Rokicki. Adaptive-Trail Routing and performance Evaluation in irregular Networks Using Cut-Through Switches. *IEEE Transactions on Parallel and Distributed Systems*, 10:1138–1158, November 1999.

[131] U. Ramachandran, G. Shah, S. Ravikumar, and J. Muthukumarasamy. Scalability Study of the KSR-1. *Parallel Computing*, pages 739–759, 1996.

[132] J. C. Sancho and A. Robles. Improving the Up*/Down* Routing Scheme for Networks of Workstations. In *Proceedings of the Euro-Par*, August 2000.

[133] J. C. Sancho, A. Robles, and J. Duato. In *Proceedings of Conference on High Performance Computing (ISHPC'01)*, September 2000.

[134] J. C. Sancho, A. Robles, and J. Duato. An effective methodology to improve the performance of the up*/down* routing algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 15, August 2004.

[135] J. C. Sancho, A. Robles, J. Flich, P. López, and J. Duato. Effective Methodology for Deadlock-Free Minimal Routing in InfiniBand Networks. 2002.

[136] J. C. Sancho, A. Robles, J. Flich, P. López, and J. Duato. Effective methodology for deadlock-free minimal routing in InfiniBand networks. In *Proceedings of the 2002 International Conference on Parallel Processing, IEEE Computer Society*, 2002.

[137] J. C. Sancho, A. Robles, P. López, J. Flich, and J. Duato. Routing in InfiniBand™ Torus Network Topologies. In *IEEE Proceedings of the International Conference on Parallel Processing*, 2003.

[138] M. D. Schroeder et al. Autonet: A High-speed, Self-configuring Local Area Network Using Point-to-point Links. *Journal on Selected Areas in Comm*, 9(8), October 1991.

[139] S. L. Scott and G. Thorson. Optimized routing in the Cray T3D. In *Proceedings of the Workshop on Parallel Computer Routing and Communication*, pages 281–294, May 1994.

[140] S. L. Scott and G. Thorson. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. In *Proceedings of the Symposium on High Performance Interconnects*, August 1996.

[141] M. Seager. Operational machines: ASCI White. In *Proceedings of the 7^{th} Workshop on Distributed Supercomputing (SOS7)*, The Lodge at Tamarron, Durango, Colorado, USA, March 2003. `http://www.cs.sandia.gov/SOS7/`.

[142] T. Skeie, O. Lysne, and I. Theiss. Layered Shortest Path (LASH) Routing in Irregular System Area Networks. In *Proceedings of the Communication Architecture for Clusters*, 2002.

[143] Y. J. Suh, B. V. Dao, J. Duato, and S. Yalamanchili. Software-based re-routing for fault tolerance pipelined communication. *IEEE Transactions on Parallel and Distributed Systems*, 11:193–211, 2000.

[144] I. T. T. Theiss and O. Lysne. FRoots, A Fault Tolerant and Topology Agnostic Routing Technique. *IEEE Transactions on Parallel and Distributed Systems*, October 2006.

[145] S. C. Woo et al. The Performance Advantages of Integrating Block Data Transfer in Cache-Coherent Multiprocessors. *Proceedings of the 6^{th} International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI)*, October 1994.

[146] T. Woodall, R.L.Graham, R. Castain, D. Daniel, M. Sukalski, G. Fagg, E.Gabriel, G.Bosilca, T.Angskun, J.J.Dongarra, J.M.Squyres, V.Sahay, P. Kambadur, B.Barrett, and A.Lumsdaine. Open MPI's TEG Point-to-Point Communication Methodology: Comparison to Existing Implementations. In *Proceedings of the EuroPVM/MPI*, pages 105–111. Springer Verlag, 2004.

[147] J. Wu. Unicasting in faulty hypercubes using safety levels. In *Proceedings of the 1995 International Conference on Parallel Processing*, volume III, pages 133–136, August 1995.

This page intentionally left blank

Index

A

A-SPFTR, 81
Ack/nack flow control, 16
Adaptive routing, 22
APM, 48, 51
Arbiter, 16
Availability, 39

B

Backtracking routing, 22
Bandwidth, 9

C

Centralized routing, 21
Channel Dependency Graph, 23
Contention, 12
Credit-based flow control, 14
Crossbar, 18

D

Deadlock recovery mechanism, 101
Dependability, 39
Deterministic routing, 22
DFS, 56
Dimension order routing, 25
Direct networks, 17
Direct reachability, 86
Direct reachability table, 86
Disjoint paths, 52
Distributed routing, 21
Distributed Shared Memory, 2
DLID, 48
Double Scheme, 115
Dynamic fault model, 41
Dynamic reconfiguration, 28

E

EBR, 95
End node, 10, 31
Epoch, 101

F

Fault models, 40
Fault region, 43
Fault ring, 43
Fault tolerance degree, 39
Flits, 11
Flow control, 14
Fully adaptive routing, 22

H

Head-of-Line blocking, 60
Header, 11
Hypercube topology, 18

I

IBTA, 29
Indirect networks, 18
Indirect reachability, 86
Indirect reachability table, 87
InfiniBand, 29
Irregular network, 19

L

Lamb node, 44
Latency, 9

M
Mapping conflicts, 34
Massive Parallel Systems, 4
Mesh topology, 18
Message, 11
Message-Passing Interface, 3
Metrics, 118
Minimal routing, 22
Misrouting, 22
Multicast routing, 20
Multicomputers, 3
Multiphase routing, 21
Multistage network, 19

N
Network on Chip, 158
Network reconfiguration, 27
NIC, 10
Non-shared memory multiprocessors, *see* multicomputers
Non-Uniform Memory Access, 2

O
On/off flow control, 15

P
Packet, 11
Partial adaptive routing, 22
Partially disjoint paths, 52
Payload, 11
Phit, 11
Profitable routing, 22
Progressive routing, 22

R
Reliability, 37
RFTR, 83
Round-robin, 16
Round-trip time, 15
Routing restriction, 24

S
Shared medium networks, 17
Shared memory multiprocessor, 2
Simple Reconfiguration, 115
Simulator, 107
Singular case, 40
Smart-routing, 26
Solid fault model, 43
Source routing, 21
SPFTR, 67
SR Latency Aware, 115
SR Packet Dropping Aware, 115
Static fault model, 40
Static reconfiguration, 28
Stop/go flow control, 15
Store & forward, 12
Subnet Manager, 36, 47
Subnet Manager Agent, 47
Sweep, 47
Switching, 12
Symmetric Multiprocessor, 2

T
TFTR, 55
Topologies, 17
Traffic patterns, 111

U
Unicast routing, 20
Uniform Memory Access, 2
Up*/down* routing algorithm, 26

V
Virtual channel, 13
Virtual cut-through, 12

W
Wormhole, 12, 13
Wraparound links, 18

X
XY routing algorithm, 25