Universitat Autònoma de Barcelona

Escola d'Enginyeria

Departament d'Arquitectura de Computadors i Sistemes Operatius

Multipath Fault-tolerant Routing Policies to deal with Dynamic Link Failures in High Speed Interconnection Networks

Thesis submitted by **Gonzalo Alberto Zarza** for the degree of Philosophiae Doctor by the Universitat Autònoma de Barcelona, under the supervision of Dr. Daniel Franco Puntes

Barcelona, July 2011

Multipath Fault-tolerant Routing Policies to deal with Dynamic Link Failures in High Speed Interconnection Networks

Thesis submitted by Gonzalo Alberto Zarza for the degree of Philosophiae Doctor by the Universitat Autònoma de Barcelona, under the supervision of Dr. Daniel Franco Puntes, at the Computer Architecture and Operating Systems Department.

Barcelona, July 2011

Supervisor

Dr. Daniel Franco Puntes

Abstract

English

Interconnection networks communicate and link together the processing units of modern high-performance computing systems. In this context, network faults have an extremely high impact since most routing algorithms have not been designed to tolerate faults. Because of this, as few as one single link failure may stall messages in the network, leading to deadlock configurations or, even worse, prevent the finalization of applications running on computing systems.

In this thesis we present fault-tolerant routing policies based on concepts of adaptability and deadlock freedom, capable of serving interconnection networks affected by a large number of link failures. Two contributions are presented throughout this thesis, namely: a multipath fault-tolerant routing method, and a novel and scalable deadlock avoidance technique.

The first contribution of this thesis is the adaptive multipath routing method *Fault-tolerant Distributed Routing Balancing* (FT-DRB). This method has been designed to exploit the communication path redundancy available in many network topologies, allowing interconnection networks to perform in the presence of a large number of faults. The second contribution is the scalable deadlock avoidance technique *Non-blocking Adaptive Cycles* (NAC), specifically designed for interconnection networks suffering from a large number of failures. This technique has been designed and implemented with the aim of ensuring freedom from deadlocks in the proposed fault-tolerant routing method FT-DRB.

Keywords: Interconnection networks, network fault tolerance, adaptive routing, deadlock avoidance.

Castellano

Las redes de interconexión tienen como uno de sus objetivos principales comunicar y enlazar los nodos de procesamiento de los sistemas de cómputo de altas prestaciones. En este contexto, los fallos de red tienen un impacto considerablemente alto, ya que la mayoría de los algoritmos de encaminamiento no fueron diseñados para tolerar dichas anomalías. Debido a esto, incluso un único fallo de en un enlace tiene la capacidad de atascar mensajes en la red, provocando situaciones de bloqueo o, peor aún, es capaz de impedir la correcta finalización de las aplicaciones que se encuentren en ejecución en el sistema de cómputo.

En esta tesis presentamos políticas de encaminamiento tolerantes a fallos basadas en los conceptos de adaptabilidad y evitación de bloqueos, diseñadas para redes de comunicación afectadas por un gran número de fallos de enlaces. Se presentan dos contribuciones a lo largo de la tesis, a saber: un método de encaminamiento tolerante a fallos multicamino, y una novedosa y escalable técnica de evitación de bloqueos.

La primera de las contribuciones de la tesis es un algoritmo de encaminamiento adaptativo multicamino, denominado *Fault-tolerant Distributed Routing Balancing* (FT-DRB), que permite explotar la redundancia de caminos de comunicación de las topologías de red actuales, a fin de proveer tolerancia a fallos a las redes de interconexión. La segunda contribución de la tesis es la técnica escalable de evitación de bloqueos *Non-blocking Adaptive Cycles* (NAC). Dicha técnica fue específicamente diseñada para funcionar en redes de interconexión que presenten un gran número de fallos de enlaces. Esta técnica fue diseñada e implementada con la finalidad de servir al método de encaminamiento descrito anteriormente, FT-DRB.

Palabras clave: Redes de interconexión, tolerancia a fallos, encaminamiento adaptativo, evitación de bloqueos.

Català

Les xarxes d'interconnexió tenen com un dels seus objectius principals comunicar i enllaçar els nodes de processament dels sistemes de còmput d'altes prestacions. En aquest context, les fallades de xarxa tenen un impacte considerablement alt, ja que la majoria dels algorismes d'encaminament no van ser dissenyats per tolerar aquestes anomalies. A causa d'això, fins i tot una única fallada d'enllaç té la capacitat d'embussar missatges a la xarxa, provocant situacions de bloqueig o, encara pitjor, és capaç d'impedir la correcta finalització de les aplicacions que es trobin en execució en el sistema de còmput.

En aquesta tesi presentem polítiques d'encaminament tolerants a fallades basades en els conceptes d'adaptabilitat i evitació de bloquejos, dissenyades per a xarxes afectades per un gran nombre de fallades d'enllaços. Es presenten dues contribucions al llarg de la tesi, a saber: un mètode d'encaminament tolerant a fallades multicamí, i una tècnica nova i escalable d'evitació de bloquejos.

La primera de les contribucions de la tesi és un algorisme d'encaminament adaptatiu multicamí, anomenat *Fault-tolerant Distributed Routing Balancing* (FT-DRB), que permet explotar la redundància de camins de comunicació de les topologies de xarxa actuals, a fi de proveir tolerància a fallades a les xarxes d'interconnexió. La segona contribució de la tesi és la tècnica escalable d'evitació de bloquejos *Non-blocking Adaptive Cycles* (NAC). Aquesta tècnica va ser específicament dissenyada per funcionar en xarxes d'interconnexió que presentin un gran nombre de fallades d'enllaços. Aquesta tècnica va ser dissenyada i implementada amb la finalitat de servir al mètode d'encaminament descrit anteriorment, FT-DRB.

Paraules clau: Xarxes d'interconexió, tolerància a fallades, encaminament adaptatiu, evitació de bloquejos.

Acknowledgements

Many people have been with me on this journey of four years, where we have shared some endless nights of hard work but also lots of good times.

First and foremost I would like to thank my supervisor Dani Franco for his valuable guidance, patience and peace of mind throughout this four years. I also want to give thanks to Diego Lugones, who has invested a significant amount of time and effort in my projects. My thanks also go to Emilio Luque for trusting me four years ago, and especially for teaching me to walk in this world of science. Special thanks also go to Dolores Rexachs, for her constant and selfless help, and for being always there. I would also like to thank Juan Carlos Moure because he has always encouraged healthy discussions.

I wish to thank all the people in the Simula Research Laboratory for their hospitality during my stay in Oslo, especially to Frank Olaf Sem-Jacobsen, Tor Skeie and Olav Lysne. Furthermore, I want to thank all the people that have influenced my undergraduate studies at the Universidad Nacional del Litoral in Argentina. I would like to thank those professors who were devote to our education even beyond the call of duty. Now I can see the purpose of that hard work, and I wish to thank professors of the Sinc center for the healthy suffering.

I also want to thank Claudia for her love and support, which has truly meant a lot to me. She, together with my dear friends, Alvaro and Zeynep, Moni and Maru, and Hayden and Andrea, have been some of the best gifts I've found in the Old World.

Finally, I thank my parents Alberto and Teresita, and my sisters Andre and Valen, for all their support, understanding and love even though thousands of kilometers separate us.

Contents

List of Figures

List of Tables

List of Algorithms

List of Equations

Chapter 1

Introduction

"We choose to go to he moon in this decade and do the other things, not because they are easy, but because they are hard."

John F. Kennedy

1.1 Parallel Computing

Computer science has become an indispensable tool and a valuable source of knowledge for modern societies, especially in the last few decades. Along these years, computing systems have opened a trend in daily behavior and lifestyle of many people by becoming the engine of an increasing number of essential applications and services. Since then, relations between human societies and computing systems have become remarkably strong and the demand for even more computing power has never stopped.

The steady and undeniable increase in computing power demand highlighted the need for massive parallelization approaches and high-performance computing (HPC) systems. At first, computing power was dedicated almost exclusively to complex and computationally intensive scientific applications. Some well-known examples of these applications are the study and prediction of natural disasters, including earthquakes and tsunami, fire forecasting, etc. Despite this scientific origin, HPC systems have undergone a major expansion in recent years, primarily to serve emerging application areas requiring greater amounts of computing power. These new applications include DNA sequencing, molecular dynamics simulations, weather forecasting, and world-wide banking transactions, among others. Even the simplest Google search is currently based on HPC systems [5].

Clusters of computers, together with massively parallel processing (MPP) systems, have become the two prevailing approaches for achieving parallelism in current high-performance computing systems. Regardless of the approach being used, HPC systems consists of thousands of components, including processing nodes, memory banks, disks, and other peripherals [22]. In this context, the interconnection network emerges as one of the most important components of parallel computers due to its critical role as linking and communication element. Indeed, interconnection networks allow parallel systems operate as large coherent entities.

Nowadays, interconnection networks are expected to provide uninterrupted services but the steady increase in complexity and number of components that make them up leads to significantly higher failure rates. Because of this, and due to the long execution times of computationally intensive applications, many interconnection networks may show a Mean Time Between Failures smaller than the execution time of some applications. This implies the existence of safety threats for sensitive systems involving mission-critical operations, banking, and computation-intensive applications, among others [1]. Consequently, it is extremely important to avoid service interrupts and guarantee the correct finalization of applications, even in the presence of multiple failures.

Clearly, the performance of current systems is closely related to the dependability and robustness of the fault tolerance mechanisms on which they rely. Unfortunately, as explained above, the steady increase in complexity and number of components of interconnection networks significantly increases failure rates. Questions arise from the analysis of this situation such as: how do failures affect communication systems? What kinds of failures appear on real systems? Are those systems able to maintain their operation and performance standards in spite of failure occurrences? If they are not, what should the solution be? What are the best options to achieve fault tolerance and system service continuity? The mere posing of these questions highlights the importance of fault tolerance, and the need to address this problem in current high-performance computing systems.

This thesis is focused on answering these questions with the aim of providing fault-tolerance to interconnection networks, even in the presence of multiple failures.

In order to provide completeness to the scope of the thesis presented along this chapter, we will give now a short introduction into interconnection networks and network fault tolerance.

1.1.1 Interconnection Networks

Interconnection networks, together with the processing units, I/O devices, and memory banks, constitute basic blocks of current computing systems. In general terms, interconnection networks connect the individual components (processing units and memories) of computing systems through a collection of links and switches, where a switch allows a given component to communicate with several other components without having a separate link to each of them. Consequently, the performance of most computing systems is currently limited by their communication or interconnection network (not by their logic or memory). This makes interconnection networks the key factor in the success of current and future computing systems [14].

There is no single criterion for the definition of interconnection networks, as they are currently being used for many different applications. In this thesis, we will consider interconnection networks as high-speed and low-latency networks, used to communicate and link together the components of computer systems through a collection of bidirectional links and switches.

1.1.2 Network Fault Tolerance

The need for network fault tolerance becomes evident when taking into account that some specific features of the network (such as routing) may provide only a single path between a given source and a given destination, in which case any fault of a link or switch along the path will disconnect the source-destination pair. Fault tolerance in networks is often achieved by having multiple paths connecting sources and destinations [48, Ch. 4]. Network fault tolerance may be further divided into two categories: static fault tolerance, for methods that require a network shutdown to perform some sort of static reconfiguration process; and dynamic fault tolerance, grouping methods that do not require information about network faults in advance.

In this thesis, we focus on the design of adaptive fault-tolerant routing algorithms. We have chosen this approach with the aim of providing graceful and wide-range solutions to the problem of network fault tolerance, by taking advantage of the intrinsic path redundancy of some network topologies such as tori and k-ary n-trees. It is worth noting that these topologies are currently among the most extended topologies in HPC systems, according to the TOP500 supercomputer list of November 2010 [95].

1.2 Motivation

The wide range of applications of high-performance computing systems provide great benefits to modern societies, highlighting the need of keeping some of these systems up and running as long as possible. These systems are commonly used for running applications requiring a large volume of data communication. Under these circumstances, communications are based on large and complex high-speed interconnection networks, a situation that leads to considerable increases in network failure rates.

Since interconnection networks are critical components of high-performance computing systems, failures in network devices have terrible impacts on the system. As few as a single link failure has the deadly potential of halting the entire computing system, blocking any running application. Several solutions have been proposed in recent decades, with varying success, but most of them have become obsolete due to advances of the networking technology. Nevertheless, most solutions based on static fault tolerance, such as components redundancy and static reconfigurations are still valid but at the expense of high extra cost in terms of economic resources and time, respectively.

Mechanisms based on the concept of dynamic fault tolerance often achieve far better performance results, since do not require the system to be shut down. As counterpart, implementing these mechanisms in real systems is often difficult because adaptability poses a new an important problem, deadlock occurrences. Notwithstanding, given the performance improvements achievable by dynamic fault tolerance approaches, any proposal capable of providing fault tolerance and also solving the deadlock problem will have incredibly beneficial implications for both users and owners of high-performance computing systems. This is the motivation of this thesis.

1.3 Objectives

The ultimate goal of this thesis is to design, implement and evaluate fault-tolerant routing policies capable of serving interconnection networks, even in the presence of a large number of dynamic link failures. We address this problem by designing adaptive multipath routing policies for exploiting benefits of path redundancy in commonly used network topologies, including tori and k-ary n-trees. We can enumerate the objectives of this thesis as follows:

1. Conduct a study on the fault tolerance theory, in order to analyze the potential application of these concepts to the field of interconnection networks.

2. Conduct a study on the dynamic characteristics of interconnection networks taking into account results obtained in the previous point, in order to identify possible solutions to the problem fault tolerance in interconnection networks.

3. Design and implement adaptive multipath fault-tolerant routing policies capable to tolerate a large number of link failures.

4. Conduct an analysis of problems caused by the occurrence of failures; and evaluate the effectiveness of proposed solutions to the aforementioned problems.

5. Evaluate problems encountered in the previous point in order to avoid possible deadlock situations and propose solutions (if necessary).

6. Analyze the behavior of solutions proposed in points 3 and 5 by means of a simulation-based experimental study, designed to confirm the proper functioning of the proposals.

1.4 Contributions

The contributions of the thesis is directly related to the achievement of the objectives we have outlined in previous sections. To this end, we have designed and implemented fault-tolerant routing policies based on concepts of adaptability and deadlock freedom. In this thesis, we present the following contributions:

- An adaptive multipath routing method for treating a large number of network link failures in HPC systems. The method allows applications to successfully complete their executions by exploiting the redundancy of communication paths available in most network topologies. At the same time, the method treats the congestion problems caused by the occurrences of such failures. This work is based on source-destination path information and consists of three phases. The first phase is responsible for on-line fault diagnosis and uses physical level monitoring at network nodes along the source-destination path. If a message encounters a faulty link along the path, the second phase immediately reroutes that message to the destination through an alternative path. In the third and last phase, the source node is notified about the link failure in order to disable the faulty path, and to establish new alternative paths for the following messages to be sent to that destination.

  This work has followed an incremental development approach based on several extensions and enhancements of the original method. The original method has been published in [102], [106] and [104]. An extension intended maximize the use of system resources by means of detecting and applying differential treatments to permanent and transient faults, has been published in [107], [103] and [109]. In addition, the work present in [107] includes an supplementary evaluation based on the availability traces of parallel and distributed systems obtained from the public failure data repositories CFDR [98] and FTA [29].

- A complete and scalable deadlock avoidance technique specifically designed to deal with large interconnection networks suffering from a large number of dynamic faults. This technique has been designed to ensure deadlock freedom in faulty networks without using virtual channels. The aim of this proposal is to deny the *hold-and-wait* and *circular wait* conditions in order to avoid deadlock occurrences by means of adding an one-slot deadlock avoidance buffer to each input buffer, and applying a simple set of actions when accessing output buffers with no free space.

The first version of this work, including an intuitive functional proof, has been published in [105]. Finally, the fully-enhanced three-stages version of the work has been published in [108]. In addition, a draft containing some of the proposals of this work has been presented to the Spanish Patent and Trademark Office (*Oficina Española de Patentes y Marcas*).

1.5 Research Method

The research in this thesis is oriented to the design, implementation and evaluation of fault-tolerant routing policies; and is framed in the academic program of applied research of the *Universitat Autònoma de Barcelona*.

Throughout this thesis, we have followed the (iterative) hypothetico-deductive method as the methodological scheme for performing the scientific research [16]. The method is based on five major stages:

1. **Existing theories and observations.** Pose the question in the context of existing knowledge, theory and observations.
2. **Hypothesis.** Formulate a hypothesis as a tentative answer.
3. **Predictions.** Deduce consequences and make predictions.
4. **Test and new observations.** Test the hypothesis in a specific experiment/theory field.
5. **Old theory confirmed within a new context or new theory proposed.** When consistency is obtained the hypothesis becomes a theory and provides a coherent set of propositions that define a new class of phenomena or a new theoretical concept.

As a rule, the loop 2-3-4 is repeated with modifications of the hypothesis until the agreement is obtained, which leads to 5. If major discrepancies are found the process must start from the beginning. The results of stage 5 have to be published. Theory at that stage is subject of process of *natural selection* among competing theories. The process can start from the beginning, but the state 1 has changed to include the new theory/improvements of old theory [16].

This thesis share theoretical basis with the method developed and discussed in depth by Franco et al. [25] [27], [28], *Distributed Routing Balancing (DRB)*; and subsequently improved by Lugones et al. [55], [56], [57]. These methods, together with the theory of fault tolerance and interconnection networks, constitute the first stage of the scientific research method of this thesis. Throughout this stage, we have conducted the first study on fault-tolerance and interconnection networks. Books on fault-tolerance by Jalote [41], Abd-El-Barr [1], Koren and Krishna [48], and Kanoun and Spainhower [42], and interconnection networks by Duato et al. [22], Dally and Towles [14], and Hsu and Lin [37] have been particularly useful. Since DRB and its related methods have been discussed in depth in two previous theses, we focus on the design, implementation, and evaluation of novel and complete fault-tolerant mechanisms. In fact, *Stages 2* and *3* comprise the proposal of the fault-tolerant routing policies based on DRB. Then, in *Stages 4* and *5*, we have evaluated and analyzed the effectiveness of the proposed fault-tolerant routing method through simulation, using a standard discrete event simulator. To this end, we have enhanced existing models of network components [54], [58] by including the proposed fault tolerance mechanisms and policies. This has allowed us to develop the simulation models used in the experimentation of our proposals.

1.6 Thesis Outline

According to the objectives and the research method described above, the outline of the remaining chapters of the thesis is as follows.

Chapter 2: Thesis Background.
This chapter introduces some basic concepts about interconnection networks and fault tolerance. Then, presents the related work and exposes some specific concepts about network fault tolerance, fault-tolerant routing and deadlock avoidance.

Chapter 3: Fault-tolerant Distributed Routing Balancing.
This chapter explains in detail the adaptive fault-tolerant routing method *Fault-tolerant Distributed Routing Balancing* (FT-DRB), designed for treating a large number of dynamic failures in interconnection networks.

Chapter 4: Scalable Deadlock Avoidance for Fault-tolerant Routing.

In this chapter, we present the proposed scalable deadlock avoidance technique *Non-blocking Adaptive Cycles* (NAC), specifically designed for interconnection networks affected by a large number of link failures.

Chapter 5: Evaluation of Proposals.

This chapter describes the test scenarios and provides the explanation of experimental results for both proposals, the fault-tolerant routing method (FT-DRB) and the deadlock avoidance technique (NAC).

Chapter 6: Conclusions.

Concludes the thesis and presents the further work and open lines for both the fault-tolerant routing method and the deadlock avoidance technique.

The list of references and one appendix complete the document of this thesis. The Appendix A includes the complementary results of the evaluation presented in Chapter 5.

Chapter 2

Thesis Background

In this chapter, we present some basic concepts about interconnection networks and fault tolerance needed to frame the thesis. First, we introduce a general description of interconnection networks in section 2.1. Then, in section 2.2, we present some concepts of fault tolerance and focus on specific aspects of network fault tolerance (subection 2.2.1), fault-tolerant routing (subsection 2.2.2), and deadlock resolution (subection 2.2.3).

2.1 Interconnection Networks

Interconnection networks can be defined as programmable physical systems consisting of a series of elements (links and switches) that behave and interact with each other for communicating numerous components of computing systems [27]. In computer systems, interconnection networks connect processors to memories and input/output (I/O) devices to controllers; in communication switches, they connect input ports to output ports [14].

Currently, interconnection networks are essential for systems where efficient communication technologies have a direct influence in the overall performance of the system. These systems can be classified into two main groups [96], [97]: *computer clusters*, and *massively parallel processing (MPP)* systems. Computer *clusters* have become the largest of these groups, representing 82.8% of the systems in the TOP500 list of November 2010 [96]. These systems were originally conceived as a platform for implementing parallel applications, even though they have been subsequently used in other application fields such as storage networks and internet services. The other group comprises the MPP systems, such as the *IBM BlueGene/P* supercomputer [38], and represents 16.8% of the TOP500 systems [96]. These systems were the first using high-speed interconnection networks.

Although *cost constraints* and *reliability and repairability* are always important aspects, there are four functional requirements (mostly related to routing algorithms) affecting the design and applicability of interconnection networks: *connectivity*, *deadlock* freedom, *livelock* freedom, *starvation.* Since the problem of deadlock resolution plays an important role in network fault tolerance, we discuss these concepts in detail in subsection 2.2.3.

We will now give an introduction to important features of interconnection networks design related to the thesis, including topologies (subsection 2.1.1), routing (subsection 2.1.2), flow control (subsection 2.1.4), and switching techniques (subsection 2.1.3).

2.1.1 Topologies

The topology of the network refers to how nodes and channels are arranged in the interconnection network. This is probably the most commonly used criterion for the classification of interconnection networks since it defines how nodes and channels are interconnected. Network topologies can be classified in three main groups, as defined by Duato et al. [22, Ch. 1] and Dally and Towles [14, Ch. 3]: *shared-medium networks*, *direct networks*, and *indirect networks.*

Shared-medium Networks. In this kind of networks, the transmission medium is shared by all communicating devices. These networks were used in the first parallel computers but falling in disuse soon due to performance and scalability issues. *Local area networks* and *backplane buses* are commonly adopted within the context of shared-medium networks. A *local area network* is basically a bus or ring network topology that uses copper wires or fiber optics as the transmission medium for interconnecting computers into an integrated parallel and distributed environment. On the other hand, a *backplane bus* is the simplest interconnection structure for bus-based parallel computers. The classification and some examples of these networks are shown in Figs. 2.1 and 2.2, respectively.

Direct Networks. Consist of a set of nodes, each one being directly connected to a (usually small) subset of other nodes in the network. Each node is a programmable computer with its own processor, local memory, and other supporting devices. A common component of these nodes is a router, which handles message communication among nodes. For this reason, direct networks are also known as router-based networks where each router has direct connections to its neighboring routers. Direct networks have been a popular interconnection architecture for constructing large-scale parallel computers. As stated by Duato et al. [22, Ch. 1], direct networks have been traditionally modeled by a graph $G(N, C)$, where the vertices of the graph N represent the set of precessing nodes and the edges of the graph C represent the communication channels. Most direct networks have *orthogonal* topologies where nodes are arranged in an orthogonal $n-$dimensional space, and every link is arranged in such a way that it produces a displacement in a single dimension. There is an additional division within *orthogonal* network topologies; one of the subdivision corresponds to the *strictly orthogonal* topologies, where every node as at least one link crossing each dimension, as in a *n-dimensional mesh* or a *k-ary n-cube*; the other subdivision comprises the *weakly orthogonal* topologies, where some nodes may not have any link in some dimensions, such as a *binary tree*. The corresponding classification and examples are shown in Figs. 2.3 and 2.4.

Indirect Networks. In these networks, the communication between any two nodes has to be carried through some switches. Each node has a network adapter that connects to a network switch. Each switch can have a set of ports. Each port consists of one input and one output link. A (possibly empty) set of ports in each switch is either left open or connected to processors, whereas the remaining ports are connected to ports of other switches to provide connectivity between the processors. The interconnection of those switches defines various network topologies, ranging from *regular* topologies used in array of processors to *irregular* topologies currently used in network of workstations. Regular topologies have regular connection patterns between switches as in the case of *crossbars*[1] and *MINs*[2]; while *irregular* topologies do not follow any predefined pattern. As in the case of direct networks, indirect networks can also be modeled by a graph $G(N, C)$, where N is the set of switches and C is the set of unidirectional or bidirectional links between the switches [22, Ch. 1]. The classification and some examples of this kind of networks are shown in Figs. 2.5 and 2.6, respectively.

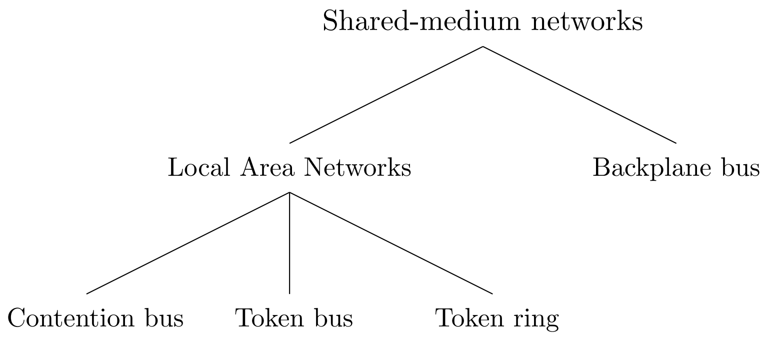
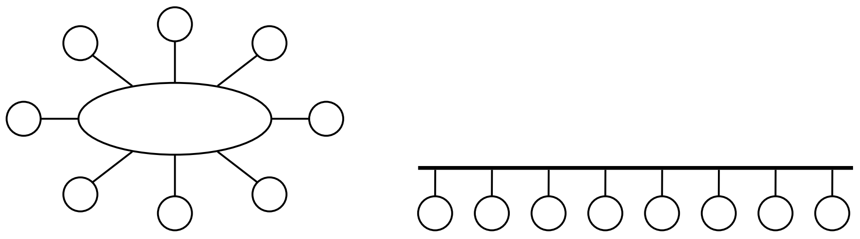


Figure 2.1: Classification of shared-medium network topologies.

[1]Crossbar networks allow any processor in the system to connect to any other processor simultaneously without contention.

[2]Multistage Interconnection Networks (MINs) connect input devices to output devices through a number of switch stages, where each switch is a crossbar network.



(a) Local Area Network (b) Bus-based Network

Figure 2.2: Examples of shared-medium network topologies.

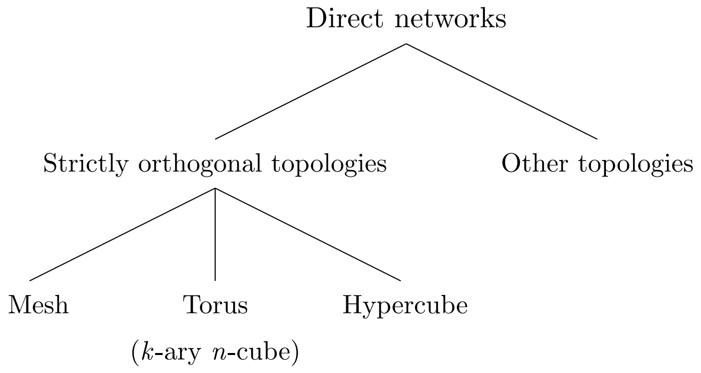
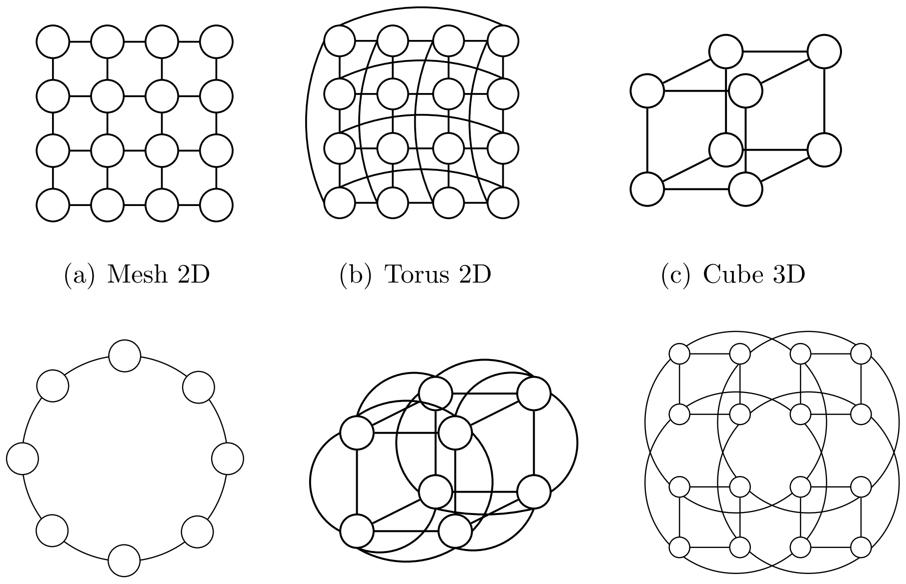


Figure 2.3: Classification of direct network topologies.



(a) Mesh 2D (b) Torus 2D (c) Cube 3D

(d) Ring (e) Torus 3D (f) Cube 4D

Figure 2.4: Examples of direct network topologies (orthogonal).

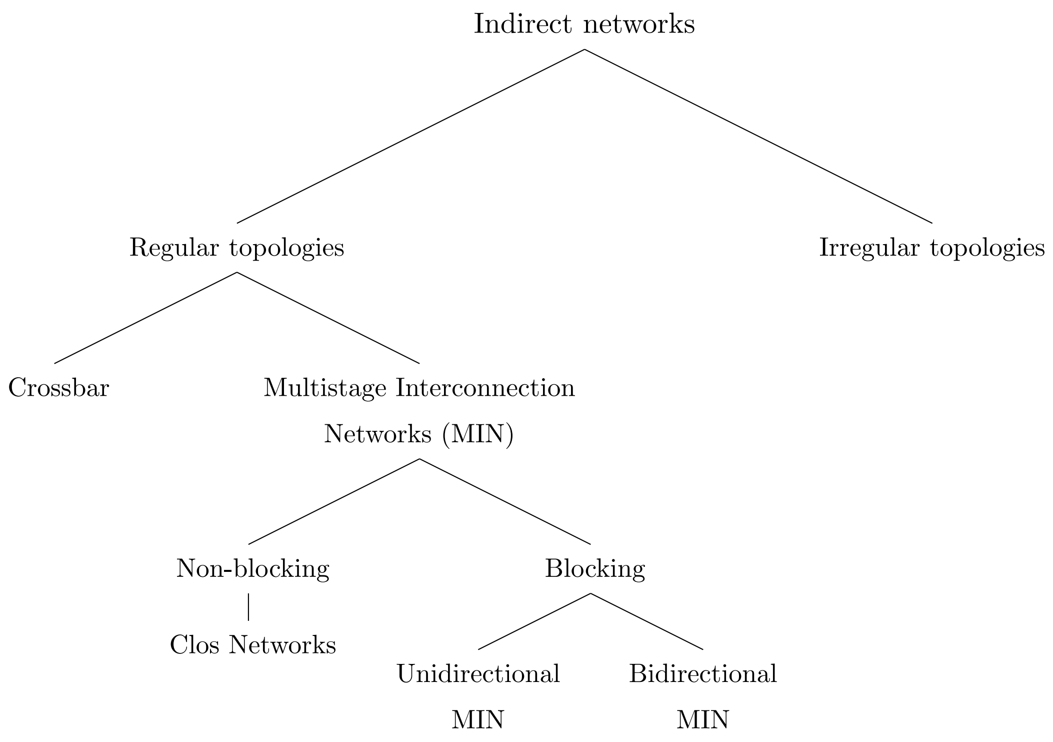
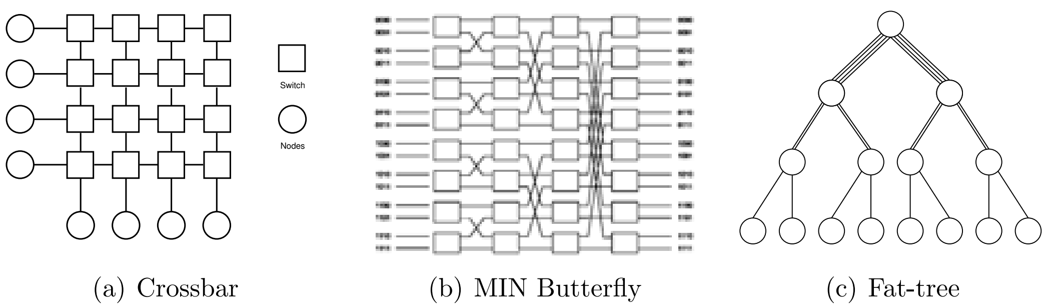


Figure 2.5: Classification of indirect network topologies.



(a) Crossbar (b) MIN Butterfly (c) Fat-tree

Figure 2.6: Examples of indirect network topologies.

2.1.2 Routing

Routing methods determine the path taken by a packet from a source terminal node to a destination terminal node. Indeed, routing algorithms are responsible of assigning one or more paths to each source-destination pair; a path is basically composed by a determined group of switches and links. Although the number of existing options is quite large, routing algorithms can be classified into four main groups using the taxonomy proposed by Duato et al. [22, Ch. 4]. The resulting classification is based on *number of destinations*, *routing decisions*, *implementation* and *adaptivity.* The classification is summarized in Figs. 2.7 and 2.8.

Number of destinations. Routing algorithms where packets have a single destination are known as *unicast* routing algorithms, while those having multiple destinations are called *multicast* routing algorithms.

Routing decisions. This criterion is based on determining *who* and *where* are taken the routing decisions. Decisions can be either taken by a centralized controller (*centralized* routing), or in a *non-centralized* manner. In the latter case, decisions can be taken at the source node prior to packet injection (*source-based* routing) or in a distributed manner while packets traverse the network (*distributed* routing). *Multiphase* routing is an hybrid scheme where the source node selects some destination nodes and the path between them is established based on distributed approaches.

Adaptivity. This is probably the most important classification criterion in the context of this thesis. Adaptivity refers to how routing algorithms select a path between the set of possible paths for each source-destination pair. *Deterministic* routing algorithms always chose the same path between a source-destination pair, even if there are multiple possible paths. *Oblivious* routing algorithms choose a route without considering any information about the network's present state (note that oblivious routing includes deterministic routing). Finally, *adaptive* routing algorithms adapt to the state of the network, using this state information for making decisions.

Implementation. Basically, routing algorithms can be based on *routing tables* storing paths information; or also on *routing functions* (logic or arithmetic) determining the path for each source-destination pair. The routing algorithm can be either deterministic or adaptive in both cases.

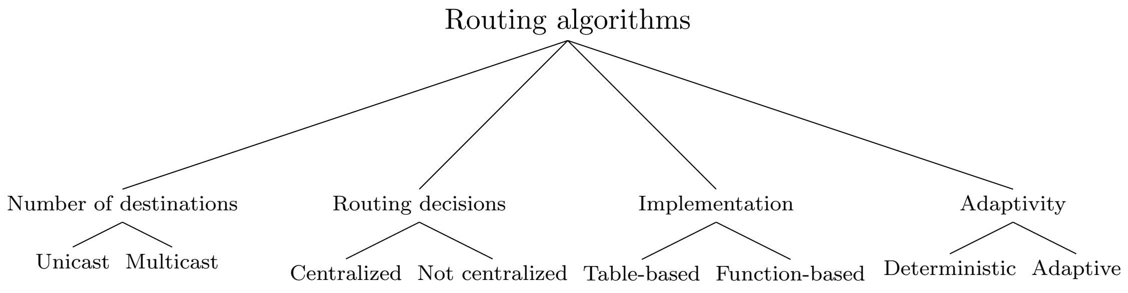


Figure 2.7: Classification of routing algorithms.

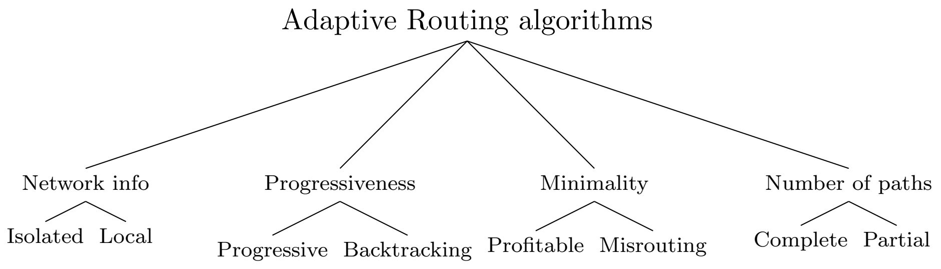


Figure 2.8: Classification of adaptive routing algorithms.

2.1.3 Switching Techniques

Switching techniques are responsible of moving packets within each switch along source-destination paths. Actually, switching is the mechanism that removes data from an input channel and places it on an output channel [64]. At this moment, the most widely accepted techniques are *Store-And-Forward*, *Virtual Cut-Through* [43], and *Wormhole* [13].

Store-And-Forward (SAF). With this technique, each node along the path waits until a packet has been entirely stored (received) before forwarding the packet to the next node. Consequently, each packet is completely buffered at each intermediate node before it is forwarded to the next node. The header information is extracted by the intermediate node and used to determine the output link over which the packet is to be forwarded.

Virtual Cut-Through (VCT). When using this technique, packets can be forwarded as soon as the header (containing routing information) is received and resources (buffers and channels) are acquired, without waiting for the entire packet to be received. However, if the header is blocked on a busy output channel, the complete message is buffered at the node and this technique behaves like SAF.

Wormhole. In *wormhole* switching, a packet is broken up into *flits* (the smallest unit of resource allocation in a router) and input and output buffers are typically large enough to store a few flits. A packet is pipelined through the network at the flit level since each packet is typically too large to be completely buffered within a router. In terms of operation *wormhole* is similar to VCT but with channel and buffers allocated to flits rather than packets, therefore, small buffers can be used. Just a few flits need to be buffered at a router. Consequently, the small buffer requirement and packet pipelining enable the construction of routers that are small, compact and fast [22, Ch. 2].

2.1.4 Flow Control

Flow control is a point-to-point synchronization protocol that determines how network's resources are allocated to packets traversing the network. This protocol is used for transmitting and receiving data employing request/acknowledgment signaling to ensure successful transfer and the availability of buffer space at the receiver [22, Ch. 2], [14, Ch. 13]. The most commonly used schemes are *Credit-based* and *On/Off*.

Credit-based Flow Control. In this scheme, the sender (upstream switch) keeps a count of the number of free *flits*[3] buffers, or *credits*, in the receiver (downstream switch). Then, each time the sender forwards a flit, the credit count is reduced. If the count reaches zero because there are no more free flit buffers, no further flits can be forwarded until a buffer becomes available.

On/Off Flow Control. In this scheme, the sender state is based on a single control bit that represents whether the sender is permitted to send (*on* state) or not (*off* state). This scheme can greatly reduce the amount of sender signaling in certain cases. *Stop-and-go* is a different scheme also based on this concept.

2.2 Fault Tolerance

In the previous section, we have presented some concepts about interconnection networks, let us now introduce some concepts borrowed from the fault tolerance theory.

In simple words, a system is *fault tolerant* if it can mask the presence of faults in the system by using redundancy. As stated by Koren and Krishna [48, Ch. 1], *redundancy* is the property of having more of a resource than is minimally necessary to do the job at hand. As failures happen, redundancy is exploited to mask or otherwise work around these failures, thus maintaining the desired level of functionality.

[3]The flit size can be less than a packet size when wormhole switching is used; it is equal to the packet size when virtual cu-through switching is used.

An important topic in fault tolerance is the difference between the concepts of *fault*, *error* and *failure*. A *fault* is often related to the notion of defect or bug, while *errors* constitute manifestations of faults. It is worth noting that the presence of faults does not ensure the occurrence of errors. A *failure* occurs when the system cannot provide the desired service or when the behavior of the system first deviates from that required by its specifications [50]. The relation between these concepts is shown in Fig. 2.9.

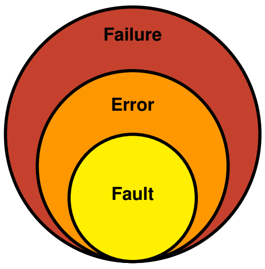


Figure 2.9: Relation between Fault, Error and Failure.

Regarding their duration, faults can be *permanent*, when the component simply goes down; *transient*, when after a malfunctioning time the normal functionality of the component is fully restored; or *intermittent*, when the component oscillates between malfunctioning and normality. Another classification of hardware faults is into *benign* and *malicious*. A fault that just causes a component to go dead is called *benign* or *fail-stop*, while a fault that causes incorrect results that resemble real ones is called *malicious* or *byzantine* [48, Ch. 1]. In the field of interconnection networks, failures are divided into failures of communication channels (*link failures*) and failures of the entire processing element or the its associated router (*node failures*). On a node failure, all physical links incident on the failed node are also marked faulty at adjacent nodes [22, Ch. 6]. Concepts of link and node failures are graphically exemplified in Fig. 2.10.

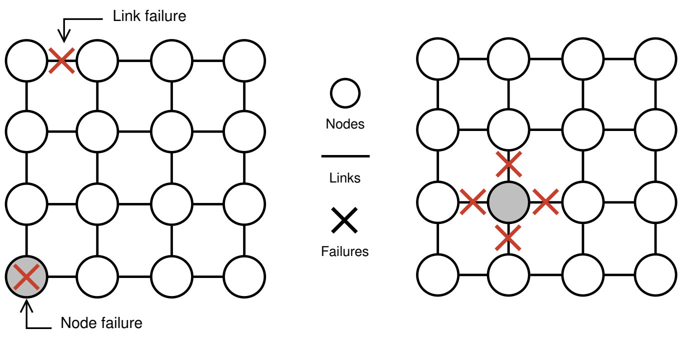


Figure 2.10: Network failures nomenclature.

Four major phases can be identified in providing fault tolerance [41, Ch. 1]:

1. **Error Detection.** It is the starting point of any fault tolerance activity since faults and failures cannot be directly observed but have to be deduced from the presence of errors, commonly by performing checks to see if there is an error or not. After detecting an error in the system state, it is certain that faults are present and failures have occurred somewhere in the system.

2. **Damage Confinement.** This phase is aimed to determine which parts of the system state are corrupted, since there could be a time delay between the failure and the event of error detection, and errors may propagate and spread other parts of the systems.

3. **Error Recovery.** Once the error has been detected and its extent identified, the *error recovery* phase is responsible for achieving the error-free state of the system. Unless the error is removed, the erroneous state may cause the failure of the system in the future.

4. **Fault Treatment and Continued System Service.** If the system error was caused by a permanent fault, the *fault treatment and continued system service* phase is responsible for identifying and isolating the faulty component after the recovery phase.

Having introduced the theory of fault tolerance, we will describe the concepts of network fault tolerance in subsection 2.2.1; fault-tolerant routing in subsection 2.2.2; and deadlock avoidance in subsection 2.2.3.

2.2.1 Network Fault Tolerance

Network fault tolerance borrows concepts from two different fields, interconnection networks and fault tolerance. Within this context, network fault tolerance can be classified according to many aspects. A widely accepted classification scheme is the one that propose the division into *static* and *dynamic* network fault tolerance. Static fault tolerance comprises methods that require a network shutdown to perform a static reconfiguration process after the failure detection, either using new routing algorithms or by adding new hardware. Dynamic fault tolerance groups methods capable of treating dynamic failures *on-the-fly* without requiring network shutdowns, such as dynamic reconfiguration methods and adaptive routing (also called dynamic re-routing). Moreover, there is another valid and commonly used classification scheme that divide network fault tolerance in three main categories depending on the fault tolerance approach being used: *resources redundancy*, *network reconfiguration*, and *adaptive routing algorithms*. The latter classification scheme is the one adopted in this thesis thus we detail its composing categories below.

Resources Redundancy

This approach is used in some systems where spare units are switched in the system to replace the failed units. As a counterpart, this approach usually implies high extra costs in terms of economic resources.

Most of the work published in recent years in the field of fault tolerance for interconnection networks have been focused on *network reconfiguration* and *adaptive routing algorithms*. Consequently, the *resources redundancy* approach has been somehow relegated to a second place, although there are several interesting works within this approach. For instance, Sem-Jacobsen et al. [85] introduce a novel fat-tree variant which is able to tolerate single link and switch faults dynamically by adding additional links between every two switches at the same position in two parallel fat-trees. Another approach for solving this problem is based on adding some hardware to the network to increase the number of available paths for the purpose of fault tolerance, as proposed by both Valerio et al. [99] and Skeie [88].

Network Reconfiguration

Reconfiguration methods, either static or dynamic, are commonly used for adapting routing functions or routing tables to resulting network topologies after the occurrence of one or more failures. This approach is very flexible and powerful but at the expense of significant reductions in the performance of the network. When using *static reconfiguration* methods, the network should be shutdown in order to adapt the routing mechanism to the new topology, and use some sort of rollback-recovery technique to start over from a safe state. On the other hand, *dynamic reconfiguration* methods do not require networks to be shutdown but these methods usually divide the network in order to reconfigure some part of the system, allowing the rest of the network to remain in operation during the reconfiguration process.

A lot of work has been done in the field of *network reconfiguration*. For instance, Gómez et al. [33] present a deterministic routing algorithm for achieving fault tolerance in direct networks based on the use of some predefined intermediate nodes; further extensions to k-ary n-tree topologies have been presented in [31]. Mejia et al. [62] propose a deterministic routing methodology, referred to as *Segment-based Routing*, that partitions a topology into subnets and subnets into segments to gain some degree of freedom (compared to other routing strategies). Morover, some research based on the concepts of multiple paths and *regressive deadlock recovery* has been published by Montañana et al. [63]. On the other hand, Ho and Stockmeyer [36] introduce a fault-tolerant routing methodology that sacrifices a certain number of healthy nodes in order to use no more than two virtual channels, and reduce the routing time. Puente et al. [77] present a fault-tolerant routing technique based on the permanent existence of a safe path for communicating any pair of fault-free nodes. On the basis of [77], Puente et al. [78] have developed a mechanism for tolerating multiple failures applying a novel dynamic reconfiguration process. Their mechanism, known as *Immunet*, is able to manage failures in parallel systems without shutting down or restarting the system, however, the injection of new packets is required to be stopped during the global reconfiguration phase. A specific application of *Immunet* to regular networks, called *Immucube*, has been also proposed by Puente and Gregorio [74] with the aim of solving the scalability problem of *Immunet* (for a large number of nodes).

A different approach, developed for simplifying the reconfiguration process, has been adopted by Lysne and Duato [59]. They identify a restricted part of the network, the *Skyline*, as the only part where a full reconfiguration is necessary thus avoiding the need of reconfiguring the entire network. Similarly, Theiss and Lysne [93] have developed an approach to handle single dynamic faults locally in irregular networks affecting only a limited number of nodes during the reconfiguration process. The same authors have also presented a routing method aimed to handle single dynamic faults combining dynamic reconfiguration with the calculation and use of redundant paths in arbitrary networks [94]. Furthermore, Lysne et al. [60] present a protocol that overlaps various phases of static reconfiguration to provide efficient deadlock-free network reconfigurations in the event of faults (or other circumstances) without requiring multiple sets of data virtual channels. Alternatively, Pinkston et al. [72] propose a simple and general approach for achieving deadlock-free dynamic reconfiguration through the division of the network into two virtual networks (implementing two sets of data virtual channels).

Adaptive Routing Algorithms

Routing algorithms designed for fault tolerance look for alternative paths whether a fault disables the path used to communicate a pair of source-destination nodes. This approach could be outlined as one of the most interesting and suitable options for achieving network fault tolerance because it allows systems to reach better performance results at lower costs. However, designing fault-tolerant routing algorithms poses a challenging problem, specially when dealing with performance degradations and abnormal behaviors caused by fault occurrences. The vast majority of proposed fault-tolerant routing methods assume the existence of diagnostic techniques, and focus on the use of this failure information for providing fault-tolerant solutions. Consequently, diagnoses problems are not addressed by those methods; information about faults need to be known in advance; and the existence of mechanisms capable of distributing this information to corresponding network nodes is commonly assumed. Since fault-tolerant routing methods comprise an important part of the thesis, a more detailed discussion on this topic is presented in subsection 2.2.2.

Several works have achieved good performance results based on the methodology for designing deadlock-free fault-tolerant routing algorithms proposed by Duato [20], [17], [18], [19]. Unfortunately, only a few can be included in the category of *adaptive routing* since most of them are based on *network reconfiguration* methods, as explained above. For instance, Gómez et al. [32] provide an adaptive fault-tolerant routing methodology based on the use of intermediate nodes to circumvent faults. The proposed method assumes that fault detection, information distribution, and checkpoint mechanisms are provided by interconnection networks. It is worth mentioning that the use of intermediate nodes was first proposed by Valiant and Brebner [100] for the purpose of traffic load balancing. Furthermore, one of the only proposals capable of dealing with dynamic failures has been presented by Nordbotten and Skeie [66], [65] as the evolution of [87]. They propose a fault-tolerant routing methodology [66] that tolerates concave fault-regions without stopping network traffic but at expense of allowing specific packet drops and some routing restrictions inherited from the use of a variation of the *turn-model* [30]. Sem-Jacobsen et al. [86] have recently presented a dynamic local rerouting methodology for fat-trees for tolerating up to and including k-1 benign link faults, where k is the number of ports in one direction of a switch in the fat tree. Their proposal cannot be directly applied in current network devices but it can be easily implemented after updating the firmware of switches.

2.2.2 Fault-tolerant Routing

Fault-tolerant routing methods fall into two of the three categories of network fault tolerance: *adaptive routing algorithms* and *network reconfiguration* in the case of deterministic routing algorithms. Nearly all deterministic or static routing algorithms are not directly able to avoid dynamic network failures because they cannot choose and use alternative or adaptive paths *on-the-fly* without halting the system. A well-known example of a deterministic routing algorithm is the *Dimension Order Routing (DOR)* [22, Ch. 4]. By contrast, most adaptive routing algorithms are able to dynamically use alternative paths to avoid faults, thus appearing as a natural solution to the problem of fault tolerance in interconnection networks. However, adaptability poses a new problem *deadlock occurrences.*

Most of the current fault-tolerant routing methods are heavily influenced by the fault tolerance model assumed by each method, particularly by two model attributes: the *failure type* (link, node, etc) and the *failure mode* (static or dynamic) [22, Ch. 6]. The structure, complexity and even the power of fault-tolerant routing methods are closely tied to the *failure mode* adopted. For this reason, it is one of the most important attributes to be considered when designing fault-tolerant routing algorithms. In a few words, if a *static failure mode* is assumed, all failures are permanent and static and they are present in the network when the system is started (or restarted). In contrast, if a *dynamic failure mode* is used, failures may appear at random time and location during system operation, presenting variable time durations.

When assuming a *static failure mode*, the design of routing algorithms could be simplified because the distribution and number of faults is known in advance, therefore, the number of direction changes required to circumvent faulty areas can be reduced. By means of this reduction, deadlock avoidance techniques can be simplified. However, this failure mode still presents a major drawback because to avoid package drops, each time a fault is detected, the system needs to be stopped, the information about faults updated, and the system restarted from a safe state using some kind of rollback-recovery technique. All these actions have an extra cost and can affect performance seriously. In turn, routing algorithms based on the *dynamic failure mode* do not need to know the location of faults in advance, thus avoiding packet drops, and stopping and restarting the system. However, as stated above, *deadlock occurrences* emerges as an extremely important problem because the presence of faults renders existing solutions to deadlock-free routing ineffective [22, Ch. 6]. The problem of deadlock resolution is discussed in detail in subsection 2.2.3.

Another important aspect to be considered in designing fault-tolerant routing algorithms is the likelihood of facing different -and complex- distributions and combinations of network failures. According to their topological shape, fault regions can be divided into *convex* (line-shape and square-shape) and *concave* (L-shape, U-shape, etc). Many studies have addressed this problem through fault-tolerant routing methods for fault rings [6], [35] and other fault regions [7], [71], [8]. Furthermore, Safaei et al. [81], [80] provide some analytical expressions to compute the probability of messages facing fault rings in torus; while Farahabady et al. [23], [24] propose a solution to calculate the probability of occurrences of common fault patterns in torus and mesh interconnection networks.

Unfortunately, the randomness of network failures increase the complexity of dynamic fault tolerance mechanisms since routing algorithms must be able to circumvent faults *on-the-fly*, without knowing the temporal and spatial distribution of faults. Each time the routing algorithm adds a direction or dimension change to avoid failures, routing functions are redefined and new routing cycles can appear. As those new cycles may be of any shape, deadlocks are prone to occur. Under these circumstances, the number of resources needed to avoid deadlocks is directly proportional to the number of faults to be tolerated. This is the source of complexity and the biggest scalability constraint of most current deadlock avoidance techniques.

2.2.3 Deadlock Resolution

In the context of interconnection networks, a *deadlock* occurs when some packets cannot advance toward its destinations because the buffers requested by them are full. In this situation, every packet is requesting resources held by other packets while holding resources requested by other packets; therefore, a set of packets is blocked forever [22, Ch.3]. This situation has been previously defined by Coffman et al. [10] by means of the *hold-and-wait* and *circular wait* conditions; and can be represented by a *resources dependence graph* where the vertices of the graph represent the set of resources and the edges between vertices denote dependencies [14, Ch. 14]. If no cycles are present, there can be no deadlock as cycles are a necessary (but not sufficient) condition for deadlock to occur [111, Ch. 13].

Efficiently handling deadlock anomalies is one of the more critical problems to be addressed in order to achieve network reliability, robustness and high performance. Along the last decades, several techniques have been proposed for handling deadlock situations, including injection limitation, [79], [76]; virtual escape channels [17], [19], [21]; chaotic routing [47]; and progressive recovery [3]. Unfortunately, almost all these techniques have been designed for deadlock resolution in fault-free networks and most of them are not even applicable in the event of network failures [22, Ch. 6]. For instance, the vast majority of current high-performance computing systems –including *BlueGene/L* and *BlueGene/P* [38]– rely on deadlock avoidance methods based on the use of virtual channels for breaking cycle dependencies. However, the biggest limitation of such methods is their scalability since the quantity and type of resources needed to avoid faults is often proportional to the number of faults to be tolerated, thus increasing cost and complexity of interconnection networks. Consequently, most of current deadlock handling techniques turn into non-viable options when dealing with a medium or large number of failures.

In handling deadlock situations, three approaches can be adopted[4]: *prevention*, *avoidance*, and *recovery*. The complete classification of techniques for deadlock handling is shown in Fig. 2.11. The approach of *deadlock prevention* is based on reserving all needed resources before starting the packet transmission; this strategy is very conservative and may

[4]For the remainder of this section we will use a *pseudo-standardized* terminology based on concepts and works previously given by Pinkston [111, Ch. 13] and Dally and Towles [14, Ch. 14].

lead to a low utilization of resources. The *deadlock avoidance* approach is less conservative since resources are requested as packets advance through the network but at the same time, a resource is granted to a packet only if the resulting global state is free of deadlock. Deadlock can be avoided by eliminating cycles in the *resources dependence graph*, imposing a partial order on the resources allocation process. However, achieving this is not an easy task, specially in distributed systems. Techniques for accomplishing deadlock avoidance can be classified as being *queue-based* (including *channel-based* and *buffer-based* schemes), *path-based*, or some *hybrid* combination.

On the other hand, *deadlock recovery* strategies usually provide some sort of detection mechanism because in these strategies resources are granted to packets without any check and deadlocks may occur. If a deadlock is detected, some resources are deallocated (usually by dropping packets) and granted to other packets. These strategies are based on the assumption that deadlocks are rare and the recovery process can be tolerated by the system. Deadlock recovery can be resolved by removing one or more packets from the network (*regressive recovery*); or by ensuring that at least one packet no longer waits for resources occupied by other packets (*deflective recovery* and *progressive recovery*) [101].

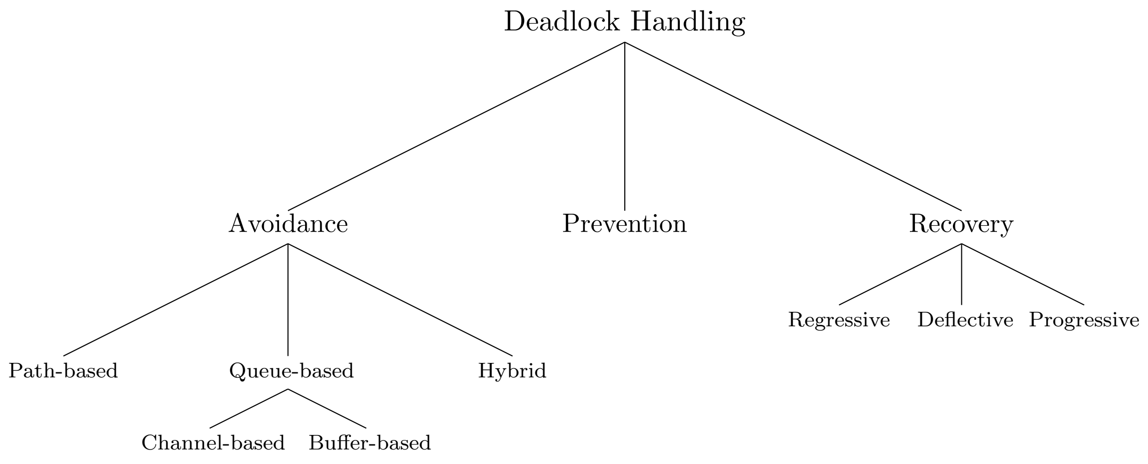


Figure 2.11: Classification of techniques for handling deadlock situations.

Let us now describe some proposed techniques for *deadlock avoidance* and *deadlock recovery.* The approach of *deadlock prevention* will not be discussed further in this thesis since it is typically used only in legacy circuit switched networks [111, Ch. 13].

Deadlock Avoidance

As explained above, cyclic situations can be avoided through different approaches. *Path-based* schemes have the advantage of not requiring virtual channels to strictly avoid deadlocks but at the expense of reduction in flexibility and applicability [111, Ch. 13].

Routing algorithms such as *Dimension-Order Routing* [84], *Turn Model Routing* [30], and *Up*/Down* Routing* [83] are some of the most popular and widely adopted *path-based* avoidance schemes.

Queue-based schemes usually decouple the use of physical links from the allocation of virtual channels or buffers (queue resources). Within this group, *buffer-based* schemes were those traditionally used in legacy packet switched networks, associating packets to some sort of buffer classes. In contrast, *channel-based* schemes usually rely on imposing ordering conditions to the use of virtual channels, as in [12] and [51].

The *hybrid* approach is a combination of both schemes *path-based* and *queue-based* (mostly of *channel-based*). It is usually applied to avoid deadlocks when performing network reconfigurations. Some well-known examples of hybrid avoidance schemes are: the *Chaotic Routing* [45], [46], [47]; *Planar Adaptive Routing* [9]; *Deflection Routing* [34]; the *MIT Reliable Router* [15]; and the *Adaptive Bubble Router* [75], [76].

The *BlueGene/L* and *BlueGene/P* supercomputers use an hybrid path/channel-based avoidance scheme, similar to the solution proposed by Puente et al. [75], [76]. These supercomputers have two dynamic virtual channels (VC); one is a *bubble escape* VC that can be used both for deadlock handling and deterministic routing; and the other is a high-priority bubble VC with dimension-order routing [2].

Deadlock Recovery

Deadlock recovery approaches aim to improve the utilization of routing resources (improving performance) in the absence of deadlock. By killing at least one packet, *regressive* recovery schemes create bubbles in potential deadlock cycles, as in *Compressionless Routing* [44].

In *deflective* recovery schemes, bubbles are not guaranteed to be always available since they are allocated randomly when multiple packets compete to consume them at each router [111, Ch. 13]. Some examples of deflective schemes are the *Software-based recovery* [61], and *Hole-based Routing* [11].

Progressive deadlock recovery is based on the concept of resolve deadlock by providing access to a deadlock-free path for progressively routing some packets out of the dependencies cycle. Some examples of progressive recovery techniques are the works based on the *Disha* scheme, including *Disha Sequential* [3] and *Disha Concurrent* [4]. A more recent progressive recovery technique have been proposed by Song and Pinkston [89], [90]. This technique, called *Ping and Bubble*, is based on the idea of tracing all cyclic dependencies in real time and to dynamically supply a bubble to those resources and force it to traverse the entire cycle.

Chapter 3

Fault-tolerant Distributed Routing Balancing

In this chapter, we present an adaptive fault-tolerant routing method for treating a large number of dynamic network link failures. The proposed method, called *Fault-tolerant Distributed Routing Balancing* (FT-DRB), exploits the communication path redundancy available in many network topologies by means of a distributed multipath routing approach. Apart from treating link failures, the method is also able to deal with congestion problems and performance degradations caused by the occurrence of failures. This method shares theoretical basis with two previous theses; it is based on the method developed by Franco et al. [25] [27], [28], *Distributed Routing Balancing* (DRB); subsequently improved by Lugones et al. [55], [56], [57].

Conceptually, the proposed fault-tolerant routing method is based on the state information of source-destination paths. This information includes latency values of the path and the links state information of the communication path. If there are no link failures along the path, each application message records latency information about the path it traverses. Once the message reaches the destination, this terminal node sends the latency information of the path back to the source terminal node, using an ACK message. If there is at least one link failure along the source-destination path, it is discovered when a packet tries to use the faulty link. In the fault tolerance theory, this first action would correspond to the *error detection phase.* After this phase has been completed, *damage confinement* and *error recovery* must be provided. To this end, the network node which discovers the failure sends back a special ACK packet, in order to alert the source node about the failure in the path. The latter action corresponds to the phase of *damage confinement.* Almost simultaneously, those messages that have been already sent through the path where the link failure has been discovered are rerouted towards the destination node. This corresponds to the *error* *recovery* phase of the fault tolerance theory. As this rerouting action is intended to be a fast and temporary response to link failures, it may not be the optimal solution. For this reason, the proposed method includes a third and last action, which represents the phase of *fault treatment and service continuity.* At this point, the source node disables the faulty path and reconfigures new paths for the following messages, in order to avoid faults, ease routing paths, and improve performance. Once those new paths have been configured, their latency values are recorded and then sent back from the destination to the source node. Counting on this information, the source node is able to calculate the number of alternative paths that must be used and can distribute messages among them, according to the network traffic burden. Using one or more alternative paths, the method is able to avoid and/or circumvent link failures, while improving the system performance by means of distributing and balancing communications among the alternative paths. The routing method uses a simple and scalable technique for avoiding deadlock occurrences caused by the treatment of failures (explained in detail in chapter 4).

The rest of the chapter is organized as follows. The procedures for error detection and damage confinement are explained in section 3.1. Then, the selection of alternative escape paths (error recovery) is defined in section 3.2. The configuration of multipaths for fault treatment and continued system service is explained in section 3.3. Procedures to properly treat both permanent and transient failures are detailed in section 3.4. The architecture of network components is exposed in section 3.5. Additionally, some design alternatives are disclosed in section 3.6. Finally, the method is summarized and discussed in section 3.7. Before describing FT-DRB, it is necessary to introduce the following assumptions.

Initial Assumptions

In the context of this thesis, we use the term *router* to reference network nodes (devices) capable of receiving packets on inputs, determining their destination based on the routing algorithm, and then forwarding packets to the appropriate output [14, Ch. 2]. Consequently, the term *node* will be used when referencing terminal and processing nodes.

The proposed fault-tolerant routing method has been designed to treat *fail-stop benign* link failures; it is not intended to treat byzantine link failures since we assume the existence of error detection and correction techniques in lower layers (to treat bit changes and errors). Likewise, we do not take into account: situations where network nodes are completely disconnected due to multiple link failures; data losses caused by network node failures; and dead-end *labyrinth-like* fault regions. It is worth nothing that these are common assumptions in the context of dynamic network fault tolerance. Remaining assumptions and some specific implementation issues are detailed in section 3.5.

3.1 Monitoring, Detection and Notification

Under normal circumstances, in the absence of failures, the proposed fault-tolerant routing method divides each application message into network packets to route them from the source to the destination, through the path defined by the routing function. While traversing routing paths, each packet monitors two parameters: the state of the corresponding output link *for error detection*; and the path latency *for improving performance*. We will come back to the latter monitoring action in detail in section 3.3.

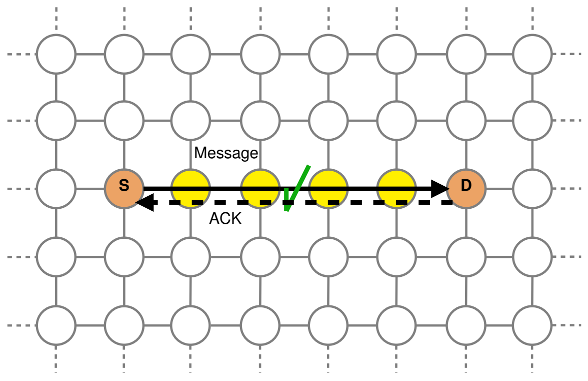
Packets monitor the state of the output link at each router, taking advantage of link level information available in network devices[1]. For instance, the *InfiniBand architecture specification* [39] defines four link states, namely: *LinkDown*, *LinkInitialize*, *LinkArm* and *LinkActive*. Although implementing FT-DRB on InfiniBand is beyond the scope of this thesis, the viability and benefits of implementing *DRB-based* methods have been previously studied by Lugones et al. [57], [52].

Upon detecting a link failure, a special ACK packet carrying information about the failure (*router ID + port ID*) is sent back to the source node. The purpose of this notification is alert source nodes about the existence of failures along communication paths allowing them to disable affected paths, thus confining and reducing the damage caused by the occurrence of link failures. In FT-DRB, notification ACK packets carrying information about failures are considerably smaller than the rest of data packets and they have a higher priority in the routing unit. A simple pseudocode outlining the link status monitoring process is shown in Algorithm 3.1. The actions involved in normal forwarding of messages (including the path latency notification) are graphically shown in Fig. 3.1(a). Similarly, the link failures detection and notification processes are shown in Fig. 3.1(b).

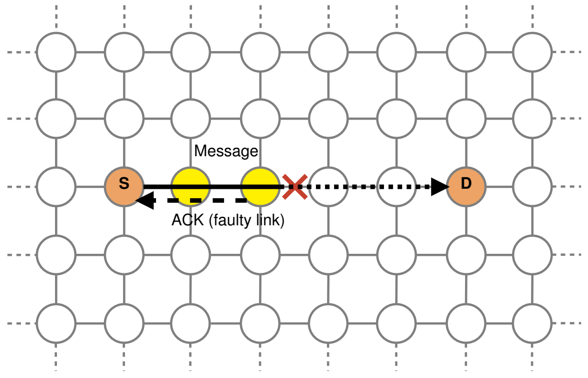
```
MonitorLinkStatus(Packet P)
/* At each FT-DRB Router */
  for each step of P do
    Check the link status
    if (link_stauts == active) then
      Continue to next router or to final destination
    else
      Send failure notification (ACK packet) back to source
      Configure an escape path [explained in section 3.2]
  end for
end MonitorLinkStatus
```

Algorithm 3.1: Link status monitoring.

[1]Modern routers check the state of links by measuring electric impedance and power.



(a) Normal forwarding of messages

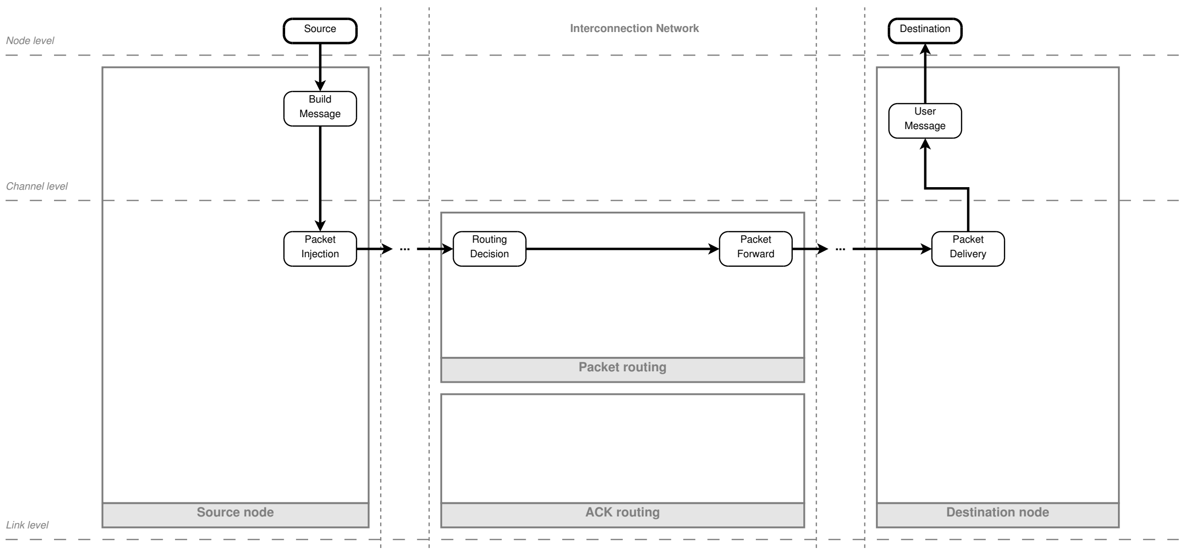


(b) Link failure notification

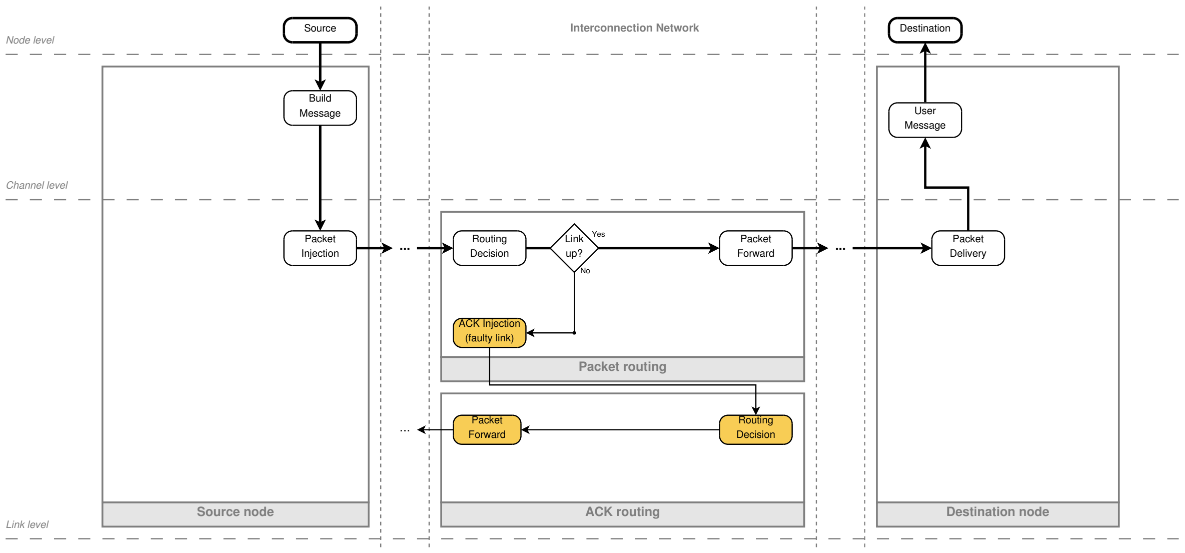
Figure 3.1: Examples of monitoring, detection and notification processes.

The entire set of actions explained above is detailed in the two functionality diagrams shown in Fig. 3.2, where the diagram of Fig. 3.2(a) details the normal forwarding of messages, and the diagram of Fig. 3.2(b) adds the functionalities for detecting link failures. Both diagrams consist of four main blocks: *Source node*; *Packet routing*; *ACK routing*; and *Destination node.* The source and destination node blocks contain the actions implemented at the source and destination nodes, respectively; while the packet routing block together with the ACK routing block represent actions carried out by routers along source-destination paths. Each block is composed by several elements (stage boxes and decision elements) representing the actions performed by routers and source and destination nodes. Colorless elements represent the set of actions performed in the absence of failures, and the colored ones correspond to the additional features included for the purpose of fault tolerance.

In providing network fault tolerance, monitoring and failure detection processes constitute the phase of *error detection.* Similarly, the injection and forwarding of link failure notifications correspond to the phase of *damage confinement.*



(a) Normal forwarding of messages



(b) Link failure notification

Figure 3.2: Monitoring, detection and notification.

3.2 Selection of Escape Paths

This process is started immediately after the detection of link failures. This step is intended to provide solutions for rerouting packets still in transit through faulty paths, as those packets that have been sent before the source node has received the notification.

The error recovery solution consists on configuring and selecting escape paths to circumvent dynamic link failures *on-the-fly*. In the process of selecting alternative paths, the router that has detected the link failure first checks the status of the remaining output links. Counting on this information, the router selects a set of intermediate network nodes to configure a non-faulty alternative path to the destination. Alternative paths are basically *multistep paths* composed by three or more path segments grouped in three main stages. The first stage corresponds to the segment between the source and the first intermediate node; the second stage comprise the set of segments between the first and the last intermediate nodes[2]; finally, the third stage corresponds to the segment ranging from the last intermediate node to the final destination. An example of an alternative escape path based on the use of two intermediate network nodes is shown in Fig. 3.3.

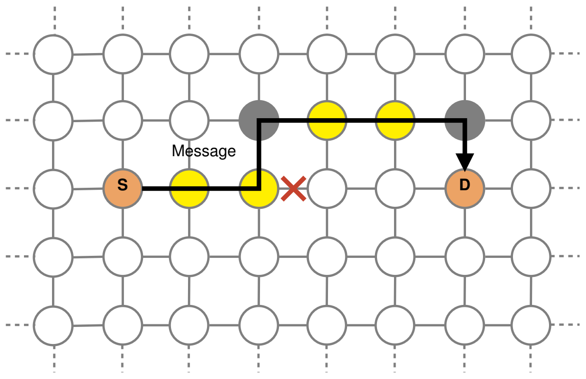


Figure 3.3: Example of an alternative escape path.

The set of intermediate network nodes that can be used in alternative paths is called *supernode* [25]. Intermediate nodes are chosen according to their distance to the router that have detected the link failure in order to maximize the number of possible paths that can be used. For instance, nodes of 1-hop distance are considered first, then nodes of 2-hop distance and so on. An example of the distance-based selection of intermediates nodes is shown in Fig. 3.4. The number of intermediate nodes that can be used in the method is not limited, so that the path could be segmented several times in order to avoid link failures. Segmented paths are called *multistep paths* and each segment relies on the use of

[2]The number of segments in this stage is equal to $i - 1$, where i is the number of non-overlaying intermediate nodes of the alternative path.

the minimal static routing provided by the network topology[3]. For example, FT-DRB uses *Dimension-Order Routing* in tori and meshes. It is important to note that our routing method relies on the use of a novel deadlock avoidance technique presented in [105] and [108] and explained in detail in chapter 4.

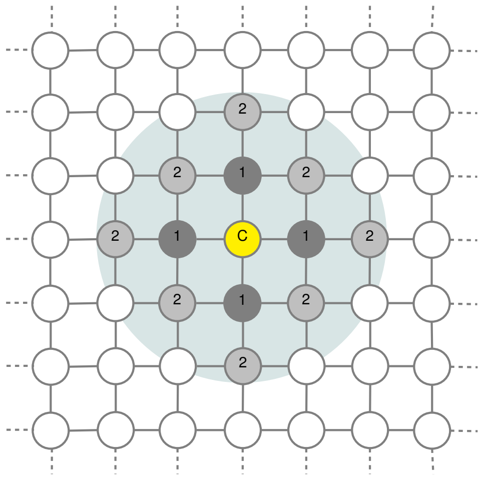


Figure 3.4: Example of a supernode with 1-hop and 2-hop intermediate nodes.

In our method, multistep paths have two different purposes: providing escape paths to avoid link failures *on-the-fly* (at any router along the path); and configuring source-based alternative paths. We will explain the latter option in detail in section 3.3. The mathematical definition of a multistep path (MSP) is given in Eq. 3.1, where S corresponds to the router that has detected the link failure and configures the alternative path[4]; D represents the final destination node; and intermediate routers are denoted as In_i. Furthermore, each P_j is a simple segment between any two nodes (terminal or intermediate); and the symbol $\bullet$ represents concatenation.

$$\begin{aligned} MSP &= \prod(S, In_1, In_2, \ldots, In_{i-1}, In_i, D) \\ &= P_1(S, In_1) \bullet P_2(In_1, In_2) \bullet \ \ldots \ \bullet P_{j-1}(In_{i-1}, In_i) \bullet P_j(In_i, D) \end{aligned} \tag{3.1}$$

It is possible to infer from the definition of Eq. 3.1 that a MSP may not be minimal in some situations, although each of their composing segments are based on minimal routing. The length of a MSP is defined in Eq. 3.2 as the sum of the length of each individual

[3]Routing through intermediate nodes and multiple headers are further explained in section 3.5.
[4]In source-based alternative paths (explained in section 3.3), S is used to denote source nodes.

segment, $Length(P)$. In case of minimal static routing, $Length(P)$ equals the minimum number of links that must be crossed to go from the source node to the destination node.

$$\begin{aligned} Length(MSP) &= Length(P_1(S, In_1)) + Length(P_2(In_1, In_2)) + \ldots + \\ &\quad Length(P_{j-1}(In_{i-1}, In_i)) + Length(P_j(In_i, D)) \end{aligned} \tag{3.2}$$

A new and updated message forwarding diagram, including the *Escape Path Selection* element (*Packet routing* block), is shown in Fig. 3.5. Notice that the functionality of error detection and escape paths selection has been also added to the *ACK routing* block.

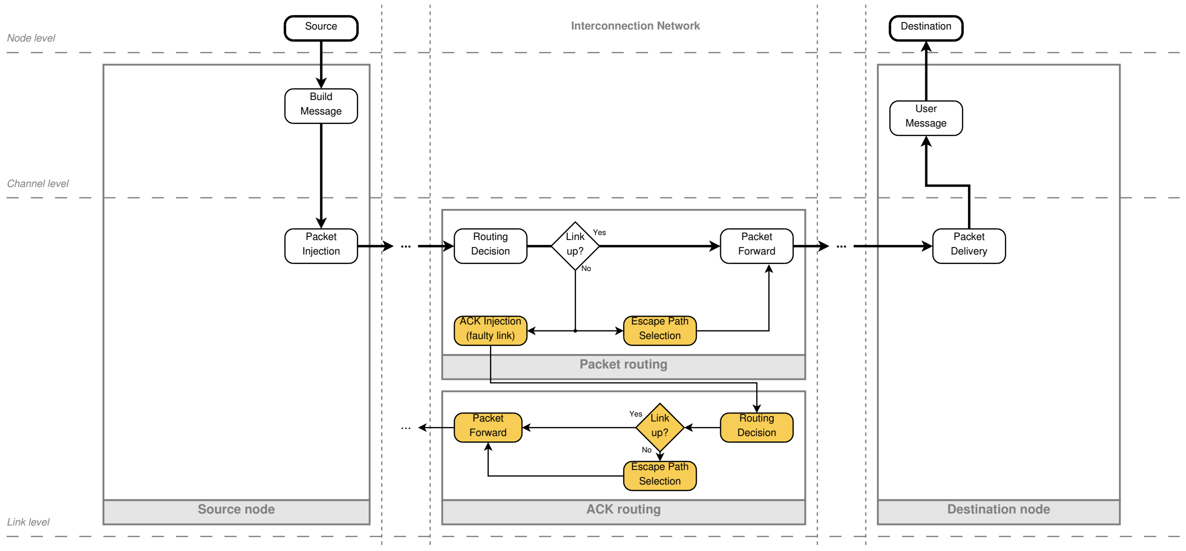


Figure 3.5: Selection of escape paths.

3.3 Configuration of Alternative Paths

The objective of this phase is to configure appropriate source-based routing paths between sources and destinations affected by network link failures, taking into account the network condition after the event of failures. This is particularly important since escape paths were designed to provide fast responses to link failures, but not to be long-term solutions.

The first step for fulfilling this objective is the selection of appropriate communication paths for source-destination pairs affected by one or more link failures. Upon receiving a link failure notification, the source node configures one or more alternative routing paths to the corresponding destination. The process for configuring these new source-based alternative paths is similar to the one explained in the previous section, but starting from the source node instead of the router that detects the link failure. In source-based alternative paths, intermediate nodes are chosen taking into account the link failure information received in the notification ACK packet. Consequently, intermediate nodes serve as scattering and gathering areas for the new alternative fault-free paths.

Once the alternative paths have been configured, sources nodes are able to monitor the traffic load and latency of each path, as briefly mentioned in section 3.1. Latency monitoring is carried out by packets as they traverse the new configured alternative paths. Each packet records and carries the information about the latency it experiences while blocked at routers along the path. The latency of the multistep path is determined according to the expression given in Eq. 3.3. When the packet finally reaches the destination node, the accumulated latency value is obtained from the packet. If the packet has arrived through a fault-free path, the source node is notified about the network traffic burden by means of an ACK packet carrying the overall path latency value[5]. This situation corresponds to the example of Fig. 3.1(a). The complete process is outlined in the pseudocode presented in Algorithm 3.2. As in the case of link failures notification, ACK packets carrying latency values have higher priority in the routing unit and count on the same fault tolerance mechanism than the rest of packets.

$$Latency(MSP) = Transmission\ time + \sum QueuingDelay(router) \tag{3.3}$$

```
MonitorPathLatency(Packet P)

/* At each FT-DBR Router */
  for each step of P do
    Accumulate latency (queue time)
    Continue to next router or to final destination
  end for

/* At the destination node */
  if (path_status == fault_free) then
    Send the path latency ACK to the source

end MonitorPathLatency
```

Algorithm 3.2: Path latency monitoring.

[5]Latency values of faulty paths are not sent back to source nodes because those values also include the latency of the alternative escape path.

The set of alternative paths between each source-destination pair is called *multipath* or *metapath* [25]. A mathematical definition of the *metapath* $P*$ is given in Eq. 3.4. In addition, an example of a *metapath* formed by five different MSPs is shown in Fig. 3.6.

$$P* = \bigcup_{\forall j} MSP_j(S, In_1, In_2, \ldots, In_{i-1}, In_i, D) \tag{3.4}$$

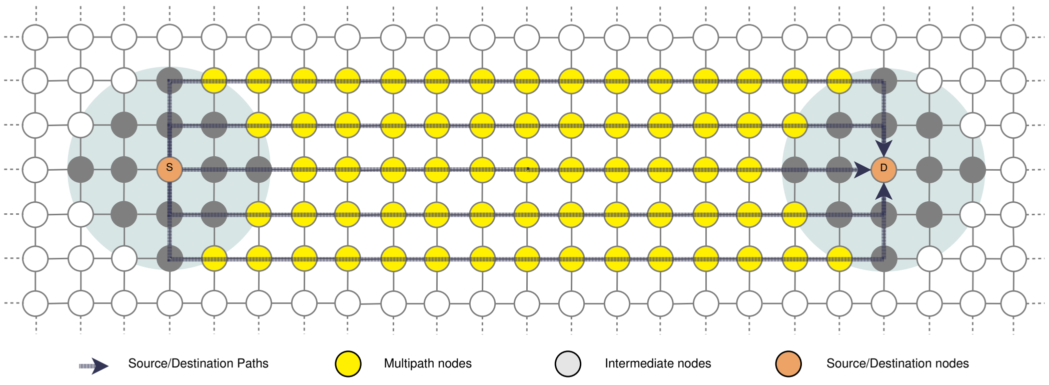


Figure 3.6: Example of a metapath composed by 5 multistep paths.

Upon receiving the latency information of a MSP, the source node calculates the new latency of the metapath using the function defined in Eq. 3.5. Then, the source node decides to increase or reduce the number of MSPs, depending on whether the latency values is off an interval defined by $[Thl - Tol, Thl + Tol]$, where Thl is a predefined latency threshold value and Tol defines the tolerated deviation.

$$Latency(P*) = (\sum_{\forall j} Latency(MSP_j)^{-1})^{-1} \tag{3.5}$$

According to the latency value $Latency(P*)$, the source node would:

- Increase the number of MSPs if $(Latency(P*) > Thl + Tol)$.
- Maintain the same number of MSPs if $((Thl+Tol) > Latency(P*) > (Thl-Tol))$.
- Decrease the number of MSPs if $(Latency(P*) < Thl - Tol)$.

Counting on the links status and path latency information, source nodes are able to calculate the number of alternative paths that must be used and to distribute packets among them, according to the network traffic burden and the spatial distribution of link failures. These actions correspond to the *Metapath Configuration* and *Multistep Path Selection* processes, respectively. A graphical example of two MSP is shown in Fig. 3.7.

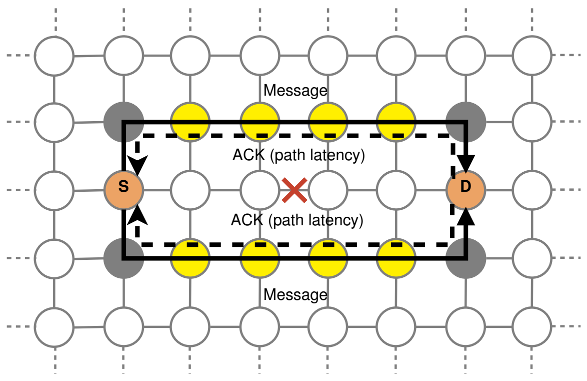


Figure 3.7: Example of a two MSPs metapath configuration.

The *metapath configuration* process is aimed to determine the appropriate number of alternative paths (the size of the metapath) needed for each source-destination pair, taking into account the information obtained from both *link status* and *path latency* monitoring. Therefore, the metapath configuration process is applied on the basis of topological and functional conditions of the network. The purpose of this process is to ensure the operation of alternative fault-free paths within the latency interval defined by $[Thl - Tol, Thl + Tol]$. If the latency increases, the size of the metapath must be also increased in order to obtain a higher bandwidth to meet the new traffic demand. Notice that network traffic conditions may vary (for worse) after the occurrence of link failures owing to changes in the network topology. However, if the latency decreases below $Thl - Tol$, the size of the metapath is too large and it is using resources that should be released to benefit the communication of other source-destination pairs. The metapath configuration process is outlined in the pseudocode of Alg. 3.3.

```
MetapathConfig(MultiStepPath MSP, Threshold Thl, Tolerance Tol)
/*At the source node (upon receiving a new ACK notification) */

  if (ACK == path_latency) then
    Calculate the Latency(P*) according to Eq. 3.5
    if (Latency(P*) > Thl+Tol) then
      Increase the number of alternative MSPs
    else if (Latency(P*) < Thl−Tol) then
      Decrease the number of alternative MSPs
  else if (ACK == failure_notification) then
    Configure alternative source−based MSPs

end MetapathConfig
```

Algorithm 3.3: Metapath configuration.

The *multistep path selection* process is invoked before any new application message is injected into the network. This process has two major goals: avoid the use of faulty paths; and distribute the communication load among the MSPs composing the metapath. Consequently, application messages are proportionally distributed among the fault-free alternative MSPs, based on the *probability density function* of MSPs bandwidths; the most available bandwidth the most frequent use. For instance, suppose a metapath composed by s alternative paths for a given source-destination pair, and its associated bandwidth, $Bandwidth(P*)$, calculated according to Eqs. 3.6 and 3.7. The selection process consist on choosing a specific MSP_j, according to the discrete value of the probability given by the *probability density function* of Eq. 3.8, where s is the size of the metapath. The process of building the *probability density function* is explained in detail in [25] and [26]. For clarity, the multistep path selection process is briefly outlined in the pseudocode of Alg. 3.4.

$$Bandwidth(MSP) = (Latency(MSP))^{-1} \tag{3.6}$$

$$Bandwidth(P*) = Latency(P*)^{-1} = \sum_{\forall j} Bandwidth(MSP_j) \tag{3.7}$$

$$\rho(MSP_j) = \frac{\sum_{i=1}^{j} Bandwidth(MSP_j)}{\sum_{i=1}^{s} Bandwidth(MSP_j)} \tag{3.8}$$

```
MultiStepPathSelection()
/* At the source node (before injecting an app. message) */
  Build the probability density function (PDF) of MSPs bandwidths
  Select the MSP using the PDF
  Inject the application message into the network
end selection
```

Algorithm 3.4: Multistep path selection.

The diagram of Fig. 3.8 summarizes the entire set of actions of the proposed *Fault-tolerant Distributed Routing Balancing* method. The path latency monitoring process is represented by three new elements: the *Latency Accumulation* in the *Packet routing* block; and the *Latency information* and *ACK Injection (path latency)* in the *Destination node* block. Similarly, the *Metapath Configuration* and *Multistep Path Selection* processes are represented by their homonyms elements in the *Source node* block.

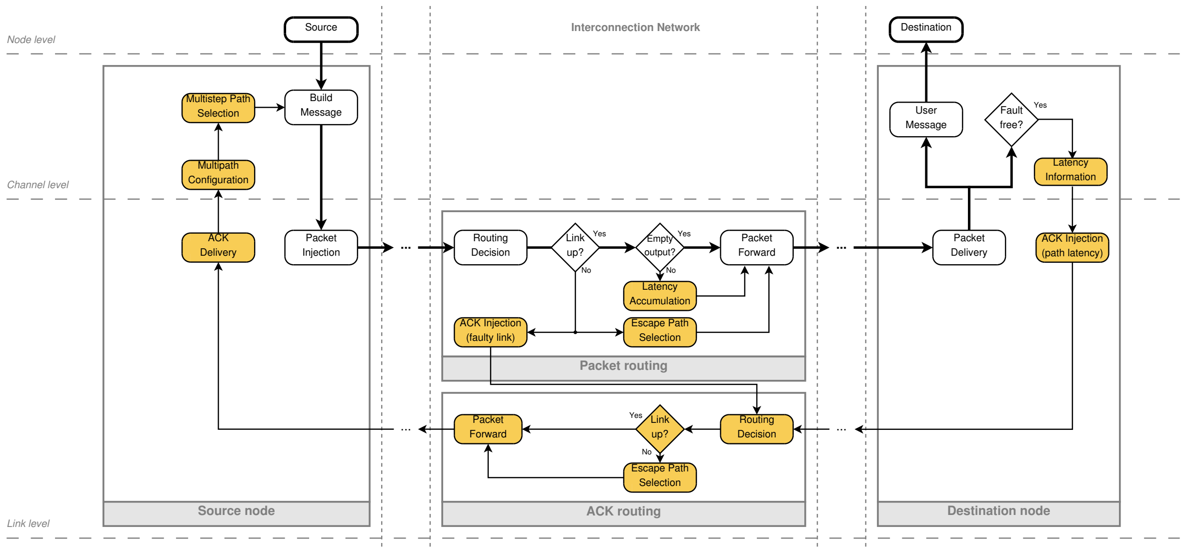


Figure 3.8: Metapath configuration.

3.4 Permanent and Transient Faults

Using the procedures explained in the previous sections, FT-DRB is able to treat a large number of dynamic link failures. These procedures are suitable for the treatment of *permanent* failures because they provide alternative paths based on network conditions (avoiding the use of faulty paths). However, these alternative paths are not entirely optimal –in terms of resource utilization– when dealing with *transient* link failures. For example, it may happen that a pair of source-destination nodes need to configure and use non-minimal or congested paths for providing fault tolerance. This situation is acceptable provided that a link failure prevents the use of best suited paths. However, the overall system performance can be improved whether the routing method is able to switch back to minimal or low-latency paths when available.

In order to maximize the use of network resources, FT-DRB detects and applies differential treatments to *permanent* and *transient* link failures. At a first stage, link failures are always considered and treated as *transient* failures. If a failure persists over time, its state is changed from *transient* to *permanent*. *Intermittent* failures would be treated by FT-DRB as *permanent* or as *chronic transient* failures, depending on the error intervals of the failure. The detection and correct treatment of permanent and transient link failures is based on properly handling ACK packets. Each notification packet carries either a link failure notification or a path latency value. Link failure notifications alert the source node about the existence of at least one link failure along the communication path and the need for configuring a new fault-free source-destination path. By contrast, as explained in the previous section, the reception of a path latency value implies the absence of failures along the path since destination nodes only send back latency values of fault-free paths.

As explained in section 3.1, when a link failure is detected by a router along the source-destination path, that network node sends back the link failure information to the corresponding source node by means of an ACK packet. At first, source nodes consider link failures as transient but if the source node receives multiple failure notifications regarding the same link, that node will treat such failure as permanent. In other words, a failure on a link will be marked as permanent only after receiving a predefined number of failure notifications regarding that specific link. Notice that the number of failure notifications is a parameter of the method, a modifiable threshold set by default to 5 notifications.

Upon receiving an ACK packet, each source node processes the information contained in the notification. If the packet carries information about a link failure, the information is stored or updated in the *Link Faults Information List* of the source node, shown in the diagram of Fig. 3.9. If the number of failure notifications of that specific link has reached the threshold value (5 notifications), its state is changed from *transient* to *permanent*. On the other hand, the reception of an ACK packet carrying path latency information indicates that failures have disappeared since at least one message has reached the destination node through that path, therefore, the state of every link along the path is changed to fault-free (their entries are removed from the *Link Faults Information List*). For clarity, the entire set of actions related to the reception of both kinds of ACK packets has been included in the flow diagram shown in Fig. 3.10.

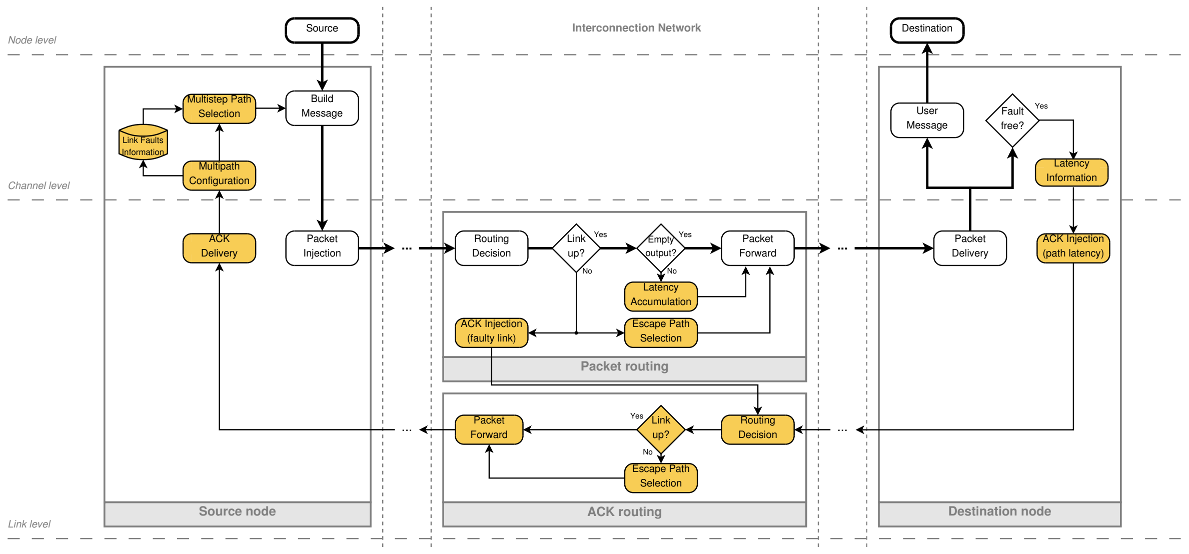


Figure 3.9: Permanent and transient faults.

Then, counting on the information stored on the *Link Faults Information List*, source nodes are able to get the overall state of their source-destination paths. Therefore, source nodes tries to send messages through a communication path until one or more links of that path shows a permanent failure. If there are permanent link failures along the path, each source node sets one or more alternative paths and sends the rest of the messages through them. All these actions are detailed in the message injection flow diagram shown in Fig. 3.11.

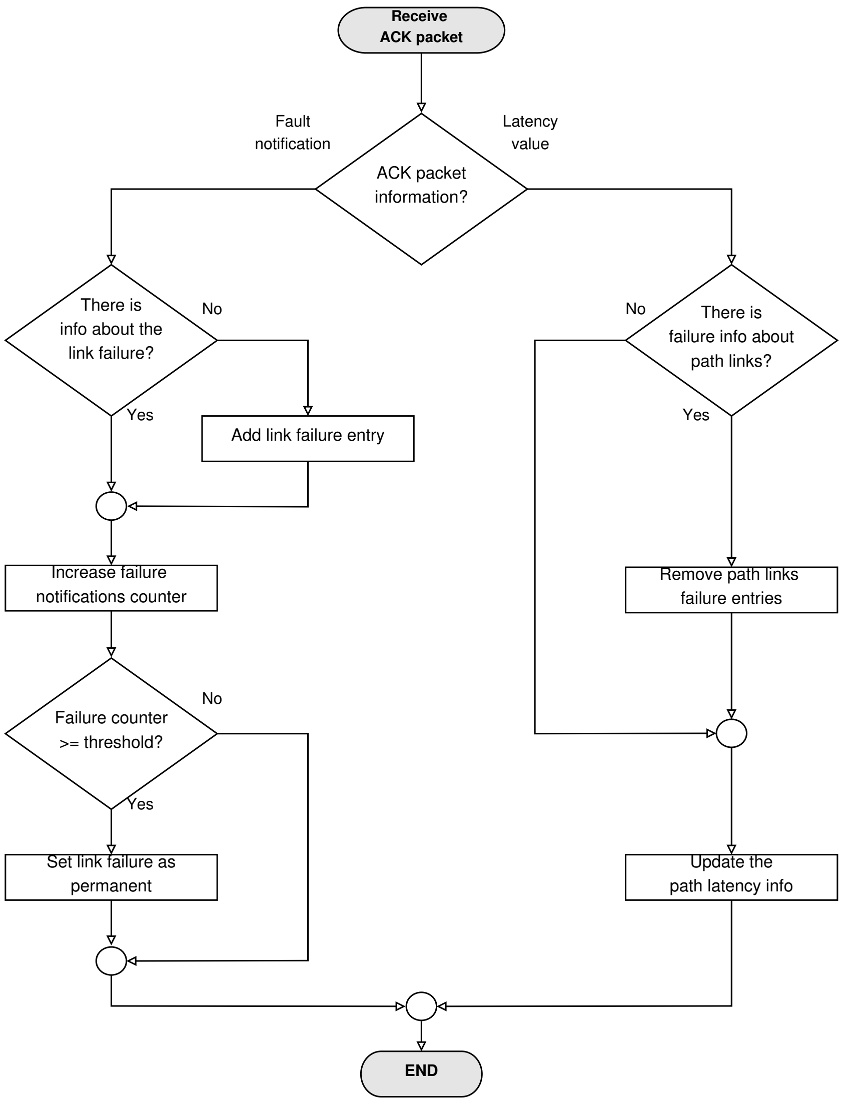

Receive ACK packet
Fault notification
Latency value
ACK packet information?
There is info about the link failure?
No
Yes
Add link failure entry
Increase failure notifications counter
Failure counter >= threshold?
No
Yes
Set link failure as permanent
There is failure info about path links?
No
Yes
Remove path links failure entries
Update the path latency info
END

Figure 3.10: Reception of ACK packets.

There are several design parameters involved in the treatment of transient link failures, that are explained in detail in section 3.6. Among these parameters, the most important are: the method for identifying changes in the state of a failed link; and the method for marking a link failure as permanent. By default, FT-DRB uses:

- An *ACK-based* marking method where link failures are set as *permanent* only after receiving five consecutive ACK notifications regarding the same failure.
- *Source-based probes* for identifying changes in the state of failed links, using real *application messages* as probes. Probes are sent according to the *request-based* approach, where for every 100 messages sent, 99 are sent through alternative paths and 1 (the probe) through the faulty path.

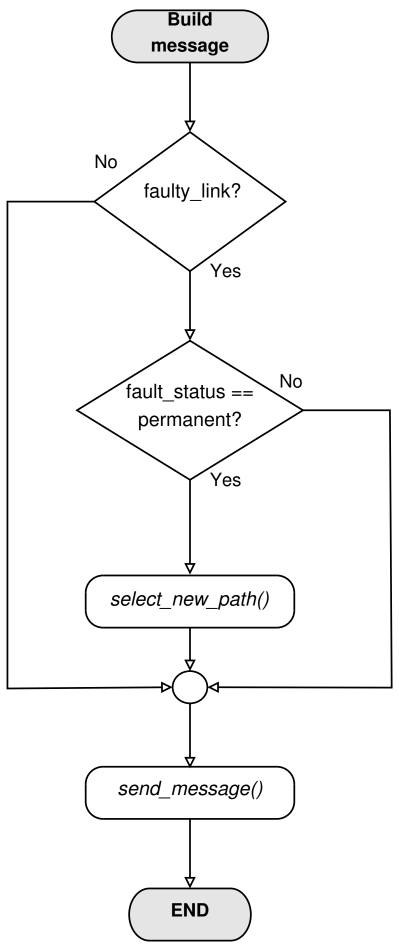


Figure 3.11: Injection of application messages.

3.5 Architecture of Network Components

This section addresses the physical design and implementation of network components for the proposed *Fault-tolerant Distributed Routing Balancing* method. Since the proposed method relies on adaptive routing, the following aspects may affect its normal functioning:

- **Deadlock.** Our method relies on a deadlock avoidance technique designed for interconnection networks suffering from a large number of failures. This technique, called *Non-blocking Adaptive Cycles* is explained in detail in chapter 4.
- **Livelock.** This is probably one of the most difficult problems to solve in fault-tolerant routing algorithms. By definition, FT-DRB does not configure paths of infinite length, therefore, packets always reach their destinations in a finite number of steps[6].
- **Starvation.** This situation is not a problem because FT-DRB does not prevent the injection of packets for indefinite time periods. Moreover, packets cannot be indefinitely blocked at intermediate routers because all packets have equal opportunities to access output links.

As mentioned in previous sections, FT-DRB is based on three major stages: monitoring and notification (section 3.1); selection of escape paths (section 3.2); and configuration of source-based alternative paths (section 3.3). In the following subsections, we discuss implementation issues of each phase in both routers and terminal nodes, together with the structure of FT-DRB packets. Finally, we include a brief discussion about required resources of the method at the end of this section.

3.5.1 Packets Format

In FT-DRB, packets include mechanisms for identifying alternative paths (either escape or source-based) and also for carrying a path latency value or link failure information.

Data packets include a multiple header for storing the additional information about the intermediate nodes used in the *multistep path* (MSP), as shown in Fig. 3.12. In order to simplify the packet format, FT-DRB configures both escape and source-based alternative paths using only two intermediate nodes. If a router detects a link failure along the path, that router locally configures an escape path replacing the information of intermediate nodes in the header, if required; this process is repeated as necessary along the path. In

[6]Note that dead-end *labyrinth-like* fault-regions are not considered by the method, as explained at the beginning of this chapter.

the case of source-based alternative paths, FT-DRB is able to configure three-segment MSPs using two intermediate nodes (in direct networks) for every valid source-destination pair. This approach has been previously adopted in other *DRB-based* routing methods such as [25], [55] and [56].

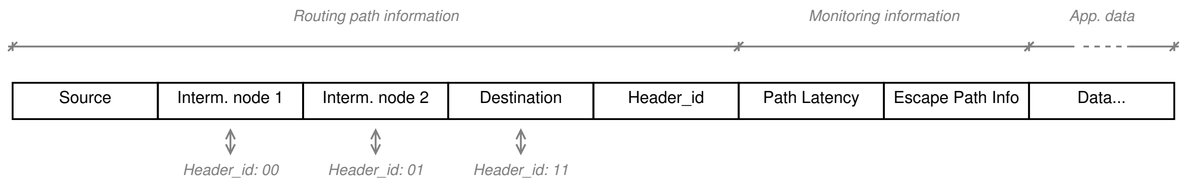


Figure 3.12: FT-DRB Data packet format.

In order to route packets correctly, each router along the source-destination path must be able to identify which of the multiple headers of the packet should be used for routing at each segment of the MSP. For this purpose, the proposed packet format includes two bits, the *Header_id* field in Fig. 3.12, for identifying which header corresponds to each segment. Thus, FT-DRB routers are able to forward packets as any other conventional router, using minimal static routing at each segment of the MSP. When a packet reaches a router identified as one of the intermediate nodes in the packet header, the bits of the *Header_id* are modified to point the new header; then, the packet is routed to the next intermediate node or to the final destination, as appropriate. This situation implies the existence of a *Header Detection and Processing* (HDP) mechanism the in *Routing and Arbitration* (R+A) module of the router, capable of detecting and eliminating the intermediate headers whenever packets reach their final destinations. Therefore, the structure of conventional routers must be modified for implementing FT-DRB; this topic will be covered in subsection 3.5.2. Additionally, data packets include the *Path latency* and the *Escape path info* monitoring fields. The first is an integer-size field used for recording the path latency value; while the *Escape path info* is a bit-size field for marking which data packets have been rerouted through alternative escape paths (to avoid link failures).

As explained in previous sections, FT-DRB uses ACK packets for two different purposes: the notification of link failures (Fig. 3.13); and reporting path latency values (Fig. 3.14). Despite being generated at different stages, both ACKs are virtually identical. Actually, they have only two differences: the value of the source field; and the information carried by the first field of the payload. For failure notifications, the *source* is the router that has detected the link failure; and the *payload* is the *ID* of the failed port. On the other hand, the *source* of a path latency report is the destination node of that specific path; the *payload* of a latency report is, in fact, the latency value.

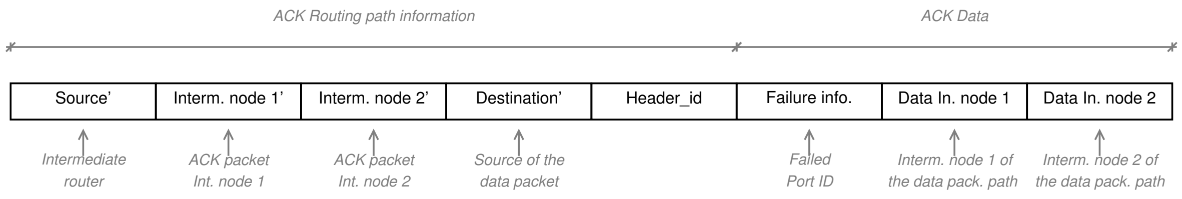


Figure 3.13: FT-DRB ACK packet format (failure information).

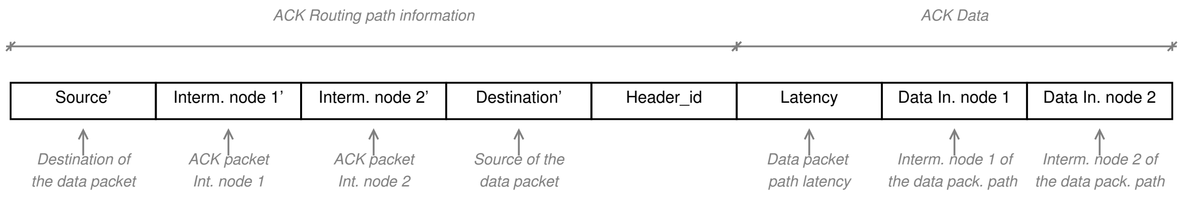


Figure 3.14: FT-DRB ACK packet format (path latency value).

3.5.2 FT-DRB Router

The architecture of the FT-DRB router shares some design characteristics with the *Multipath Distributed Dynamic Routing Balancing* (MD-DRB) [55]. More specifically, both methods include mechanisms for selecting alternative escape paths and injecting notification ACK packets. However, there is a major difference between both proposals: FT-DRB uses these mechanisms for providing fault tolerance; while MD-DRB is not able to route packets in the presence of failures[7]. The FT-DRB Router architecture, shown in Fig. 3.15, includes the following additional modules:

- **Latency Update (LU).** For updating the latency value of enqueued data packets. The waiting time is measured by means of the router *clock*, without requiring any kind of external synchronization.

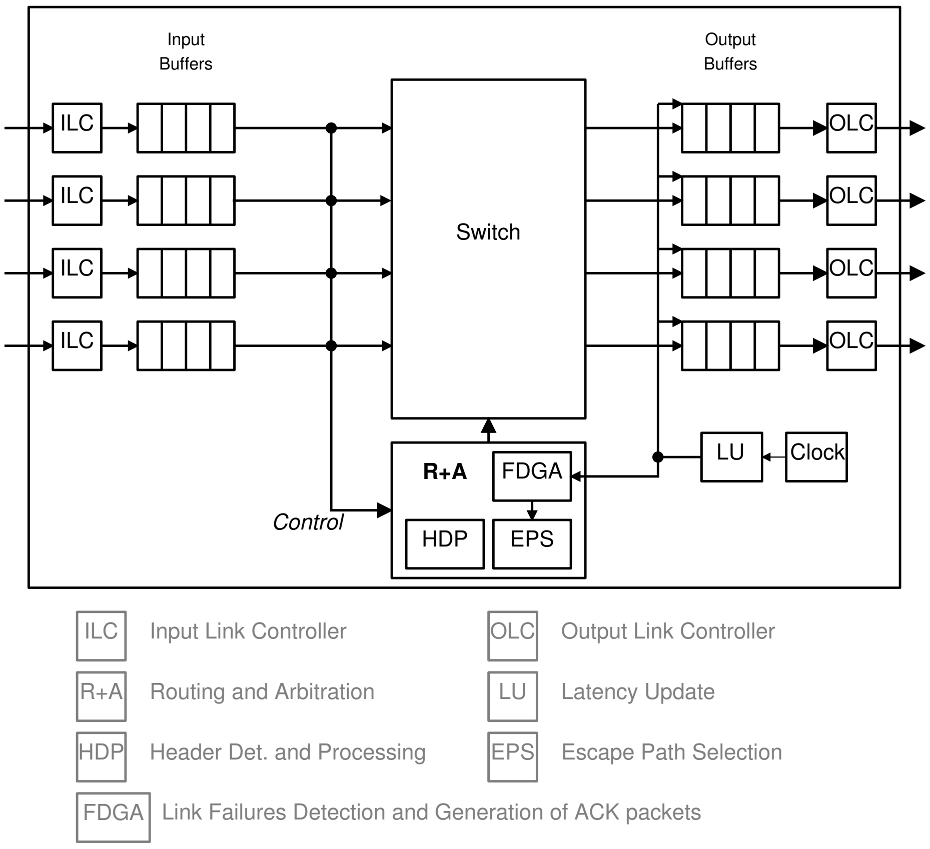
- **Header Detection and Processing (HDP).** This mechanism is part of the *Routing and Arbitration* (R+A) unit. At intermediate routers, it is responsible for modifying the *Header_id* bits of both data and ACK packets. Furthermore, at destination nodes, this unit detects and eliminates the intermediate headers.

- **Link Failures Detection and Generation of ACK packets (FDGA).** This module has been designed with two purposes: detect the existence of link failures, and generate the corresponding notification, as explained in section 3.1. In the

[7]MD-DRB uses escape paths and ACK notifications only for treating congestion problems, relying on the use of virtual escape channels for deadlock avoidance [55].

event of a link failure, this module generates the notification packet and sends it back to the corresponding source node.

- **Escape Path Selection (EPS).** This module implements the configuration of alternative escape paths explained in section 3.2; these alternative paths are used for avoiding link failures at intermediate routers.



Input Buffers
Output Buffers
ILC
Switch
OLC
R+A
FDGA
HDP
EPS
LU
Clock
Control
ILC Input Link Controller
OLC Output Link Controller
R+A Routing and Arbitration
LU Latency Update
HDP Header Det. and Processing
EPS Escape Path Selection
FDGA Link Failures Detection and Generation of ACK packets

Figure 3.15: FT-DRB Router architecture.

Upon receiving a data packet, each destination node first checks the value of the bit contained in the *Escape path info* field, as could be seen in Fig. 3.16. If the data packet has arrived through a fault-free path, i.e. the *Escape path info* bit is not set, the destination node injects the corresponding ACK packet carrying the latency value of the path, as explained in section 3.3. The selection of MSPs and the metapath configuration are performed in the network interface. In FT-DRB, the information obtained from ACK packets (either a failure notification or a latency value) is always used during the metapath configuration process, as shown in Fig. 3.16.

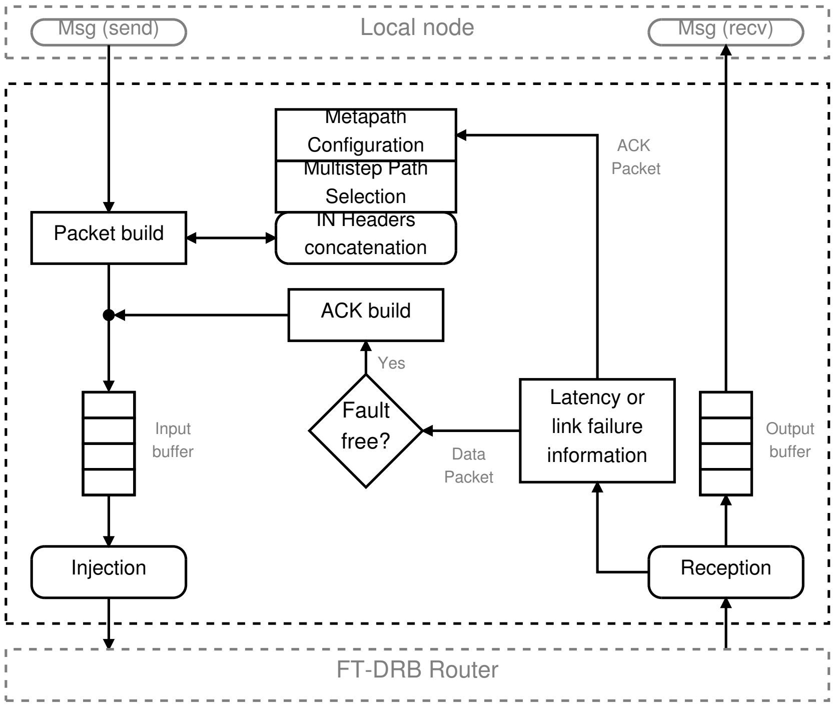


Figure 3.16: FT-DRB Network interface.

3.5.3 Required Resources

Special efforts have been put in the design of FT-DRB for avoiding increases in the critical path and cost of routers. For instance, some of the proposed features are not implemented in routers but in network interfaces, as explained above.

The set of actions applied by FT-DRB at node level have low overheads because they are simple comparisons and accumulations (locally performed) and do not delay send/receive primitives. As explained in previous sections, packets are forwarded with no overheads when output link are fault-free. Moreover, the escape path selection mechanism is invoked only when faults are detected. Similarly latency updates are performed while messages are waiting in router queues. Consequently, these operations are performed concurrently with packet delivery. Furthermore, interconnection networks usually are not designed to continuously operate at their saturation point, thus small overheads could be tolerated to avoid faults (if necessary). Regarding the memory requirements of the metapath configuration and multistep selection processes, FT-DRB only needs to store the latency values of MSPs and their intermediate nodes (in source nodes). The size of these three data values is known in advance and it is a parameter that can be configured by the designer.

3.6 Design Alternatives

In this section, we present some design alternatives for two specific aspects of FT-DRB: the notification of link failures to source nodes; and the treatment of transient faults. We first present the alternatives for the notification of link failures in subsection 3.6.1. Then we address the treatment of transient link failures in subsection 3.6.2.

3.6.1 Link Failure Notification

As explained in section 3.1, the notification approach adopted by default in FT-DRB is based on sending ACK packets from routers affected by link failures back to source nodes. We have proposed an alternative notification approach designed for situations where intermediate routers are not able to inject ACK packets for notifying source nodes about link failures. In this alternative approach, destination nodes are responsible of injecting both notifications about path latency and also about link failures, as appropriate. Consequently, intermediate routers are only responsible for setting up a link error flag in the packet header to mark the path as faulty, before rerouting affected packets through alternative escape paths, as explained in section 3.2. This alternative approach is shown in the diagram of Fig 3.17.

The overall performance of both notification approaches can be further improved by:

- Taking advantage of existing failure notifications. This improvement is based on storing the information about link failures at each intermediate router along the backward path (between affected routers and source nodes). Counting on this information, each "intermediate" source node may be notified about the existence of failures, thus reducing reaction times skipping the procedures for error detection, failures notification and selection of escape paths.

- Generating additional failure notifications. The idea behind this improvement is to send more than one failure notification at the same time. In short, after discovering a link failure, the affected router can check which of the buffered packets needs to access the failed link. Counting on this information, the router is able to send failure notifications to source nodes that have injected those packets.

3.6.2 Permanent and Transient Faults

As explained in section 3.4, there are many design parameters involved in the detection and correct treatment of permanent and transient link failures. These parameters are

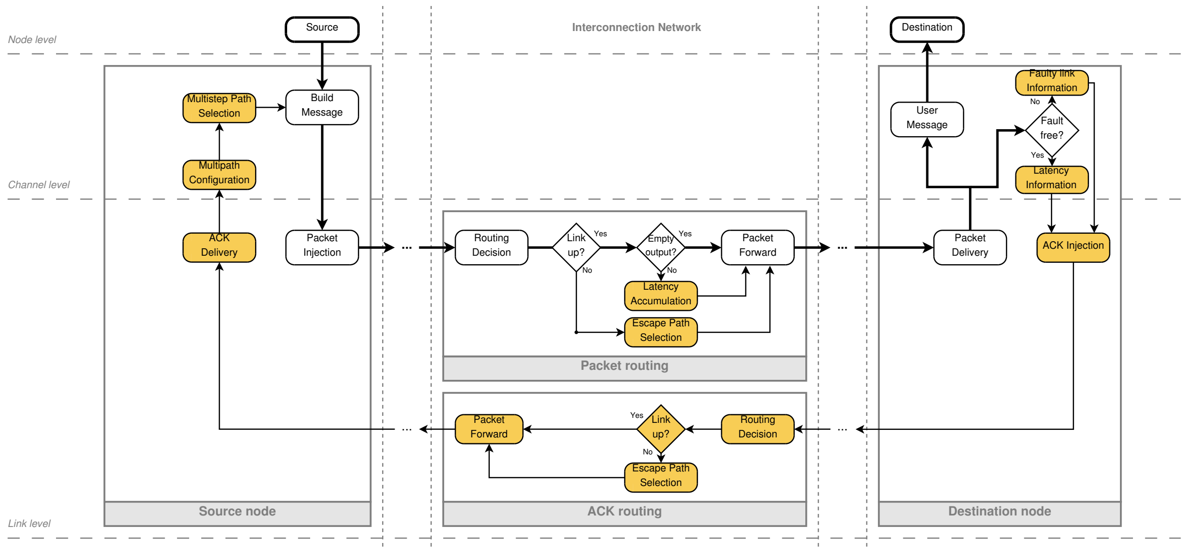


Figure 3.17: Destination-based notification of link failures.

related to the methods for identifying changes in the state of failed links; and also to the procedures for changing the state of a link failure from transient to permanent.

Methods for identifying changes in the state of failed links

When talking about changes in the state of failed links, we refer to the disappearance of transient failures[8], in other works, when after a malfunctioning time the normal functionality of the link is fully restored. There are two main options for identifying changes in the state of links suffering from transient failures:

- Router-based notifications. When adopting this method, intermediate routers are responsible for sending *link-up* notifications to the corresponding source nodes. This approach allows lower identification times based on the assumption that routers are able to identify link-up events. However, as a counterpart, routers should keep a record of which source nodes have to be informed about the availability of the link. To this end, each router need to count on extra memory space together with a sort of replacement algorithm for handling the entries of several source nodes [92], [91].

- Source-based probes. This approach –adopted by default in FT-DRB– is based on sending some probe messages from source nodes to identify changes in the state of failed links. If the failure persists, a probe would be rerouted to the

[8]Note that the state of links affected by permanent failures cannot be further modified. Moreover, changes from active to failed states in links are locally identified by routers along the routing path.

destination through an escape path, and the source node would receive a new link failure notification. In contrast, if the failure has disappeared, the probe would be able to reach the destination node trough the non-faulty path, therefore, the source node would receive an ACK packet carrying the path latency value. Therefore, it does not require any additional hardware in routers.

There are two points to be considered when using source-based probes: what kind of probes to be used; and the timing approach adopted for sending probes. A probe can be either a synthetic or a real application message. The timing approach defines the *tempo* used for forwarding probes. Three different approaches can be adopted:

- A *clock-based* approach, where probes are sent according to real time intervals defined by the source node clock. The time interval is a modifiable parameter.
- A *request-based* approach, where for every n messages sent, $n-1$ are sent through alternative source-based paths, and 1 (the probe) through the faulty path. In this approach, n is a modifiable threshold, defined by default to 100 in FT-DRB.
- An *hybrid scheme* based on the combination of the two previous approaches, where probes are sent according to the time interval or to the request threshold, whichever occurs first. The time interval and request threshold can be modified.

The classification of identifying methods is shown in Fig. 3.18. Furthermore, the timing approaches of the *source-based probes* approach are summarized in Table 3.1.

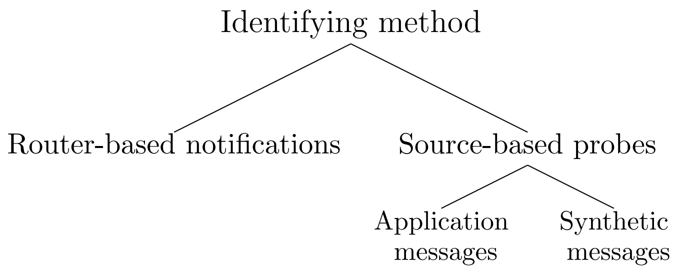


Figure 3.18: Methods for identifying changes in the state of a failed link.

Timing approach	Input parameters
Clock-based	Time intervals
Request-based	Request threshold
Hybrid scheme	Time intervals and Request threshold

Table 3.1: Parameters used in source-based probes.

Procedures for changing the link failure state

This procedure is always applied in source nodes and may be based on the following three approaches:

- *Clock-based.* This approach is based on the use of a predefined time value. If the source node does not receive new ACK packets carrying path latency information during a predefined time period, the source node marks the link failure as permanent. In this approach, the time value is a modifiable parameter.
- *ACK-based.* This approach has been previously explained in section 3.4 because it is used by default in FT-DRB. It is based on counting the number of received failure notifications for a specific link. If the number of notifications is equal or greater than a predefined (but modifiable) threshold value, the state of the link failure is changed from transient to permanent.
- *Hybrid scheme.* A combination of the two previous approaches is also possible, where link failures are marked as permanent elapsed a predefined time period, or after when the number of failure notifications received reaches the threshold value, whichever occurs first.

The procedures for changing the state of link failures are summarized in Table 3.2.

Identifying approach	Input parameters
Clock-based	Time threshold
ACK-based	Notifications threshold
Hybrid scheme	Time and Notifications thresholds

Table 3.2: Procedures for changing the state of link failures.

3.7 Discussion

Throughout this chapter, we have described the proposed fault-tolerant adaptive routing method *Fault-tolerant Distributed Routing Balancing* (FT-DRB). Let us now provide a brief comparison with the most relevant related works in literature.

Although FT-DRB can be compared with any other method, we limit the comparison only to the most relevant to our method. Given the purpose of comparing our proposal, the most relevant works can be classified into three major groups: the *Immunet family* [78],

[74]; the *Routing Methodology Based on Intermediate Nodes* [67], [33]; and the *Routing Methodology for Dynamic Fault Tolerance* [66].

Immunet [78] and *Immucube* [74] rely on dynamic reconfiguration processes to manage failures in parallel systems without shutting down or restarting the system. Additionally, both *Immunet* and *Immucube* require table-based routing and employ virtual channels and *Bubble Flow Control* [76] for deadlock avoidance. FT-DRB is based on a multipath adaptive routing approach and needs no virtual channels for deadlock avoidance. These are the biggest differences between these methods and our proposal.

On the side of the routing methodology proposed by Nordbotten et al. [67] and Gómez et al. [33], the major difference lies on the adoption of the *static fault mode*. In these methods, the machine must be rebooted, new routes calculated and the machine restarted from the last safe state with checkpointing schemes. This problem is not present in our proposal because FT-DRB has been designed to deal with both static and dynamic link failures. However, the counterpart of this adaptability is the need of using a scalable deadlock avoidance technique.

One of the only proposals dealing with dynamic failures is the *Routing Methodology for Dynamic Fault Tolerance* presented by Nordbotten and Skeie [66]. This method is based on *positive-first* routing, a variation of the *turn-model* routing [30], and virtual channels. It achieves good average latency values (about 82%) but packets may be dropped to avoid deadlocks during the dynamic transition from the old to the new routing function. The use of *positive-first* routing and packet drops are the major difference with our method.

Real high-performance computing systems usually rely on very simple solutions. For instance, *BlueGene/L* and *BlueGene/P* supercomputers may route packets either dynamically or deterministically using dimension-ordered routing (*xyz*), but the hardware does not have the capability to round around dead nodes or links [2].

Chapter 4

Scalable Deadlock Avoidance for Fault-tolerant Routing

In this chapter we explain in detail the proposed scalable deadlock avoidance technique *Non-blocking Adaptive Cycles* (NAC), specifically designed for interconnection networks suffering from a large number of failures. This technique has been designed and implemented with the aim of ensuring freedom from deadlocks in the fault-tolerant adaptive routing method presented in the previous chapter.

Non-blocking Adaptive Cycles is a three-stage approach that covers the key aspects of deadlock avoidance. These aspects include the detection of deadlock prone situations, the identification of the routing cycles involved in these situations, and the application of predefined protocols to ensure the normal functioning of the system under these circumstances. Conceptually, the proposal is based on preventing the *hold-and-wait* and *circular wait* conditions [10]. The aim is to deny these two conditions to avoid deadlock occurrences by means of adding an one-slot deadlock avoidance buffer to each input buffer, and applying a simple set of actions when accessing output buffers with no free space.

In the following section, we introduce the problem of deadlock avoidance in interconnection networks affected by link failures. Then, we describe in detail the three-stages approach adopted in *Non-blocking Adaptive Cycles* in section 4.2; *detection of deadlock prone situations* is explained in subsection 4.2.1; *identification or routing cycles* in subsection 4.2.2; and *gradual recovery of movement* in subsection 4.2.3. In addition, a semi-formal applicability proof is outlined in subsection 4.2.4. Finally, the proposal is summarized and discussed in section 4.3.

4.1 Deadlock Avoidance in Faulty Networks

One of the main impediments in the design of routing algorithms for treating dynamic failures is the probability of reaching deadlock configurations. This probability increases every time the routing function is redefined to avoid failures, since new cyclic resources dependencies can be formed. In fact, a routing algorithm may need several direction/dimension changes for avoiding a few link failures. This situation is exemplified in Fig. 4.1, for the source-destination pair *S-D*. In the example, a packet needs to change routing direction five times before reaching the destination node (because of link failures), while in a fault-free scenario the same packet can reach the destination node using a simple and minimal routing algorithm such as *DOR* [22, Ch. 4].

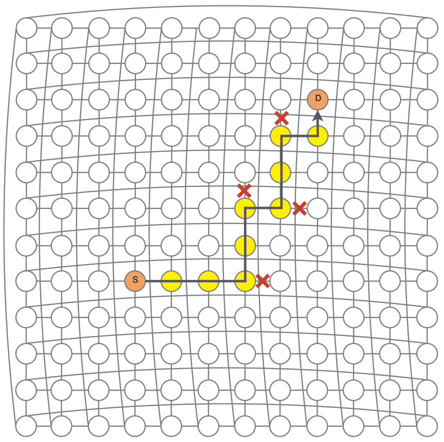
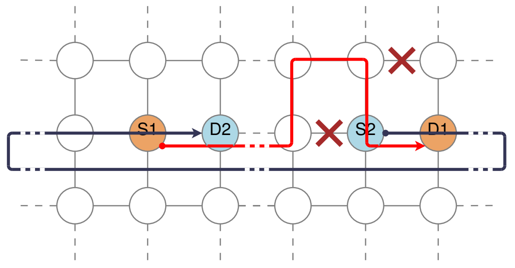


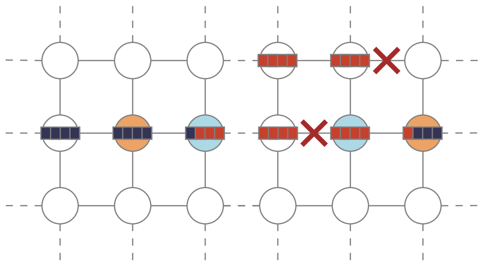
Figure 4.1: Example of direction changes in routing functions.

Unfortunately, deadlocks are far from rare when dealing with faulty networks because a few faults may generate cyclic dependencies, even under low traffic conditions. In the context of interconnection networks, deadlock occurs when some packets cannot advance because the buffers requested by them are full. In this situation, every packet is requesting resources held by other packets while holding resources requested by other packets (in a *circular wait* configuration). Therefore, a set of packets is blocked forever [22, Ch. 3].

An example of a deadlocked configuration is presented in Fig. 4.2. In this example, the communication between the two pairs of source-destination nodes *S1-D1* and *S2-D2* is interrupted by two link failures. In order to ease the visualization, Figs. 4.2(a) and 4.2(b) only show the portions of the network needed for understanding the possible deadlock situation. Consider such network as a bidimentional torus, where dots represents the set of omitted network nodes and links. Routing paths are represented as solid lines between the source-destination pairs in Fig. 4.2(a); the dark line corresponds to the *S2-D2* pair



(a)



(b)

Figure 4.2: Example of a deadlocked configuration.

and the light one to the *S1-D1* pair. A similar color scheme is assumed for packets in the buffers of Fig. 4.2(b).

In the example presented in Fig. 4.2(a), packets traversing the *S1-D1* path change routing direction to avoid the failure in the original path. Later on, the same group of packets return to the original routing direction to reach the destination node *D1*. This behavior may lead to a cyclic dependency when interacting with other pairs of source-destination nodes. From this cyclic dependency, a deadlock configuration would arise when all the buffers involved in the communication cycle have exhausted their available buffer space, as shown in Fig. 4.2(b). It is possible to infer from the example of Fig. 4.2 that deadlock occurrences are not rare and constitute a serious problem since a single fault may render inoperative the whole computing system.

4.2 Non-Blocking Adaptive Cycles

Since deadlock occurs when a set of packets cannot advance because the buffers requested by them are full, it can be avoided by preventing the saturation of such buffers. This is equivalent to ensure that the *hold-and-wait* and *circular wait* conditions are not satisfied [10]. A *hold-and-wait* condition is a situation where a packet requests resources held by other packets while holds resources requested by other packets. On the other hand, in a *circular wait* condition, circular chains are formed along the network where packets hold resources that are being requested by the next packet in the chain. The aim of our proposal is to deny these two conditions to avoid deadlock occurrences.

In order to clarify the ideas we are proposing, an intuitive approach will be employed to exemplify the ideas used in the deadlock avoidance technique. First of all, consider an arbitrary cycle of dependencies with no free slots in any buffer along the cycle, except at the local router. Furthermore, consider buffers as one-slot FIFOs. This situation is outlined in Fig. 4.3, where the central router of each sub-figure represents the local router, and circles represent buffers (input at the left, output at the right and a simplified deadlock avoidance buffer at the bottom). A white circle represents an empty buffer, while colored circles represent buffers storing packets from different source-destination pairs. We will take the situation of Fig. 4.3(a) as the initial configuration before applying the set of actions proposed in our technique. Notice that the situation of Fig. 4.3(a) may lead to a deadlocked configuration since there is only one free slot along the cycle. As a first step in the process of providing a solution, the packet stored in the input buffer of the local node is moved to the deadlock avoidance buffer (Fig. 4.3(b)) and the injection into the full output buffer is temporally stopped. After applying the previous actions, the input buffer of the local router is free to store a new packet of the cycle, thus allowing packets in the cycle to move towards their destination. As shown in Fig. 4.3(c), a new incoming packet is stored in the input buffer. This action is intended to guarantee that at least one packet is able to progress along the cycle, therefore the output buffer of the local router will eventually be released, as in Fig. 4.3(d). At that point, the deadlock avoidance buffer can be also freed (Fig. 4.3(e)), thus returning to the normal system functioning (Fig. 4.3(f)).

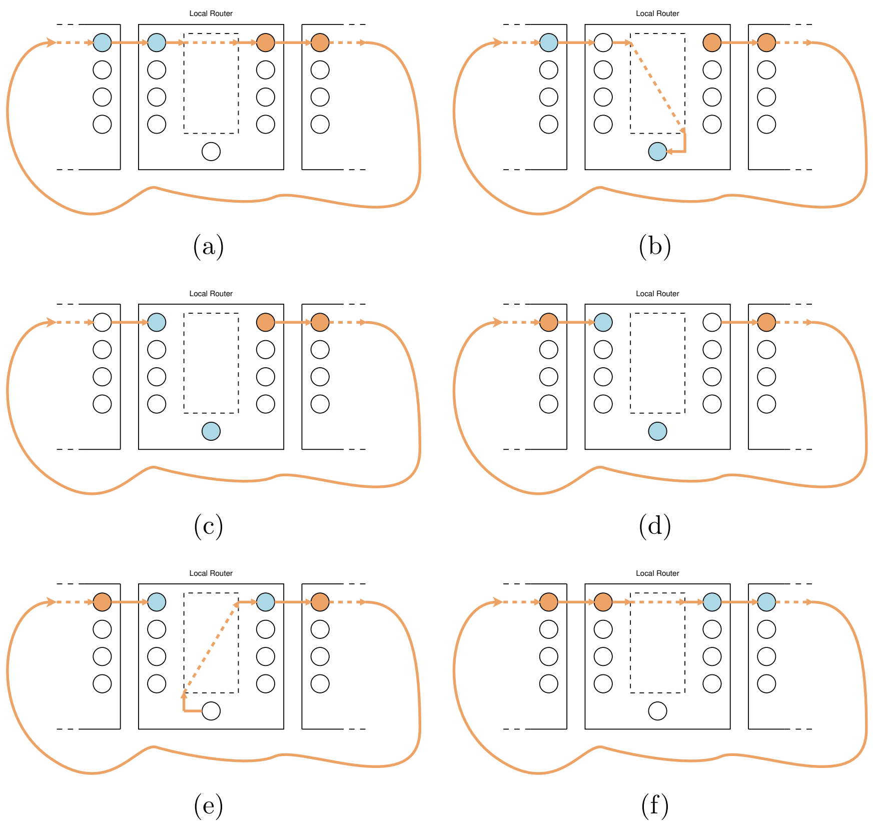


Figure 4.3: Example of the deadlock avoidance technique.

The *Non-blocking Adaptive Cycles* (NAC) technique exploits this idea by means of adding an one-slot deadlock avoidance buffer to each input buffer, and applying a simple set of actions when accessing output buffers with no free space. These actions are only applied under specific circumstances directly related to the free space left in the buffers of the local router as well as in the next router in the path along source-destination pairs. It is worth noting that each router already knows its buffer space availability as well as the available space at the input buffer of all its neighbors by means of the flow control mechanism (i.e. credit-based systems). This is part of the information that our technique needs for avoiding deadlock occurrences. A simplified router architecture diagram of our proposal is shown in Fig. 4.4. In addition, the combine router architecture of FT-DRB and NAC proposals is shown in Fig. 4.5. The summary of the specific notation used in the rest of the chapter is given in both Table 4.1 and Fig. 4.6. In the context of this thesis, we assume *Virtual Cut-Through (VCT)* as the switching technique of the system and a credit-based flow control. Furthermore, we consider that every network node (i.e. every router) is able to inject point-to-point flow control packets at anytime, and also that such action is independent and asynchronous with respect to the routing of application messages.

NAC is composed by three main stages: the detection of deadlock prone situations; the identification of the routing cycles involved in these situations; and the application of predefined protocols to ensure the normal functioning of the system under these circumstances. These three stages are explained in subsections 4.2.1, 4.2.2 and 4.2.3, respectively.

4.2.1 Detection of Deadlock Prone Situations

A set of triggering conditions has been defined in order to detect situations where the network is in a state prior to a deadlocked configuration. These conditions occur when a full input buffer in one node tries to choose a routing direction of which both the output and the input buffers it connects to are full, as defined in Eqs. 4.1, 4.2, and 4.3; and is graphically shown in Fig. 4.7.

$$size(c_{k,j}^{i,m}) \; = \; cap(c_{k,j}^{i,m}) \tag{4.1}$$

$$output(a_j^i) = b_k^i \tag{4.2}$$

$$size(a_j^i) = cap(a_j^i) \tag{4.3}$$

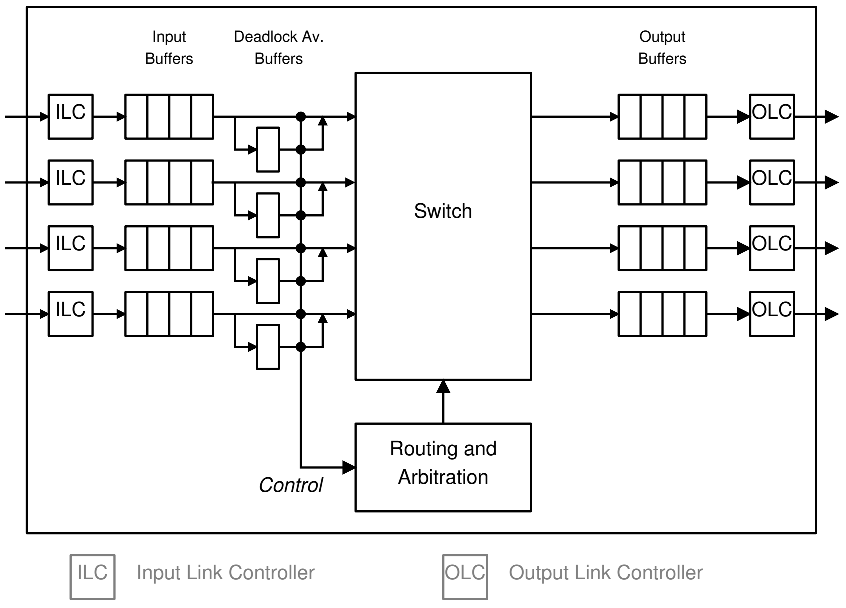


Figure 4.4: NAC Router.

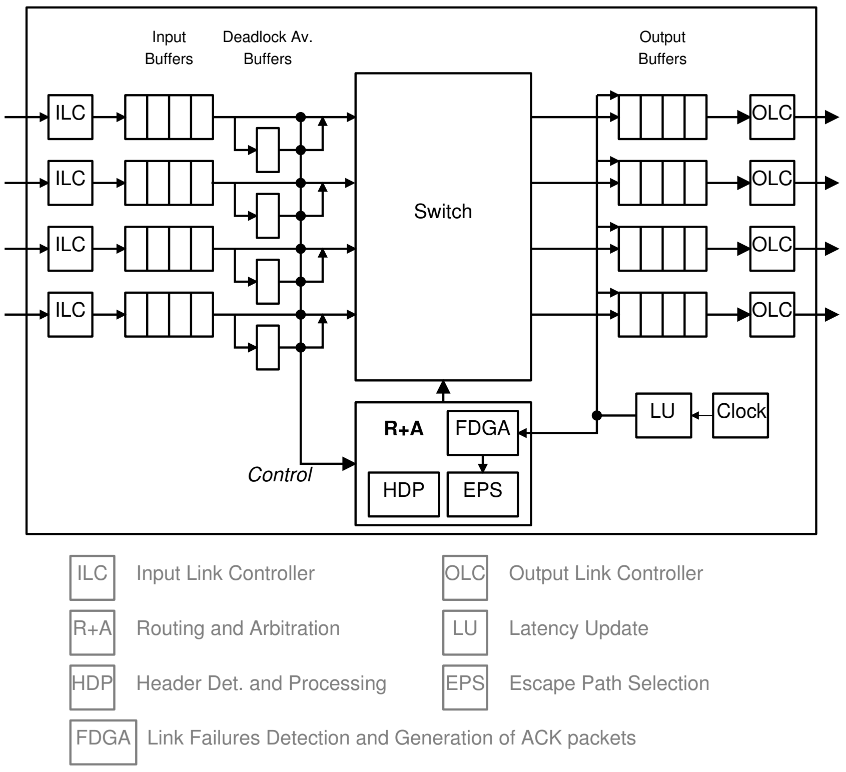

Input Buffers
Deadlock Av. Buffers
Output Buffers
ILC
ILC
ILC
ILC
Switch
OLC
OLC
OLC
OLC
R+A
FDGA
HDP
EPS
LU
Clock
Control
ILC Input Link Controller
OLC Output Link Controller
R+A Routing and Arbitration
LU Latency Update
HDP Header Det. and Processing
EPS Escape Path Selection
FDGA Link Failures Detection and Generation of ACK packets

Figure 4.5: NAC + FT-DRB Router.

Notation	
r^i	a network node (router),
a^i_j	an input buffer of router r^i,
b^i_k	an output buffer of router r^i,
d^i_j	a deadlock avoidance buffer of router r^i,
cap(*arg*)	the capacity of the buffer *arg*,
$size$(*arg*)	the number of packets in the buffer *arg*,
output(*arg*)	the output buffer b^i_k assigned to the first packet in the buffer *arg*.
$c^{i,m}_{k,j}$	a logic buffer composed by the output buffer b^i_k and the input buffer a^m_j linked to it.
Operators	
move(*arg1*, *arg2*)	Removes the first packet in the buffer *arg1* and enqueues it in the buffer *arg2*.
stop_st(*arg*)	Stops injection of packets in the buffer *arg*.
start_st(*arg*)	Allows injection of packets in the buffer *arg*.
stop_fw(*arg*)	Stops packet forwarding from the buffer *arg*
start_fw(*arg*)	Allows packet forwarding from the buffer *arg*.

Table 4.1: Summary of notation and operators used in NAC.

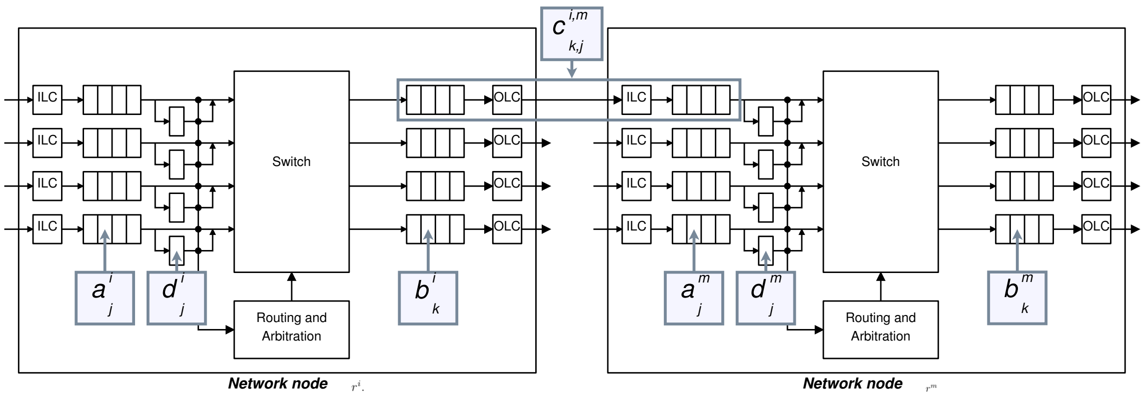


Figure 4.6: NAC Notation.

Conditions (4.1), (4.2) and (4.3) are evaluated locally at each router r^i before forwarding packets in each routing turn. If all three conditions are met, the router is in a state prior to a deadlocked configuration. Notice that such state does not represent a deadlocked configuration because there are still available resources –free slots in deadlock avoidance buffers– that may be used [14, Ch. 14]. Under these circumstances, the following set of actions is applied locally at router r^i as the first step to avoid deadlock occurrences:

1. Stop the injection of packets in the logic buffer $c_{k,j}^{i,m}$.
   Action: **stop_st**(b_k^i) / $b_k^i \in c_{k,j}^{i,m}$

2. Stop packets forwarding from the input buffer connected to local processing nodes (when applicable).
   Action: **stop_fw**(a_j^i) $\forall$ a_j^i *linked to a local processing node*

At this point, the router r^i must identify which routing cycles are involved in the *deadlock prone* situation to avoid real deadlocked configurations. The identification process is explained in detail in the following subsection.

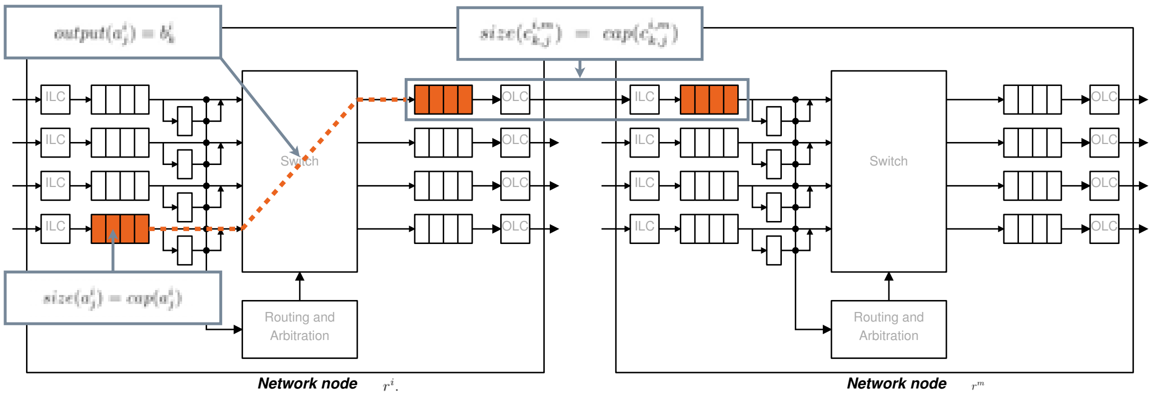


Figure 4.7: NAC Triggering conditions.

4.2.2 Identification of Routing Cycles

The goal of this process is to provide a reliable identification of the input and output buffers (a_j^i, b_k^i) involved in routing cycles that may lead to deadlock configurations. This process takes advantage of point-to-point flow control packets to identify the beginning and the end of cycles in a specific router.

Conceptually, a router r^i may need to identify the correct cycle for two different reasons. One of these reasons is the detection of triggering conditions (4.1), (4.2) and (4.3). In this case, the identification process is intended to ensure the access to the deadlock avoidance buffer d_j^i by the appropriate input buffer a_j^i (to deny the *circular-wait* condition). The second reason is to enter the *gradual recovery protocol*, presented in subsection 4.2.3, where the identification process is applied to avoid false-positive situations during the recovery protocol. In these case, the recovery conditions (4.4), (4.5), (4.6) and (4.7) are evaluated instead of the triggering conditions (4.1), (4.2) and (4.3).

In order to identify a routing cycle, a router r^i composes and sends back a new identification flow control packet to the output buffers $b_k^h \in c_{k,j}^{h,i}$ of routers r^h linked to the input buffers a_j^i where the triggering or recovery conditions are met, as appropriate. Each one of these new flow control packets carries the *ID* of the router r^i which has created the packet, the *ID* of the input buffer a_j^i of the router r^i where the conditions are met, and the *ID* of the output buffer b_k^i of the router r^i where the packet has to be received if part of the cycle (i.e. the b_k^i involved in the triggering or recovery conditions).

Upon receiving one of these special flow control packets, each router r^h locally verifies the triggering or recovery conditions. If at least one input buffer a_j^h meets these conditions, the router r^h replicates the incoming flow control packet to the appropriate buffers. Otherwise, the router drops the packet.

While identification takes place, the remaining application packets are routed normally. If the flow control packet arrives to the router r^i through the correct output buffer b_k^i, the pair of input and output buffers (a_j^i,b_k^i) identified by this packet are part of the routing cycle prone to generate a deadlock configuration. Then, a **move**(a_j^i, d_j^i) operation is performed in order to free a one-slot position in the input buffer a_j^i. This action will introduce a new free space in the routing cycle, allowing packets to move along the cycle towards their destinations (according to the protocol described in subsection 4.2.3). By contrast, if the correct flow control packet has not been received, this may be caused by two situations:

- The packet has been sent through the correct input buffer a_j^i but there is at least one router r^h where the triggering or recovery conditions are not met (so the packet has been dropped). Therefore, the routing cycle is not full and there are no deadlock configurations (but congestion) along the cycle.
- The packet has been sent through an input buffer a_j^i that is not involved in the routing cycle. No actions must be taken.

In summary, the entire set of actions presented at this subsection constitute a sort of enlightenment process intended to facilitate the identification of routing cycles.

4.2.3 Gradual Recovery of Packet Forwarding

After a deadlock prone cycle has been identified, a set of actions must be taken to ensure the gradual recovery of packet forwarding along the routing cycle (until the normal functioning conditions are met).

Once the corresponding $\mathbf{move}(a^i_j,\ d^i_j)$ action has been applied at the router r^i, a new flow control packet has to be sent from that router in order to cause the router r^h to enter into the *gradual recovery protocol.* This new flow control packet carries a single signal used to notify the router r^h that the new space in the output buffer b^h_k correspond to the "deadlock avoidance space" generated by the router r^i. Based on this notification, the router r^h is able to distinguish between these two situations:

- The space availability was caused by the disappearance of a deadlock prone situation. **The router r^{i-1} may continue under normal functioning.**
- The space availability was caused by the $\mathbf{move}(a^i_j,\ d^i_j)$ action applied at router r^i. **The router r^{i-1} must start the *gradual recovery protocol.***

After receiving the flow control packet with the signal to enter the *gradual recovery protocol*, the router r^h has to locally identify which input buffer a^h_j is involved in the routing cycle. This action is critical because each router along the cycle has to apply the $\mathbf{move}(a^h_j,\ d^h_j)$ action to the correct input buffer. Otherwise the cycle may reach a deadlocked configuration. In order to identify the correct cycle, each r^h applies the set of actions presented in subsection 4.2.2 provided that all the four recovery conditions defined in Eqs. 4.4, 4.5, 4.6, and 4.7 are met. The set of recovery conditions is graphically shown in Fig. 4.8.

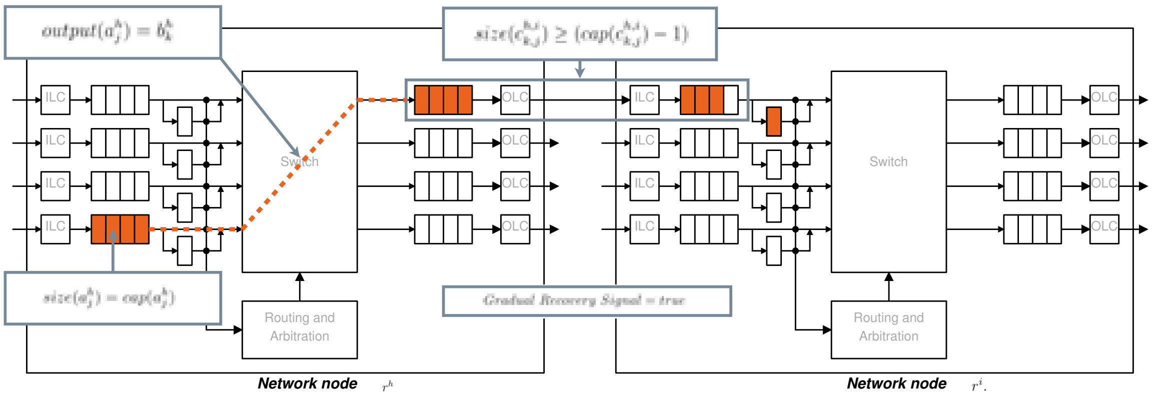
$$size(c^{h,i}_{k,j}) \geq (cap(c^{h,i}_{k,j}) - 1) \tag{4.4}$$

$$output(a^h_j) = b^h_k \tag{4.5}$$

$$size(a^h_j) = cap(a^h_j) \tag{4.6}$$

$$Gradual\ Recovery\ Signal = true \tag{4.7}$$

When at least one of these four conditions is not met, the normal functioning conditions of the router r^h may be restored. First of all, injections to the logic buffer $c^{h,i}_{k,j}$ are re-allowed by means of the $\mathbf{start\_st}(b^h_k)$ action. Packets buffered at deadlock avoidance buffers d^h_j



$output(a_j^h) = b_k^h$
$size(c_{k,j}^{h,i}) \geq (cap(c_{k,j}^{h,i}) - 1)$
ILC
OLC
Switch
$size(a_j^h) = cap(a_j^h)$
Routing and Arbitration
Gradual Recovery Signal = true
Network node r^h
Network node r^i

Figure 4.8: NAC Recovery conditions.

are firstly forwarded to their destinations. After that, injections from the local *processing node* are allowed again applying **start_fw**(a_j^h).

In most situations the initial identification process creates a sort of chain reaction leading to an overlap and reduction of the overall identification time. Under these circumstances the time for treating the deadlock prone situation is given by Eq. 4.8, where n is the number of network nodes involved in the cycle and and t_f represents the clock cycles required to replicate a flow control packet (replication $\neq$ injection). In this situation, there are two *runs* around the identified routing cycles, that corresponds to the value 2 in Eq. 4.8. The first run around the cycle corresponds to the *identification* of the routing cycle; and the second run corresponds to the *recovery* of packets forwarding. The worst case scenario (in terms of timing) occurs when each router along the cycle starts the identification of the routing cycle just after receiving the *Gradual Recovery Signal* of Eq. 4.7. In this case, considering a pessimistic approach, the upper bound is given by Eq. 4.9.

$$NAC\ Av.\ Time = 2 * n * t_f \tag{4.8}$$

$$NAC\ Worst\ Time = \sum_{i=1}^{n} n * t_f \tag{4.9}$$

A simplified graphical example of the two runs around a 5 nodes routing cycle is shown in Fig. 4.9, where the *right-to-left* arrows represents the point-to-point flow control packets. The numbers corresponds to the ID of the router that has generated the packet, and the *Rn* to the recovery signal. The first run –upper part of the graphic– corresponds to the *identification* process that ends when a node receives its own identification packet (in the example the node 1). The second run –lower part of the graphic– corresponds to the *recovery* process, where each node sets the recovery signal downwards to its neighbor node.

This last run ends when the first node in the cycle (i.e. node 1) receives the recovery signal from the last node in the cycle (i.e. node 5).

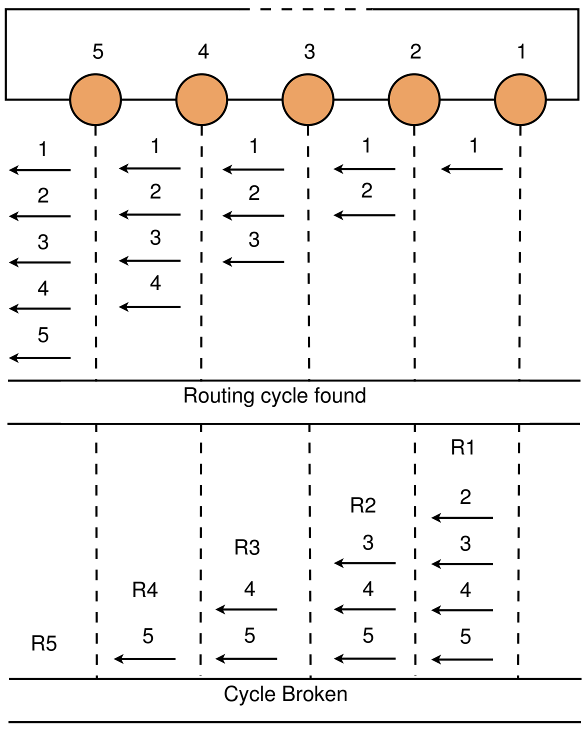


Figure 4.9: Example of identification and recovery processes.

4.2.4 Applicability Proof

The *Non-blocking Adaptive Cycles* technique depends mainly on the process of routing cycles identification (explained in subsection 4.2.2) for detecting deadlock prone situations. From the mathematical point of view, the architecture of a network can be represented as a graph[1] where the vertices represent the network devices or processing nodes, as appropriate, and the edges represent the links that connect them. If considering logic buffers $c_{k,j}^{i,m}$ as nodes, and the internal connections of the router (commonly implemented by means of switches or crossbars) as the edges, it is possible to obtain an equivalent graph representation of the network, as shown in Figs. 4.10 and 4.11. Given this last representation, it is possible to consider the process of routing cycles identification of NAC as a *breadth-first search* (BFS). Then, taking into account that BFS are commonly used for computing a cycle in graph or reporting that no such cycle exists, it is possible to infer the applicability of the routing cycles identification process and, consequently, the applicability of the *Non-blocking Adaptive Cycles* approach.

[1]Abstract representation of a set of vertices or nodes, and edges [37]

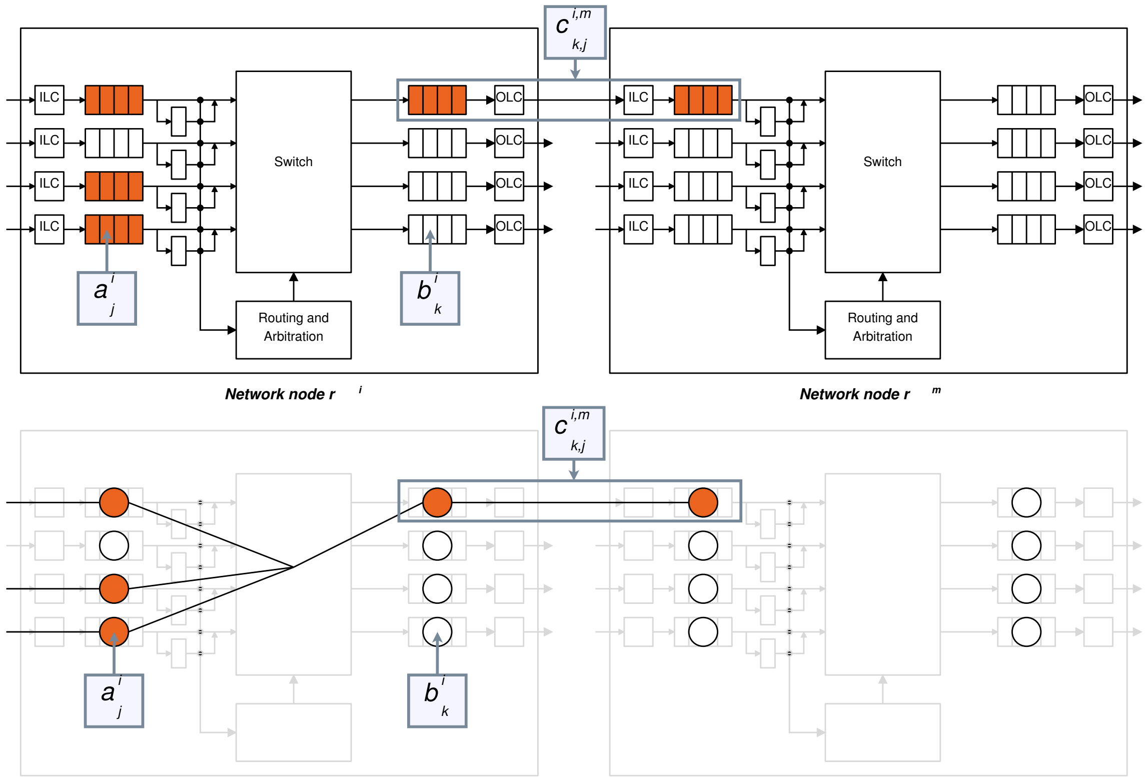


Figure 4.10: Equivalent graph representation of network nodes.

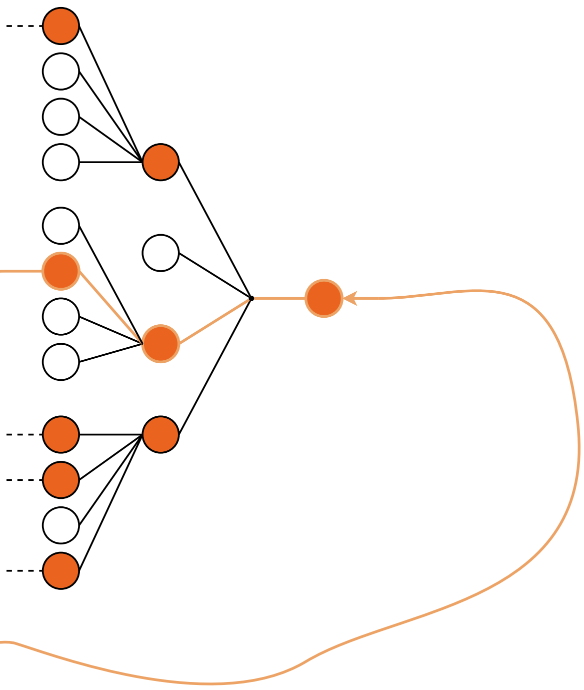


Figure 4.11: Graph representation of a deadlock prone routing cycle.

4.3 Discussion

In this chapter we have presented a complete deadlock avoidance technique designed with the aim of solving the problem of deadlock avoidance for interconnection networks suffering a large number of failures. This problem arises from the scalability limitations of current deadlock avoidance techniques, particularly those based on the use of virtual escape channels.

The proposed technique does not require the use of virtual escape channels thus avoiding their scalability problems. This is one of the advantages of NAC because the resources needed to avoid faults is independent from the number of faults to be tolerated. Additionally, the reduction of the number of virtual channels may increase routers speed.

When networks are operated near saturation points, NAC can introduce some internal overhead in network devices since some congestion conditions may force one or more routers to begin the identification process explained in section 4.2.2. Under these circumstances, unnecessary identification flow control packets are injected into the network. However, it is important to notice that those identification processes do not worsen the congestion problems of the networks because NAC is based on *point-to-point* flow control packets that do not compete for buffering resources.

Some possible solutions to these performance problems are:

- Including a reduced number of virtual channels, in order to improve performance, avoiding situations such as *Head-Of-Line (HOL) blocking* [14, Ch. 19].
- Implementing deeper buffers to reduce the probability of deadlock prone situations (and also congestion).
- Redefining the condition (4.1) as $(size(c_{k,j}^{i,m}) \geq (cap(c_{k,j}^{i,m}) - (j-1)))$ where j is an upper bound given by the number of input buffers. This action would guarantee at least one free slot for each input buffer not linked to the local processing node thus reducing the number of routing cycles identifications.

As closure, we can conclude that the *Non-blocking Adaptive Cycles* technique allows the design of fault-tolerant routing algorithms capable of treating a large number of dynamic failures. This poses NAC as a feasible solution to the problem of deadlock avoidance for current HPC systems.

Chapter 5

Evaluation of Proposals

In this chapter, we present the evaluation of the *Fault-tolerant Distributed Routing Balancing* (FT-DRB) method and the *Non-blocking Adaptive Cycles* (NAC) technique we have explained along the previous chapters. This evaluation aims to corroborate that our proposals allow interconnection networks to perform in the presence of link failures, using network models as the main testing tool. It is worth noting that the most significant information about a network's fault tolerance is whether it can function at all in the presence of faults.

There are three aspects to be addressed to carry out the evaluation, namely: the *simulation models* that represent the system under study, e.g. the interconnection networks; the *workloads* used as inputs of the simulation models; and the *metrics* used to assess the benefits of the proposal. The purpose of the evaluation process is to confirm the operation of FT-DRB and NAC as they were described in previous chapters. To this end, we have defined a set of metrics to observe both functional and performance features of both proposals. These metrics have been chosen with the aim of measuring several aspects related to the capability of interconnection networks to perform in the presence of links failures. The most representative metrics are the average network *latency* and *throughput*. In addition, some significant workloads have been included in the evaluation of the proposals, enabling a reliable analysis of interconnection networks. Both proposals have been implemented using accurate *InfiniBand-based* simulations models.

In the following sections, we describe the three main aspects of the evaluation process; *workloads* in section 5.1; *simulation models* in section 5.2; and *metrics* in section 5.3. Finally, we present the evaluation method and results of each proposal separately. The evaluation method used in NAC and FT-DRB are explained within section 5.4, most precisely in subsections 5.4.1 and 5.4.2, respectively. Test scenarios and results of each proposal are presented and discussed in sections 5.5 (NAC) and 5.6 (FT-DRB).

5.1 Workloads

The evaluation methodology, and the thesis itself, is influenced by both the theory of interconnection networks and fault tolerance. Consequently, we have included two different kinds of synthetic workloads with the aim of evaluating both aspects of the proposed routing method and deadlock avoidance technique.

We have used a collection of application-inspired *performance benchmarks* that describe conditions frequently found in scientific applications. These benchmarks have been applied as inputs during the evaluation of both NAC and FT-DRB. The set of *synthetic traffic patterns* used in these benchmarks is explained in subsection 5.1.1. In addition, an approach based on *dependability benchmarks* [42] have been included in the evaluation of FT-DRB to characterize the system behavior in the presence of faults, as explained in subsection 5.1.2.

5.1.1 Synthetic Traffic Patterns

Synthetic traffic patterns have been used because they are commonly applied in computational intensive scientific applications. These patterns take into account permutations that are usually performed in parallel numerical algorithms [22, Ch. 9], [14, Ch. 3]. The destination nodes for messages generated by a given node are always the same. Concretely, the traffic patterns used are: *Bit reversal*, *Perfect shuffle*, *Butterfly*, *Matrix transpose*, *Complement*, and *Tornado*. Their mathematical descriptions are shown in Table 5.1, where source and destination nodes are denoted as s and d, respectively; and n is the number of bits used to represent the nodes. The *Uniform* traffic pattern has been also included in the evaluation. In this pattern, each node randomly selects its destinations.

Pattern	Destination	
Bit reversal	$d_i = s_{n-i-1}$	$\forall i : 0 \le i \le n-1$
Perfect shuffle	$d_i = s_{(i-1)\ mod\ n}$	$\forall i : 0 \le i \le n-1$
Butterfly	$d_{n-1} = s_0,\ d_0 = s_{n-1}$	$\forall i : 0 \le i \le n-1$
Matrix Transpose	$d_i = s_{(i+\frac{1}{2})\ mod\ b}$	$\forall i : 0 \le i \le n-1$
Complement	$d_i = \overline{s_i}$	$\forall i : 0 \le i \le n-1$
Tornado	$d_i = s_{i+([\frac{k}{2}]-1)\ mod\ k}$	$\forall i : 0 \le i \le n-1$

Table 5.1: Mathematical description of synthetic traffic patterns.

In addition, two collective communication patterns have been also included: *One-to-All (Scatter)* and *All-to-One (Gather)*. In the *One-to-All* pattern, one source s_i

delivers different packets to different destinations d_j $\forall j \neq i$. This is also referred to as *personalized broadcast*. In the *All-to-One* pattern, different packets from different sources are concatenated together for a sole destination.

5.1.2 Availability Traces of Real Systems

Dependability benchmarks are based on the availability traces of real parallel and distributed systems, downloaded from the public *Computer Failure Data Repositories* (CFDR) [98] and *Failure Trace Archive* (FTA) [29].

These availability traces have been obtained from real systems belonging to the *Los Alamos National Laboratory* (LANL) [49] and the *Pacific Northwest National Laboratory* (PNNL) [73]. These traces are composed by hundreds of failure records that contain the time when the failure started, the time when it was resolved, the system and node affected, the type of workload running on the node and the root cause [82]. Some examples of root causes are: *Human* error; *Environment*, including power outages or A/C failures; *Network* failure; *Software* failure; and *Hardware* failure.

Four computing systems have been chosen to be simulated, taking into account their number of nodes and network failures, as detailed in Table 5.2. The entire set of availability traces of these systems has been parsed to obtain the relevant data for performing the evaluation of FT-DRB. The set of processed attributes is summarized in Table 5.3.

Machine	Nodes	Procs.	Net Faults	Trace duration
LANL 12	512	1024	52	09/2003-11/2005
LANL 18	1024	4096	62	05/2002-11/2005
LANL 19	1024	4096	58	10/2002-11/2005
PNNL MPP2	980	1960	89	11/2003-09/2007

Table 5.2: Characteristics of the systems of the availability traces.

Attribute	Description
resolution	Resolution of the traces in seconds
node_id	Unique ID for the node
event_type	Type of event (0: unavailability, 1: availability)
event_start_time	Start of the event (UNIX epoch time)
event_end_time	End of the event (UNIX epoch time)
event_end_reason	Reason the event type or state changed at the end of this trace (Network code range: 3000-3999)

Table 5.3: Attributes of availability traces used in the evaluation of FT-DRB.

5.2 Network Models

The simulation environment is provided by the commercial modeling and simulation tool OPNET Modeler [70]. This standard tool gives support for modeling communication networks, and allows faults injection in model components. It is suitable to design and analyze communication equipment and network protocols, also improving product performance and reliability. OPNET Modeler is endowed with a three level hierarchy for modeling purposes, namely: network, node, and process levels. Network level includes nodes, links, and subnets interconnected between them, and composing topologies. At this level, models attributes are set and parametric simulations are configured.

At node level, network components are represented by using modules with such features as: Messages processing (creation, transmission, reception, and storage); and internal routing, content analysis, queuing, multiplexing, etc. Modules typically represent applications, protocol layers, and physical resources, such as buffers, ports, and buses. Finally, the behavior of modules is programmable via their process models. They consist of finite state machines (FSM) containing blocks of enhanced C/C++ user code and OPNET Kernel Procedures (OKP). Finite state machines respond to interrupts generated by the simulation kernel and support detailed specification of protocols, resources, applications, algorithms, and queuing policies. Users can specify link parameters such as bandwidth, bit error rate, propagation delay, packets supported, as well as other attributes. The simulation environment provides a Discrete Event Simulator (DES) engine. The simulation kernel handles a single global event list and a shared simulation time clock. Events are attended from the list in the appropriate time order. The whole actions and functionalities of FT-DRB and NAC have been implemented using this tool, taking as starting point some previous *InfiniBand-based* models for adaptive routing [53], [54]. Processing nodes are explained in subsection 5.2.1, and network nodes (routers) in subsection 5.2.2.

Simulation Management tool

In order to reduce the overall time of the hundreds of simulations conducted during the evaluation of our proposals, we have designed and implemented a simulation management tool [110], [68] acting as an interface between OPNET Modeler and non-dedicated computer clusters. This tools has allowed us to increase the simulation capacity of the modeling tool exponentially by taking full advantage of available computing systems, including an efficient use of multicore processing nodes.

5.2.1 Processing Nodes

The implemented processing node, shown in Fig. 5.1, includes a processor node that simulates the communication pattern of an application or a synthetic workload, and a network interface to connect the processing node to a network node. The processor consists of a packet source (*src*) and a packet consumer (*sink*). The source module (*src*) generates link layer data packets according to a specific synthetic traffic pattern, a probability density function, or an application trace. This module is shown in Fig. 5.2. Additionally, processing nodes are provided with several attributes related to packet generation such as: injection rate, start and stop time, packet format and length, workload characterization, etc. The consumer module (*sink)*, shown in Fig. 5.3, analyzes received packets in order to update traffic statistics and performance metrics (packet latency, network throughput, etc.). The FSM of the input buffer is shown in Fig. 5.4. This module is responsible for receiving incoming packets from network nodes and buffering them while waiting to be processed by endnodes. The flow control unit, shown in Fig. 5.5, is responsible for sending and receiving flow control packets, including NAC-based identification control packets. On the other hand, the CCA module shown in Fig. 5.6 includes functionalities for processing the FT-DRB ACK packets in order to increase or reduce the number of alternative paths.

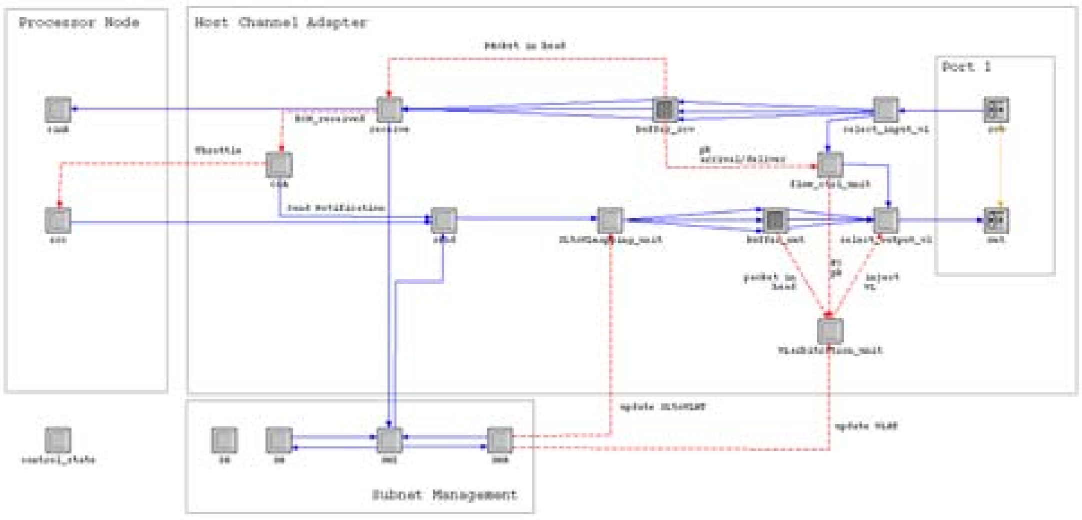


Figure 5.1: Processing node model implementation.

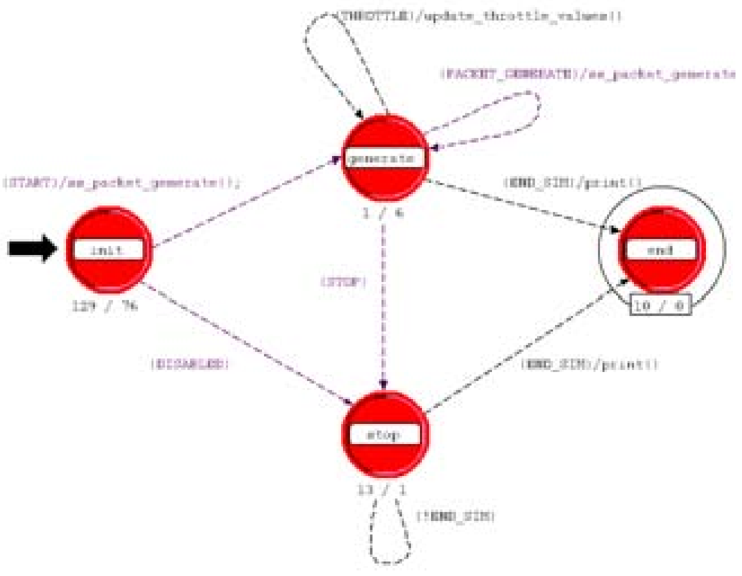


Figure 5.2: Processing node Source module FSM.

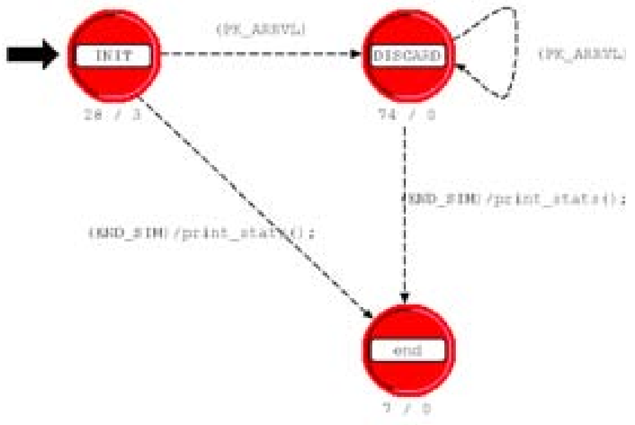


Figure 5.3: Processing node Sink module FSM.

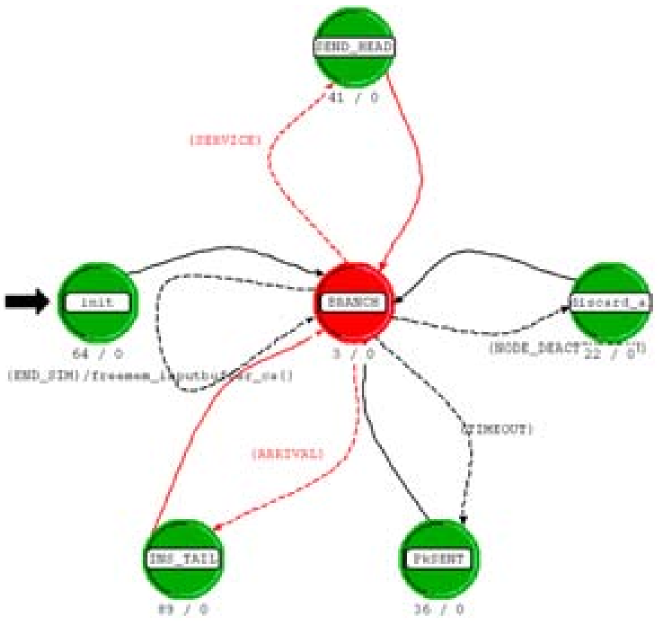


Figure 5.4: Processing node Input Buffer module FSM.

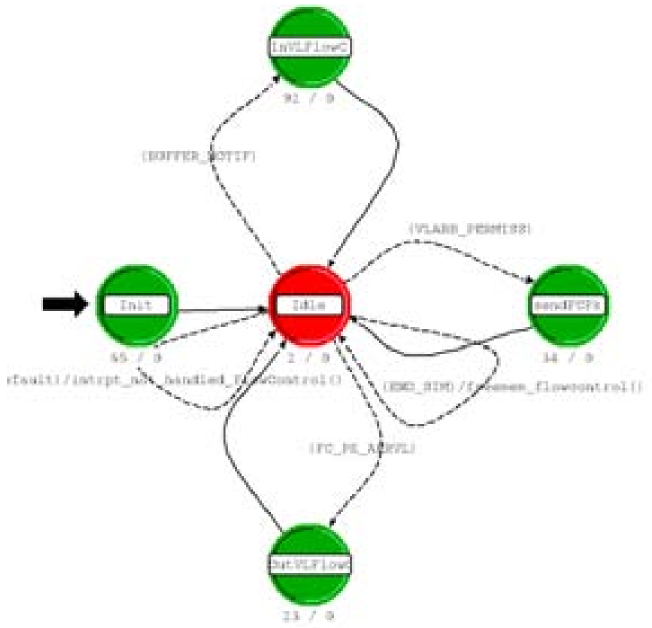


Figure 5.5: Processing node Flow Control module FSM.

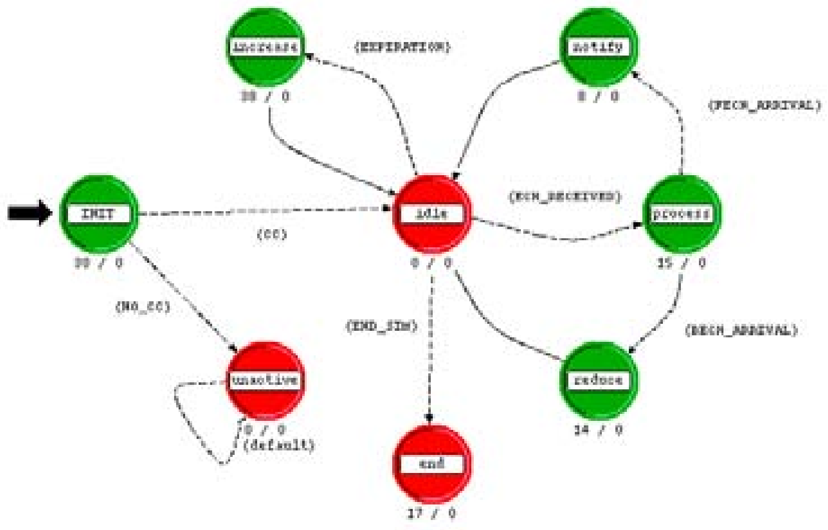


Figure 5.6: Processing node CCA module FSM.

5.2.2 Network Nodes

The internal structure of the implementation of the 8-ports network node is shown in Fig. 5.7. This model is based on the *InfiniBand Architecture specification* [39] and provides a set of modules that allow to experiment with several routing policies. The logical behavior of the router is given by four main modules: the subnet manager, the crossbar, the arbitration unit, and the routing unit.

The FSM of the Subnet Manager module is shown in Fig. 5.8. This module is responsible for the initialization and configuration of the entire network topology. The routing unit, shown in Fig. 5.9, assigns an output port to each incoming packet, and handles possible routing errors. The routing unit processes simultaneous requests applying a round-robin policy and can use both forwarding tables or algorithmic routing. Packets contending for a given output channel are stored in a structure and are attended in a sequential way. Basically, FT-DRB is implemented within this module. Once the routing process has assigned an out port to a packet, the arbitration unit sends a signal to the crossbar in order to set up the corresponding connection between input and output ports. Similarly to processing nodes, network nodes also includes input and output buffers and the corresponding flow control unit. The functionality of these modules is almost identical to previously explained FSM of Figs. 5.4 and 5.5.

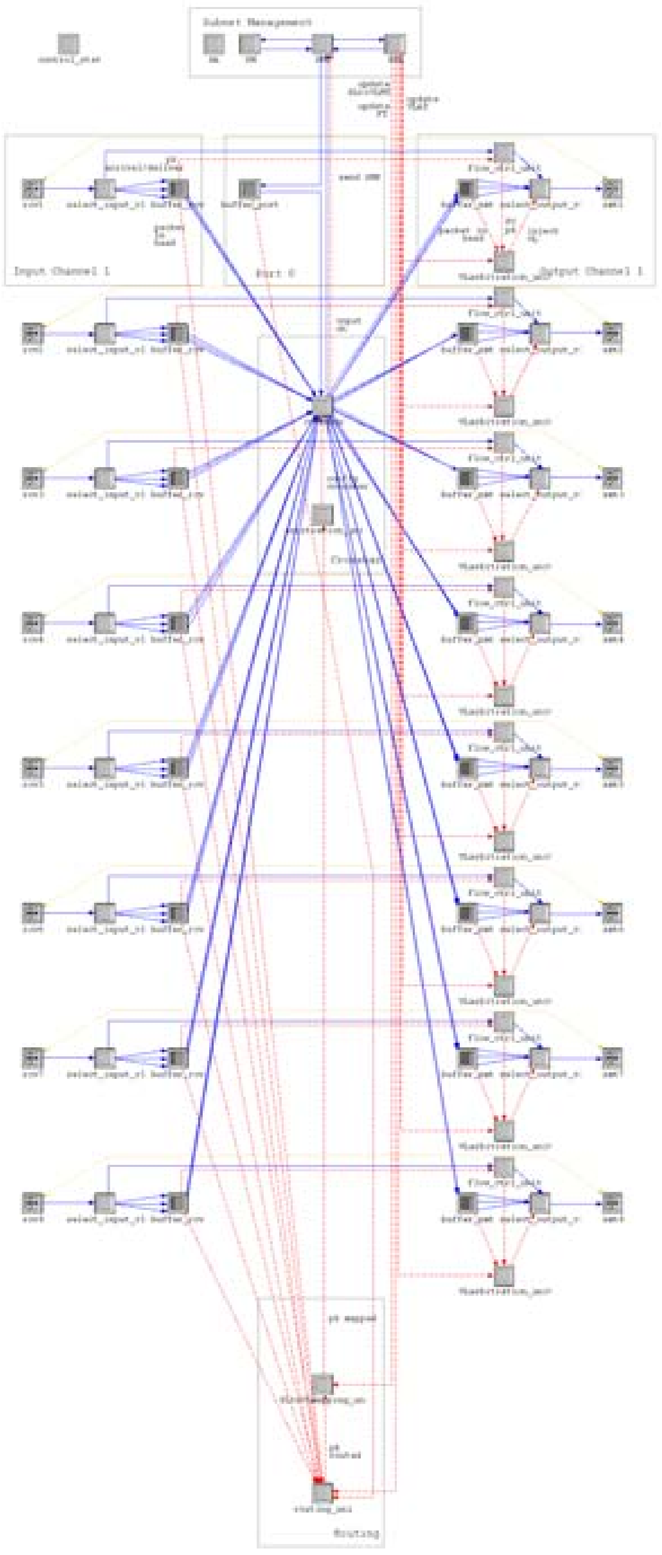


Figure 5.7: Network node model implementation.

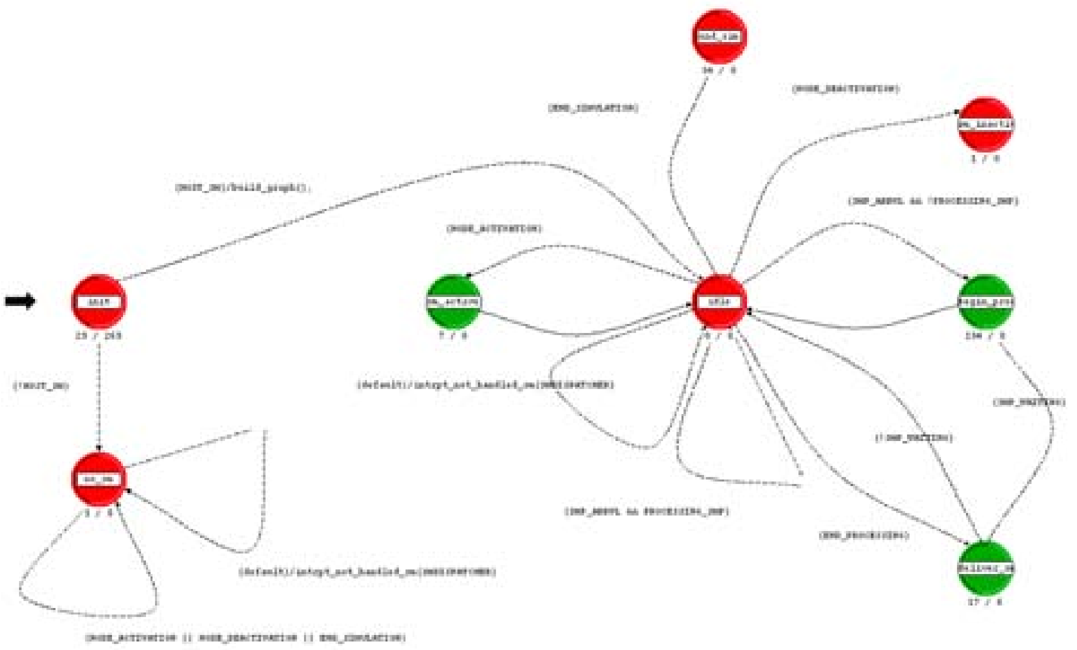


Figure 5.8: Network node Subnet Manager module FSM.

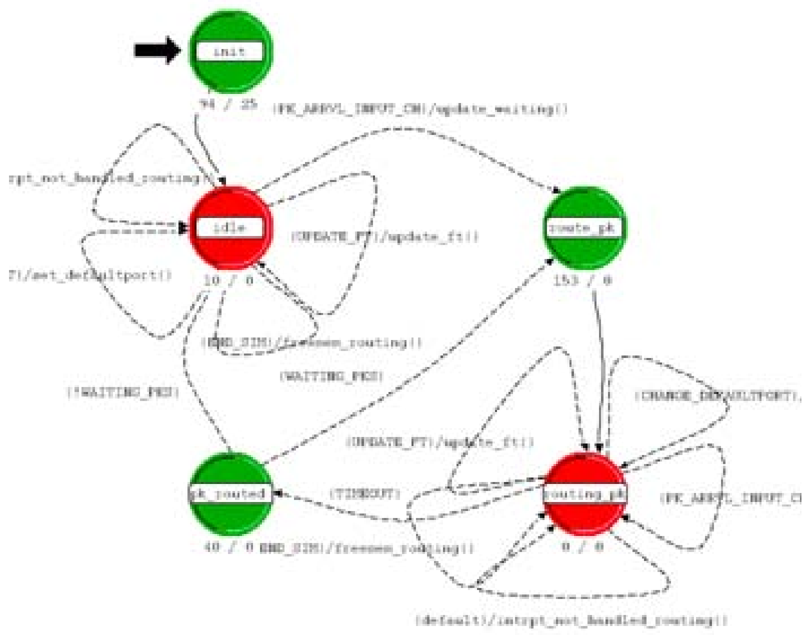


Figure 5.9: Network node Routing module FSM.

5.3 Evaluation Metrics

In order to observe the performance degradation of the network in the presence of link failures, we have measured the average latency of communications in the interconnection network. Latency is the time required for a packet to traverse the network, from the time the head of a packet arrives at the input port to the time the tail of the packet departs the output port [14, Ch. 3]. The average latency for each packet x reaching a destination node i is given in Eq. 5.1, where $l_i[x]$ is the latency value of the packet x at the node i.

$$\overline{L}_i[x] = \frac{1}{x}(l_i[x] + (x-1) * \overline{L}_i[x-1]), \quad \forall x \neq 0 \tag{5.1}$$

The global average latency is calculated by averaging the latencies of every packet, and is measured in seconds as defined in Eq. 5.2, where n is the number of destination nodes.

$$\overline{L} = \frac{1}{n} \sum_{i=1}^{n} \overline{L}_i \tag{5.2}$$

We have measured *throughput* as the data rate in bits per seconds that the network accepts per input port. To this end, we have taken into account the ratio between the number of packets received at destination nodes (accepted load) and the number of packets injected at source nodes (offered load).

Additionally, we have performed measurements on the number of sent and received packets after each simulation run. This extra control is intended to confirm the correct operation of both proposals, ensuring that there are no packet drops during the simulation process and the network has been able to function in the presence of faults.

5.4 Evaluation Method

The evaluation of proposals is aimed to confirm the correct operation of both FT-DRB and NAC as they were described in previous chapters. To this end, we have first evaluated the *Non-blocking Adaptive Cycles* without including –disabling– the fault-tolerant functionalities of FT-DRB. This first step is intended to prevent that the multipath distribution characteristics of FT-DRB affect the measurement of the identification and recovery times of NAC. The NAC evaluation method is further described in subsection 5.4.1. The second part of the evaluation is focused on testing the functionalities of the *Fault-tolerant Distributed Routing Balancing* method. Unlike the first step, the evaluation of FT-DRB includes the functionalities of both proposals, FT-DRB and NAC (for deadlock avoidance). The FT-DRB evaluation method is further described in subsection 5.4.2.

To ensure the statistical validity of discrete event simulations, it was necessary to execute multiple instances of the simulation with varying random number seeds. As a rule of thumb, two to three tens runs of a simulation with different random number seeds should provide enough data to build a useful confidence interval around your simulation results [40]. Therefore, to consider such statistical anomalies, the simulation model should be evaluated several times using different seed sequences; and the results obtained in the individual simulations should be combined in some way (e.g. through their average values) in order to estimate a typical behavior of the system. This avoids erroneous and inaccurate analysis of the results and allows a greater degree of confidence.

The experiments presented along this chapter have been conducted taking into account the above mentioned aspects for statistical validity. More precisely, we have used the methodology proposed in [69]. Evaluation results and test scenarios of both approaches are explained in detail in sections 5.5 (NAC) and 5.6 (FT-DRB).

5.4.1 NAC Evaluation

The evaluation of the proposed deadlock avoidance technique has been conducted according to an incremental three-steps approach. In this evaluation, we have configured several test scenarios that lead to deadlocked configurations in the absence of deadlock avoidance mechanisms. Basically, these scenarios consists of a 8x8 torus topology with limited buffer size in networks nodes and a *tornado* synthetic communication pattern in some nodes. The summary of the simulation parameters used for configuring these scenarios is given in Table 5.4.

Network Parameters	Value
Network topologies	Torus 8x8
Link bandwidth	2 Gbps
Buffer size	4 KBytes
Packet payloads	2 KBytes
Packet generation rate	400 packets/sec/node 500 packets/sec/node 600 packets/sec/node
Length of routing cycle	8 nodes (one ring of the torus)
Traffic pattern in the cycle	Tornado
Traffic patterns in other nodes	Bit Reversal, Butterfly, Perfect Shuffle, Matrix Transpose

Table 5.4: Simulation parameters used in the evaluation of NAC.

In the first step of the evaluation, we have studied and measured the response of NAC in one of the deadlock-prone scenarios explained above. For this step, we have included the *tornado* traffic in the nodes of the routing cycle and no traffic in the rest of network nodes. An example of this kind of scenario is shown in Fig. 5.10. This kind of 3D figures are known as *surface-maps*, where axis x and y represent the coordinate of a node, and the z axis represents some specific values in that node, such as the number of received packets in our example. We will use *surface-maps* with different values in the z axis to ease the visualization of the evaluation results in section 5.5.

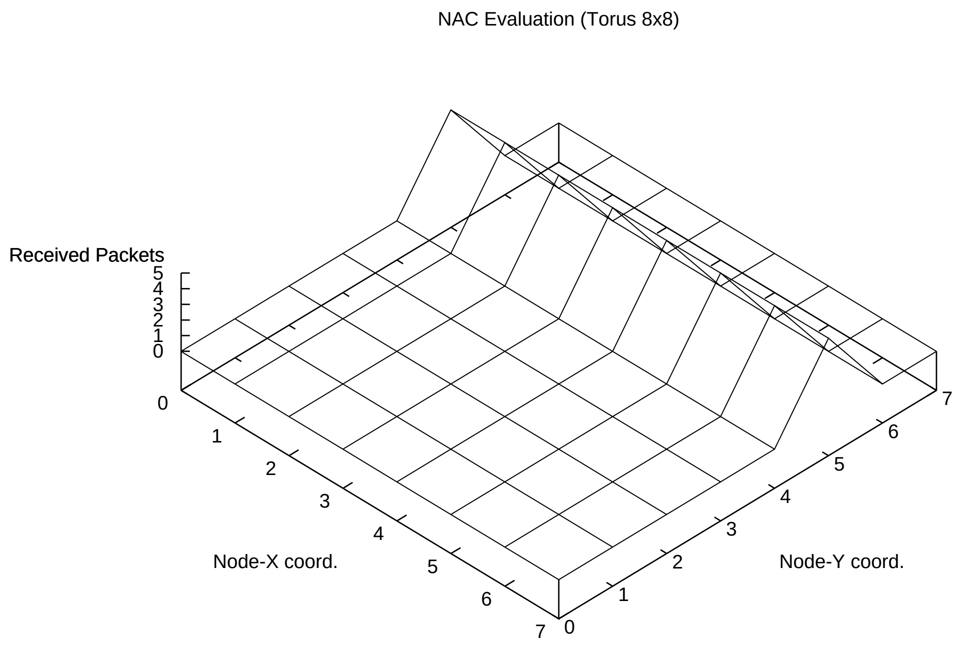


Figure 5.10: Surface-map for tornado traffic in a 8x8 torus (row y = 5).

For the second step of the NAC evaluation, we have varied the packet generation rate of the tornado pattern. This step is aimed on evaluating the differences in average latency and throughput values during the operation of NAC. Finally, in the third step we have included the four additional traffic patterns listed in Table 5.4 to the rest of network nodes. Then, we have repeated the evaluation of latency and throughput differences.

5.4.2 FT-DRB Evaluation

The evaluation of FT-DRB was conducted in two steps. First, each scenario was simulated thirty times with no link failures. Later, link failures were injected in the scenarios used in the first step (for each approach). Finally, performance degradation was measured as the difference between latency values obtained from the faulty and fault-free scenarios. As the aim of these experiments is to evaluate the functionality of the method (and congestion problems caused by the occurrence of failures), the simulations were conducted using moderated traffic loads.

Experimentation is based on torus and fat-tree network topologies chosen mainly due to their multiple alternative paths between nodes and their current popularity. The network was modeled based on interconnection elements and endnodes previously described in section 5.2, that provide the interface to connect processing nodes to the network through links. Simulations were conducted for three different torus topologies and one fat-tree, considering several standard packet sizes and a constant packet injection rate. Link bandwidth was set to 2 Gbps, and the buffers of network nodes to 8 KB. A more detailed description of simulation parameters used throughout the evaluation of FT-DRB is presented in Tables 5.5 and 5.6. The specific simulations parameters used on each evaluation are summarized within the corresponding results section.

In order to provide a complete evaluation of the fault-tolerant routing method, we have defined an incremental evaluation approach comprising the following steps:

1. Performance evaluation for *permanent* link failures. Three different groups of evaluations have been conducted:
   (a) Performance when dealing with synthetic traffic patterns.
   (b) Performance when facing spatial fault patterns.
   (c) Performance when applying some collective communication libraries widely used in HPC systems.
2. Performance evaluation for *transient* link failures using synthetic traffic patterns.
3. Performance evaluation for *variable-duration* link failures based on the information of availability traces and using synthetic traffic patterns.
4. Brief evaluation of the number of alternative paths in the performance of permanent link failures using synthetic traffic patterns.

The entire set of evaluation results is detailed in section 5.6; permanent failures in subsections 5.6.1, 5.6.2, and 5.6.3; transient failures in subsection 5.6.4; variable duration failures in subsection 5.6.5; and the evaluation of alternative paths in subsection 5.6.6.

Network Parameters	Value
Network topologies	Torus 8x8, Torus 16x16, Torus 32x32, Fat-tree 4-ary 3-tree
Link bandwidth	2 Gbps
Buffer size	8 KBytes
Packet payloads	256 Bytes, 512 Bytes
Packet generation rate	400 packets/sec/node
Traffic patterns	Uniform Permutation, Bit Reversal, Butterfly, Perfect Shuffle, Matrix Transpose, Complement, One-to-all (Scatter), All-to-One (Gather)

Table 5.5: Simulation parameters used in the evaluation of FT-DRB (part I).

FT-DRB Parameters	Value
Number of link failures	0, 2, 4, 6, 8, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 10% of links in torus 8x8: 12 10% of links in torus 16x16: 51 10% of links in fat-tree 4-ary 3-tree: 12
Start time of link failures	0, 50%, Random
Duration of link failures	Real-based, Permanent 100%, Permanent 50%, Permanent random, Transient random
Number of alternative paths	1, 2, 3, 4, 5
Availability traces	LANL 12, LANL 18, LANL 19, PNLL MPP2
Spatial fault regions	Convex: line, square Concave: L, U

Table 5.6: Simulation parameters used in the evaluation of FT-DRB (part II).

5.5 NAC Evaluation Results

This section summarizes the evaluation results of the *Non-blocking Adaptive Cycles* technique. Results of the evaluation of NAC in a deadlock-prone scenario with *tornado* traffic is detailed in subsection 5.5.1. Then, results obtained applying different traffic loads to the *tornado* traffic are presented in subsection 5.5.2. The evaluation including additional traffic patterns in the rest of network nodes is then presented in subsection 5.5.3.

5.5.1 Tornado Pattern

In this section, we study the response of NAC to the deadlock-prone test scenario explained previously in subsection 5.4.1. This scenario consists on a 8x8 torus topology with limited buffer size (4 KB) in networks nodes and a *tornado* communication pattern in one row of the torus topology. All the processing nodes of this row have been configured with a generation rate of 500 [pk/node/sec]. In order to study temporal variations in the working conditions of the network, we have collected several data values at different time slots of 0.01 seconds throughout simulations. For each 2-seconds simulation scenario we have $\frac{2\ [s]}{0.01\ [s]} = 200$ time slots, where the first slot corresponds to the time interval ranging from 0.00 to 0.01 [s]. A *surface-map* showing the number of received packets in the second time slot of the previously explained scenario is shown in Fig. 5.11.

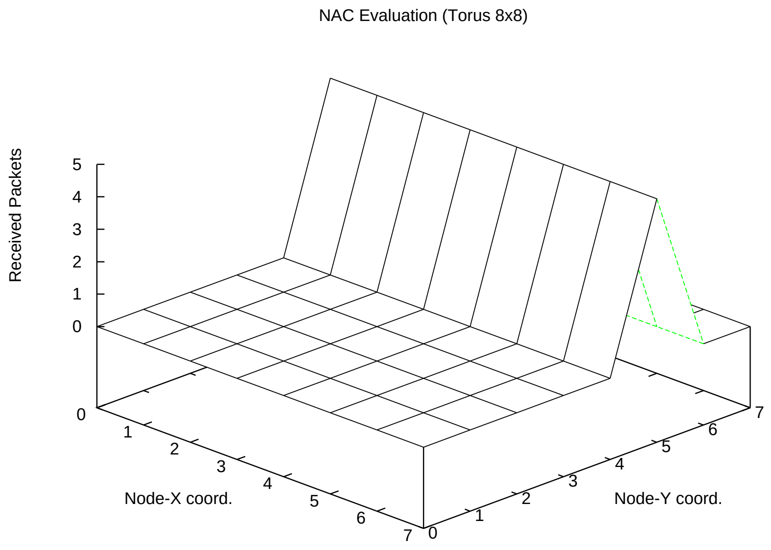


Figure 5.11: Packets received in the time slot 0.01-0.02 [s].

In the absence of mechanisms for providing deadlock freedom to the network, the above test scenario reaches a deadlock configuration and remains blocked until the end of the simulation, as shown in Fig. 5.12. When applying NAC for providing deadlock avoidance, the system does not reach a deadlock configuration and is able to continue operating until the end of the simulation. This situation can be observed in the average latency diagram of Fig. 5.5.1 where the deadlock-prone situation arises at around 1.18 [s]. The most important aspect to be inferred from this figure is that NAC is able to successfully avoid the deadlock occurrence thus allowing the system to remain in operation. In these scenarios where the routing cycle is composed by 8 nodes, the total average NAC time for treating deadlocks is about 6.36 [μs] and the difference –degradation– between the latency values of the points before and immediately after the deadlock prone situation is about 10.22 [μs]. The same analysis can be done with the throughput values shown in Fig. 5.5.1, where the degradation is about 0.25%. A detailed view of the throughput values during the treatment of the deadlock-prone situation is shown in Fig. 5.15.

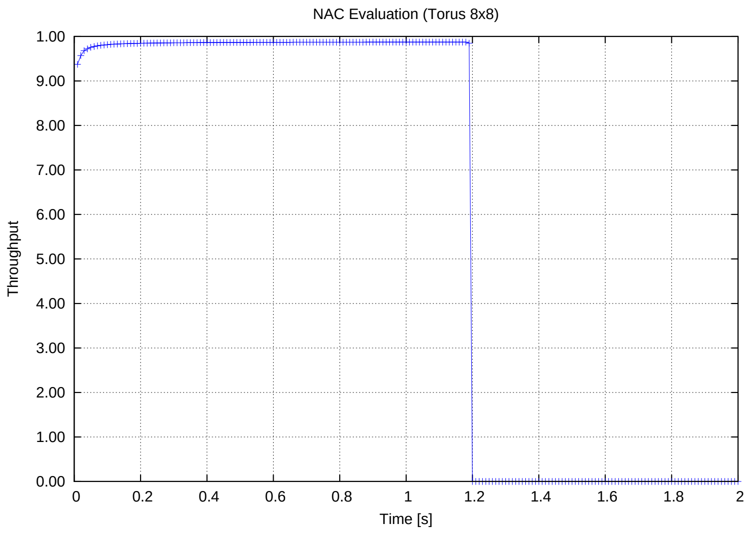


Figure 5.12: Throughput throughout simulation time in a deadlock configuration. Tornado traffic pattern with generation rate = 500 [pk/node/sec].

The *surface-map* showing the number of received packets in the time slot where NAC treats the deadlock-prone situation is shown in Fig. 5.16. Is is possible to observe in these figures that the network node at (x: 3; y: 5) presents a lower number of received packets, that is precisely the node that detects the NAC triggering conditions (explained in subsection 4.2.1) and starts the detection of the routing cycle. A complete sequence including four slots ranging from 1.17 to 1.21 [s] is shown in Fig. 5.17. This sequence of time slots shows four key points: the time period before the deadlock-prone situation (slot 1.17-1.18); the time period where NAC is applied (slot 1.18-1.19); the time period following the treatment of the deadlock-prone situation where the firstly blocked node received it's blocked packets (slot 1.19-1.20); and the time period when the system has recovered it's normal functioning conditions (slot 1.20-1.21).

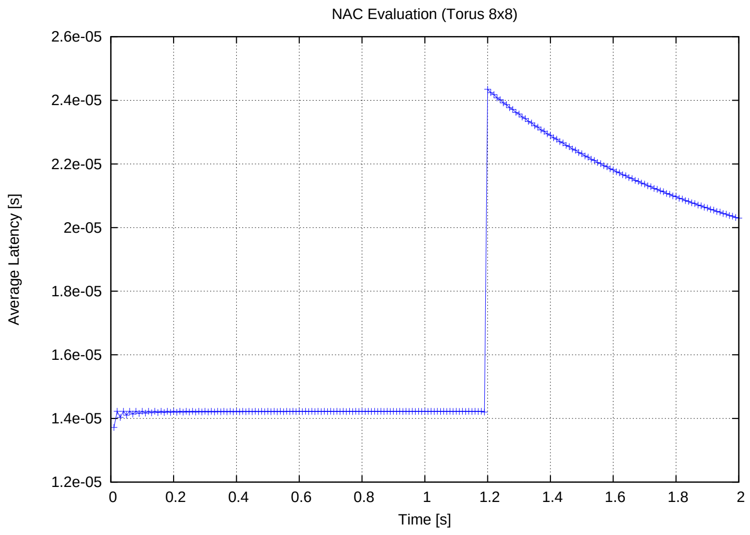


Figure 5.13: Average latency throughout simulation time applying NAC. Tornado traffic pattern with generation rate = 500 [pk/node/sec].

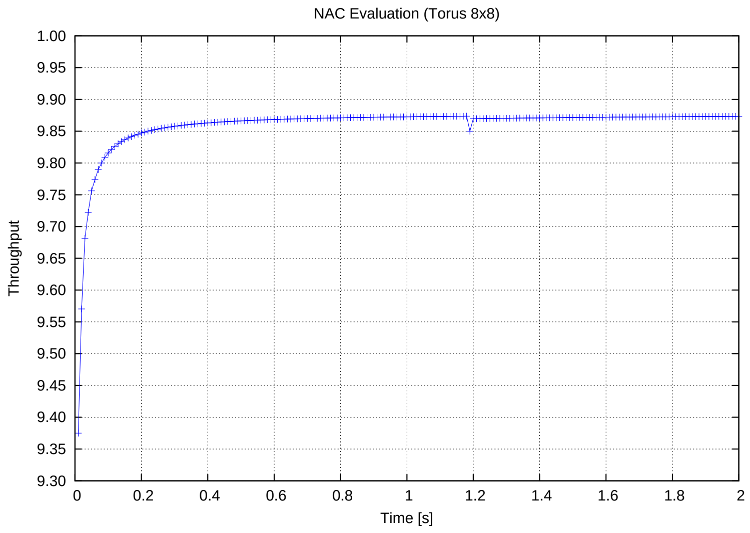


Figure 5.14: Throughput throughout simulation time applying NAC. Tornado traffic pattern with generation rate = 500 [pk/node/sec].

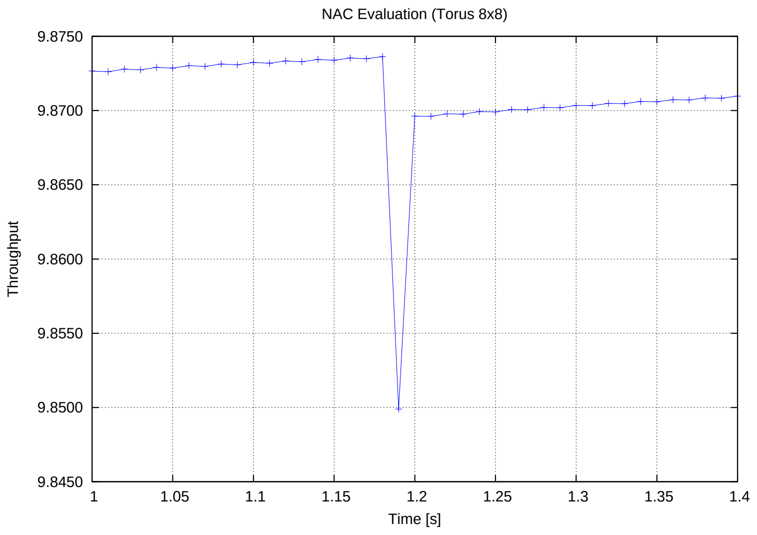


Figure 5.15: Detailed view of throughput values in the time interval 1.00-1.40 [s]. Tornado traffic pattern with generation rate = 500 [pk/node/sec].

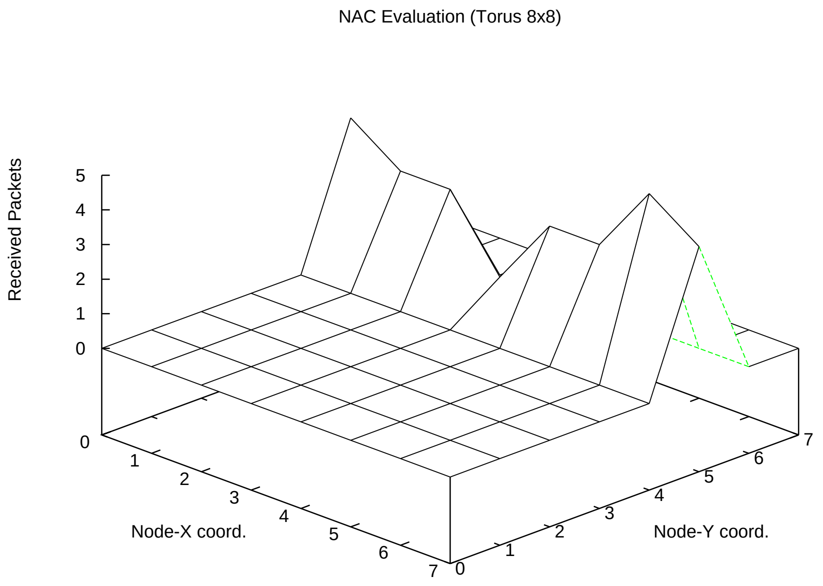
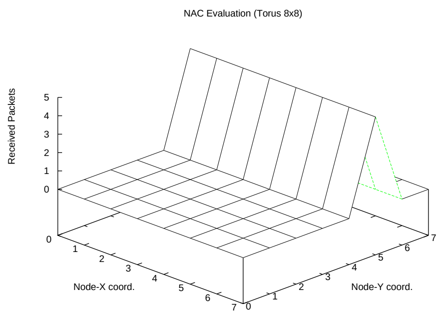
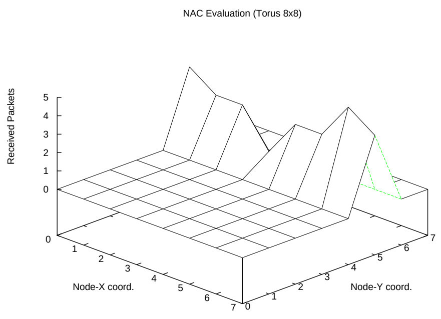


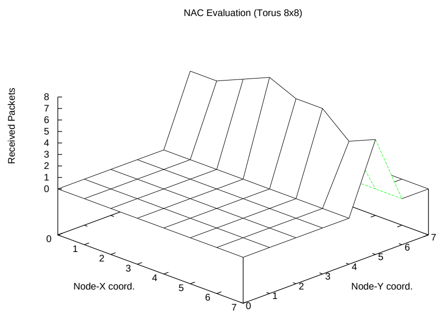
Figure 5.16: Packets received in the time slot 1.18-1.19 [s]. Tornado traffic pattern with generation rate = 500 [pk/node/sec]. Within this time slot, NAC treats a deadlock situation in the network node (x: 3, y: 5).



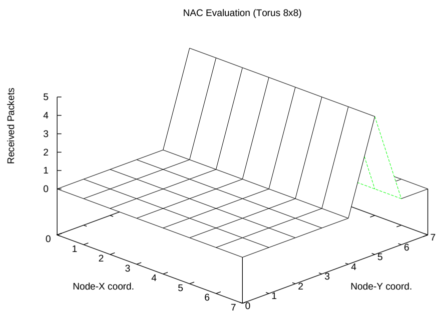
(a) Time slot 1.17 to 1.18



(b) Time slot 1.18 to 1.19



(c) Time slot 1.19 to 1.20



(d) Time slot 1.20 to 1.21

Figure 5.17: Packets received in four time slots ranging from 1.17 to 1.21 [s]. Tornado traffic pattern with generation rate = 500 [pk/node/sec]. Slot 1.17-1.18 corresponds to the time period before the deadlock situation. Slot 1.18-1.19 to the treatment of the deadlock situation with NAC. Slot 1.19-1.20 to the time period after the treatment of the deadlock situation (gradual recovery of normal functioning). And in the slot 1.20-1.21 the system has recovered it's normal functioning conditions.

5.5.2 Tornado Pattern with Variable Load

As a second step of the NAC evaluation, we have varied the packet generation rate of the network nodes in the *tornado* pattern row of the 8x8 torus. The purpose of this modification is to evaluate the differences in average latency and throughput values when NAC is treating the deadlock-prone situation. For this evaluation, we have considered three different generation rates: 400, 500 and 600 [pk/node/sec]. Some complementary set of results of this section, including *surface-maps* of received packets per each generation rate have been included in Appendix A, subsection A.1.1.

The absolute values and percentages of latency degradation[1] are shown in Figs. 5.18 and 5.19, respectively. The degradation of applying a generation rate of 500 [pk/node/sec] is about 12% higher than the degradation obtained whit 400 [pk/node/sec]. Similarly, the generation rate of 600 [pk/node/sec] generates an increment of about 8% in latency. It is worth noting that a 50% of increase in latency values is not a high cost when dealing with potential deadlock situations. The absolute latency values are shown in Fig. A.1 (Appendix A). A similar analysis can be applied to the throughput degradation values presented in Fig. 5.20, where the maximum degradation value of this evaluation is about 0.33%.

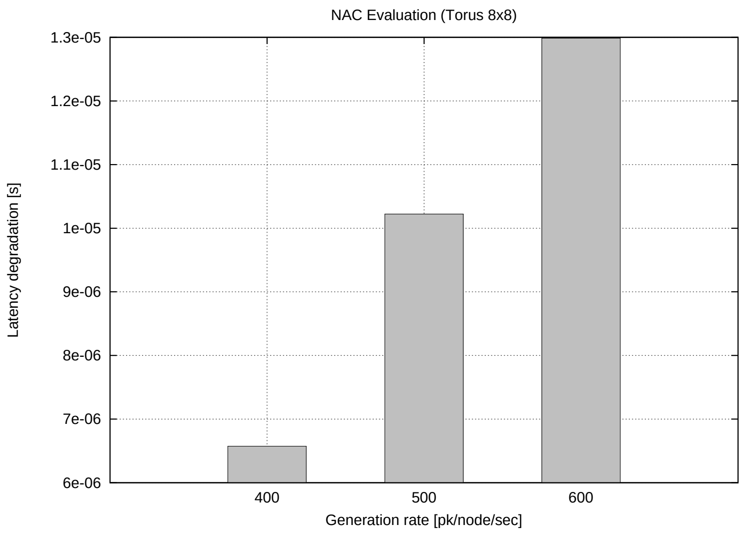


Figure 5.18: Difference between absolute latency values before and after treating deadlocks.

[1]Measured as the difference between the latency values of the points before and immediately after the deadlock prone-situation.

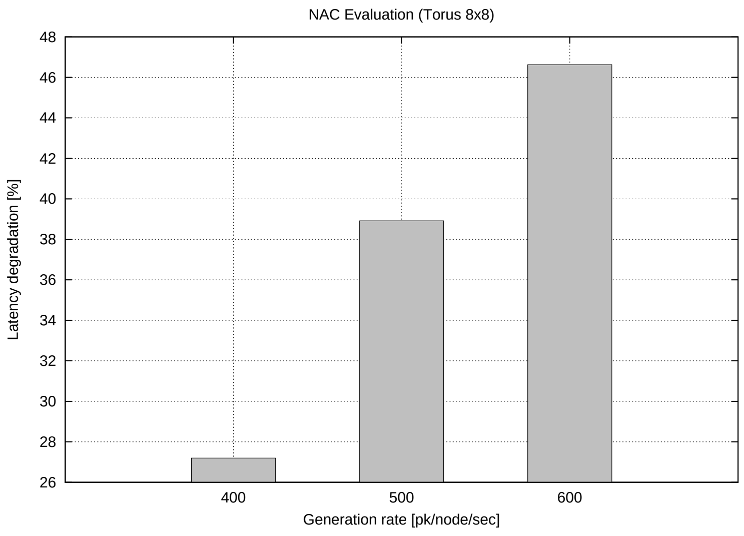


Figure 5.19: Difference between latency values [%] before and after treating deadlocks.

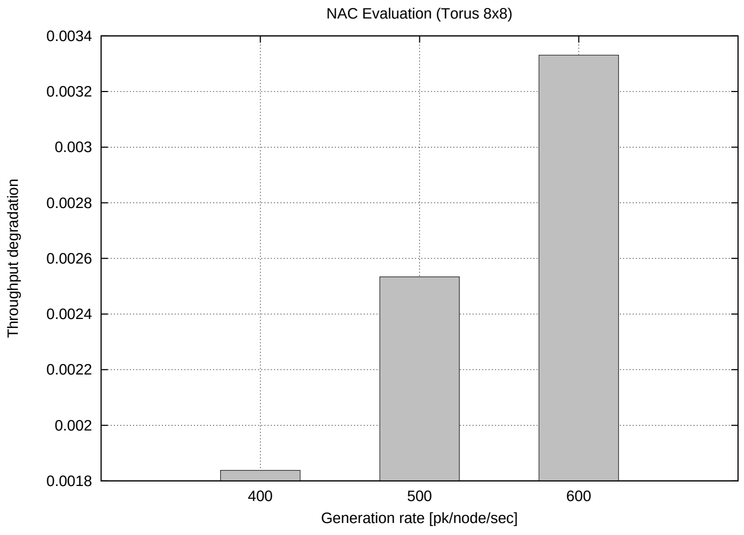


Figure 5.20: Difference between throughput values before and after treating deadlocks.

5.5.3 Tornado Pattern with Additional Traffic

The final step of the NAC evaluation consists on extending the previous evaluation by including additional synthetic communication patterns to the rest of nodes of the 8x8 torus. Four additional patterns have been applied separately with a generation rate of 200 [pk/node/sec]. These patterns are: *Bit reversal*, *Butterfly*, *Matrix transpose*, and *Perfect shuffle*.

The absolute values and percentages of latency degradation for each test scenario are presented in Figs. 5.21 and 5.22, respectively. It is possible to observe from these figures that the presence of additional communication patterns have considerable impacts for low generation rates, as could be inferred from the differences between the values when applying *Bit reversal* and *Matrix transpose* patterns. This gap in latency values is considerably smaller when increasing the generation rate, as in the case of 600 [pk/node/sec]. The throughput values of Fig. 5.23 present a similar behavior. The absolute latency values for each test scenario are shown in Fig. A.5 (Appendix A).

The sequence of *surface-maps* of received packets for the four key time slots is shown in Fig. 5.24, namely: the time interval before the deadlock situation; the time interval during the NAC treatment; the time interval after the NAC treatment; and finally the time interval after the deadlock situation. The complete set of *surface-maps* of received packets per each generation rate and traffic pattern has been included in Appendix A, subsection A.1.2.

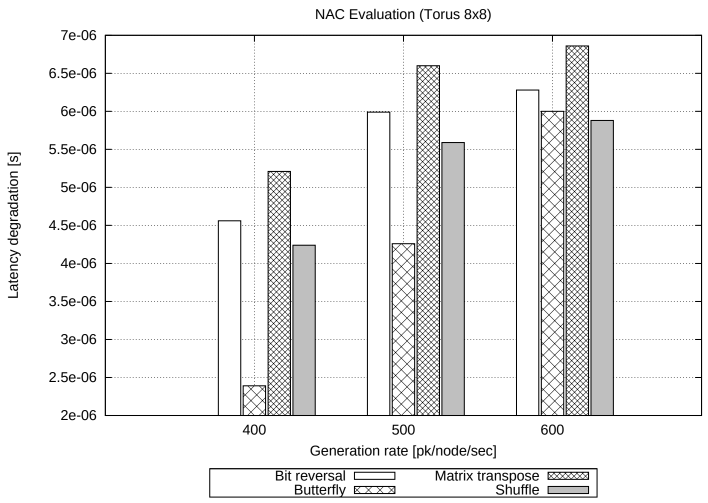


Figure 5.21: Difference between absolute latency values before and after treating deadlocks with additional traffic patterns.

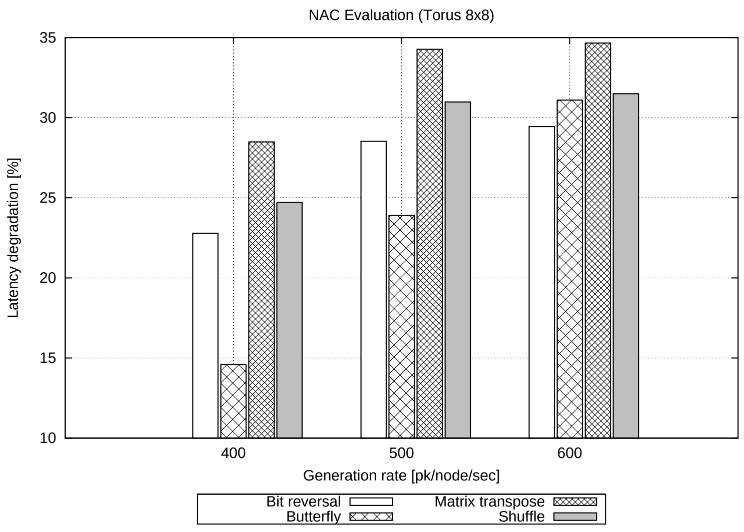


Figure 5.22: Difference between latency values [%] before and after treating deadlocks with additional traffic patterns.

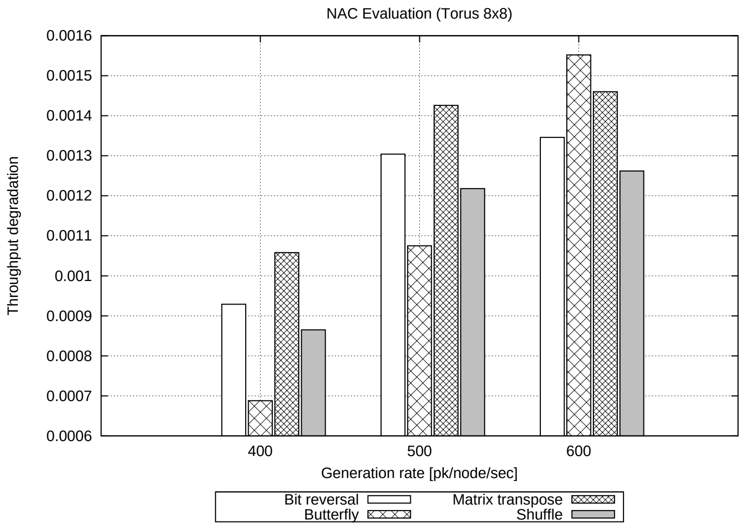
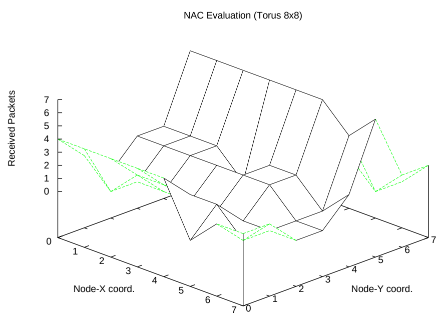
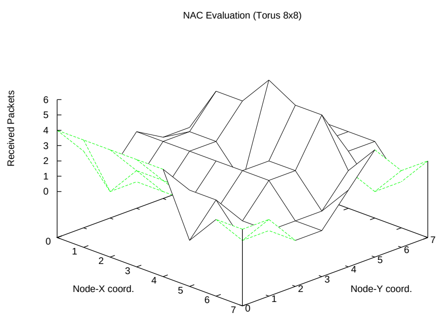


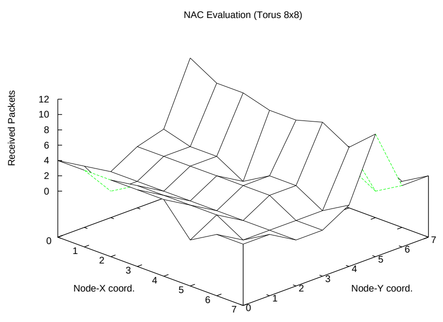
Figure 5.23: Difference between throughput values before and after treating deadlocks with additional traffic patterns.



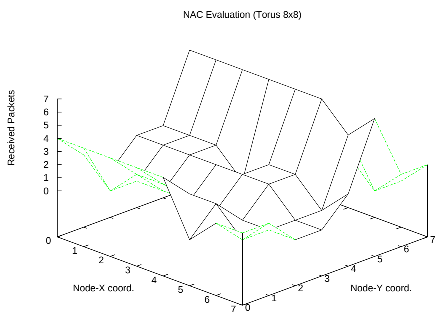
(a) Before deadlock



(b) During NAC treatment



(c) After NAC treatment



(d) After deadlock

Figure 5.24: Packets received in four time slots. Tornado traffic pattern with generation rate = 500 [pks/node/sec]. Additional bit reversal traffic pattern with generation rate = 200 [pks/node/sec]. Slot 5.24(a) corresponds to the time period before the deadlock situation. Slot 5.24(b) to the treatment of the deadlock situation with NAC. Slot 5.24(c) to the time period after the treatment of the deadlock situation (gradual recovery of normal functioning). And in the slot 5.24(d) the system has recovered it's normal functioning conditions.

5.6 FT-DRB Evaluation Results

This section summarizes the evaluation results of the four incremental steps proposed in the previous section for FT-DRB. The performance evaluation of permanent link failures comprises three major groups of test scenarios: evaluation when dealing with synthetic traffic patterns (subsection 5.6.1); evaluation when facing spatial fault patterns (subsection 5.6.2); and finally the evaluation with two collective communication patterns (subsection 5.6.3). Then, the evaluation of transient link failures is presented in subsection 5.6.4. Real-based evaluation is summarized in third place in subsection 5.6.5. The evaluation of the number of alternative paths is briefly presented at subsection 5.6.6. Finally, a brief discussion about the evaluation of FT-DRB is outlined at the end of the section.

5.6.1 Permanent Failures with Synthetic Traffic

This evaluation is aimed to study how permanent failures affect synthetic traffic patterns commonly applied in computational intensive scientific applications [22, Ch. 9]. The simulation parameters used in this evaluation is summarized in Tables 5.7 and 5.8.

The evaluation results of the aforementioned traffic patterns are grouped according to the network topology used in the evaluation. Additional results for permanent failures with random start times have been included in subsection A.2.1 of Appendix A.

There are two important points to emphasize about these results. First, the high performance levels achieved by FT-DRB, where the lower performance value, failing 10% of network links is about 80%, as shown in Fig. 5.25. Second, the fact that performance losses are gradual, avoiding abrupt transitions in performance and thus in the average latency values of high-speed interconnection networks. These high performance values are based on the use of alternative paths and an effective and successful distribution of communication load through those paths. There were no congestion problems in the fault-free scenarios, therefore no additional paths were necessary between each source-destination pair. However, the occurrence of faults reduces the bisection width of the network while increasing the utilization of other links to circumvent faults, thus generating congestion problems. For this reason, multipath were automatically established between some source-destination pairs to treat the congestion problem in the faulty scenarios.

The results for 2, 4, 6, 8 and 10 link failures in the 8x8 torus topology are shown in Fig. 5.26; results of the 16x16 torus topology in Fig. 5.27; and results of the Fat-tree 4-ary 3-tree in Fig. 5.28. The lower but still very promising performance value of these results corresponds to the *Uniform* pattern in the Fat-tree 4-ary 3-tree topology (Fig. 5.28). For the rest of traffic patterns, performance is above 95% in all three evaluated topologies.

Network Parameters	Value
Network topologies	Torus 8x8, Torus 16x16, Fat-tree 4-ary 3-tree
Link bandwidth	2 Gbps
Buffer size	8 KBytes
Packet payloads	256 Bytes, 512 Bytes
Packet generation rate	400 packets/sec/node
Traffic patterns	Uniform Permutation, Bit Reversal, Butterfly, Perfect Shuffle, Matrix Transpose, Complement

Table 5.7: Simulation parameters for permanent failures and synthetic patterns (part I).

FT-DRB Parameters	Value
Number of link failures	0, 2, 4, 6, 8, 10, 10%
Start time of link failures	50%, Random
Duration of link failures	50%, Random
Number of alternative paths	4

Table 5.8: Simulation parameters for permanent failures and synthetic patterns (part II).

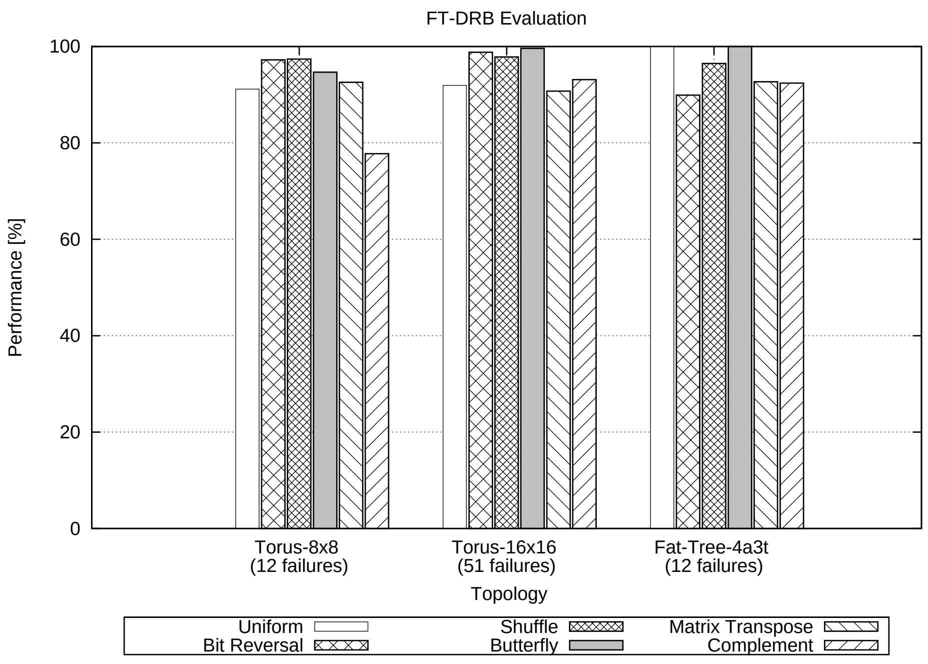
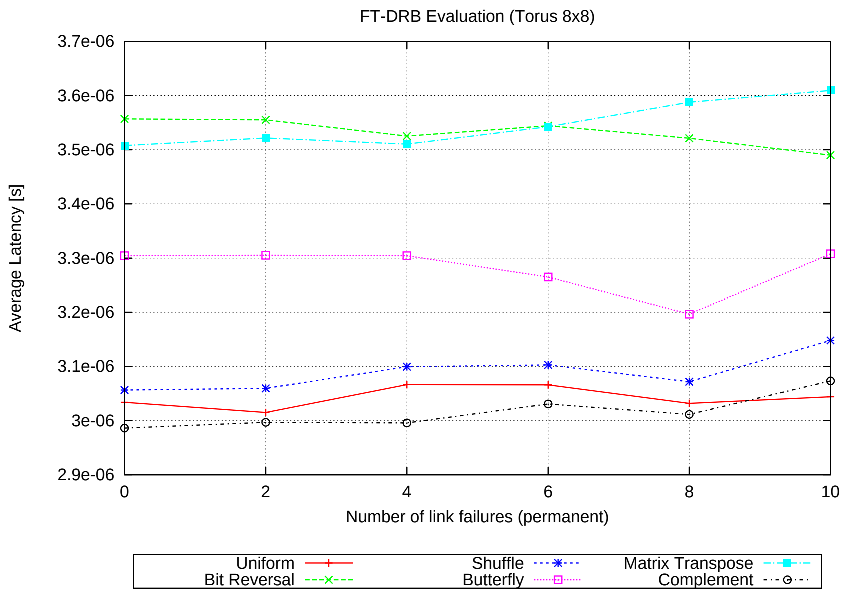
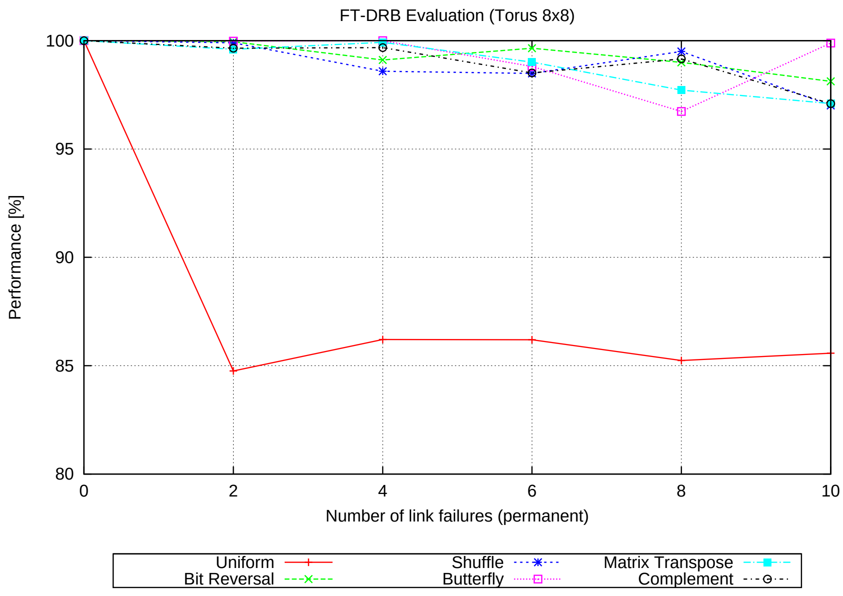


Figure 5.25: Evaluation results for 3 topologies with 10% of links failed (permanent).

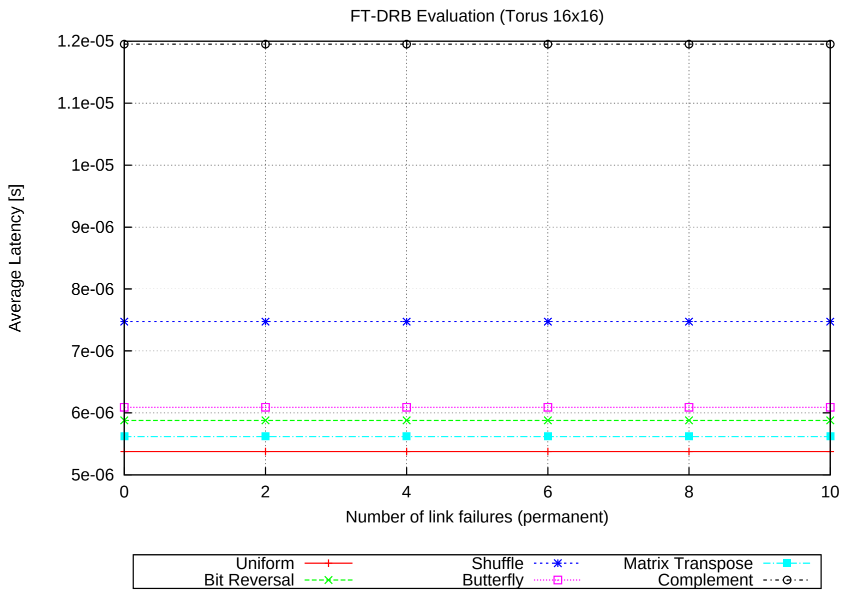


(a) Absolute values

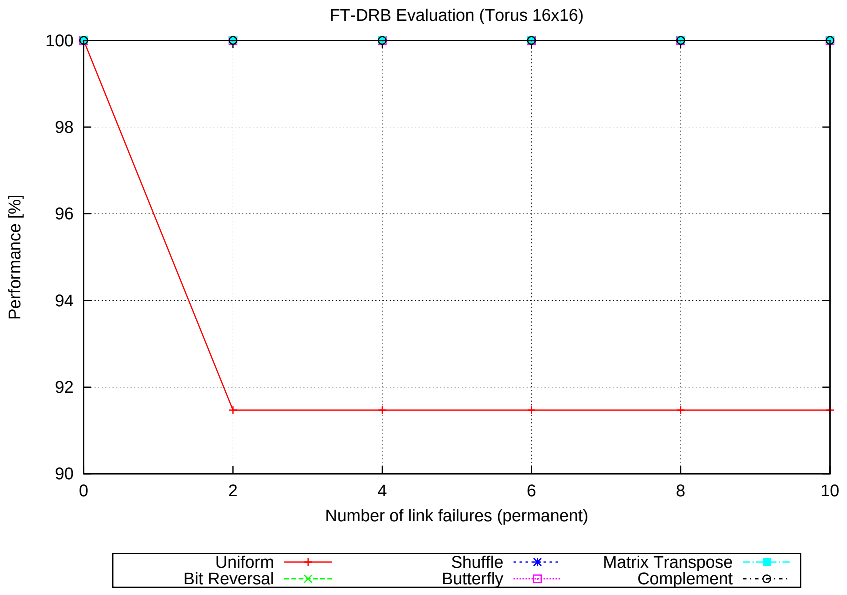


(b) Percentages

Figure 5.26: Evaluation results of permanent link failures in Torus 8x8.

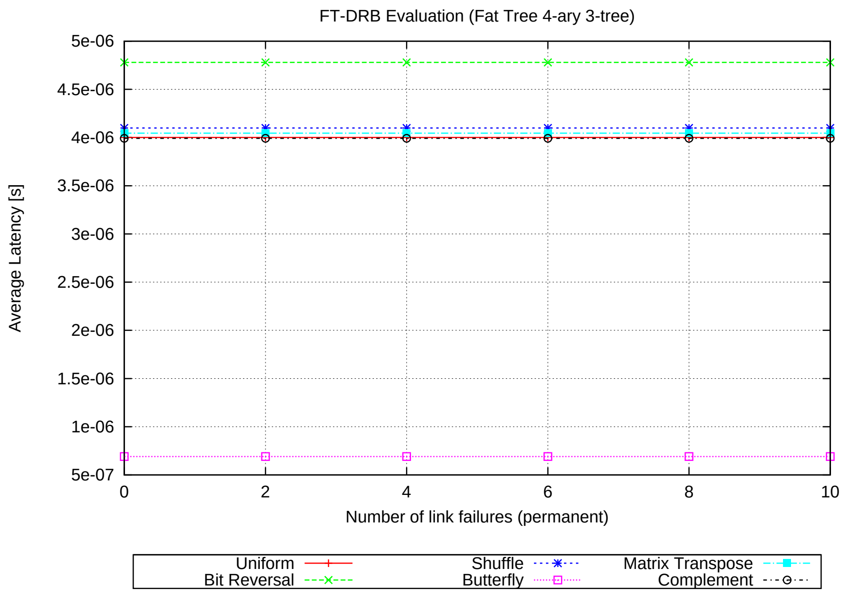


(a) Absolute values

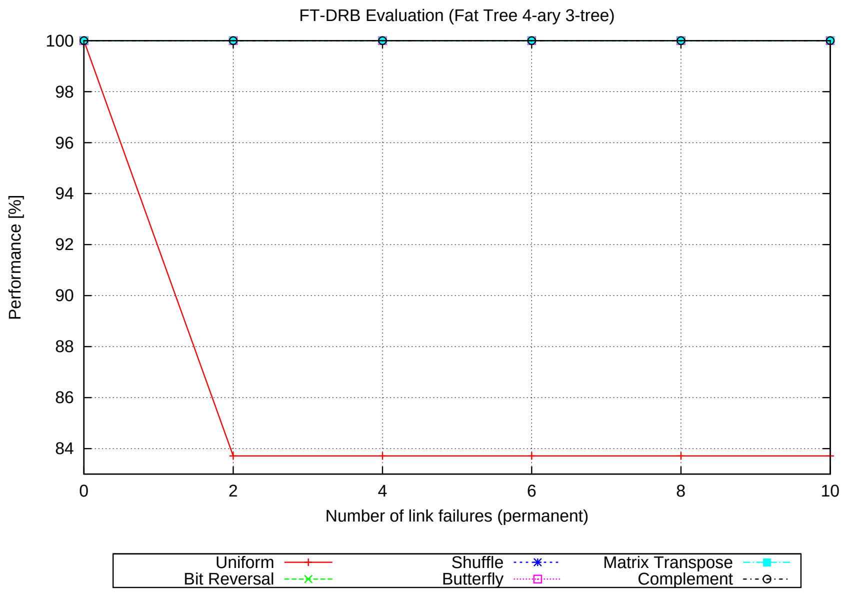


(b) Percentages

Figure 5.27: Evaluation results of permanent link failures in Torus 16x16.



(a) Absolute values



(b) Percentages

Figure 5.28: Evaluation results of permanent link failures in Fat-tree 4-ary 3-tree.

5.6.2 Permanent Failures with Spatial Fault Patterns

The evaluation of spatial fault patterns is intended to validate the behavior of the method when dealing with some common and complex fault combinations, whose occurrence probability was studied at [23]. This evaluations has been carried out using a 32x32 torus topology, considering failures in links instead of network nodes. In multidimensional k-ary n-cube networks, the groups of adjacent faults can be classified in two major categories: convex and concave. Convex regions may be further constrained to be regions whose shape is rectangular, i.e. line and square shapes. Moreover, concave regions present unique difficulties for fault-tolerant routing algorithms [22, Ch. 6]. Some concave regions are: L-shape, U-shape, etc. The set of simulation parameters used in this group of evaluation is summarized in Tables 5.9 and 5.10.

The test scenarios used to evaluate the spatial fault patterns are: *line* and *square* shapes (convex regions); *L* and *U* shapes (concave regions). Graphic examples of these spatial fault patterns are presented in Figs. 5.29(a), 5.29(b), 5.29(c), and 5.29(d), respectively.

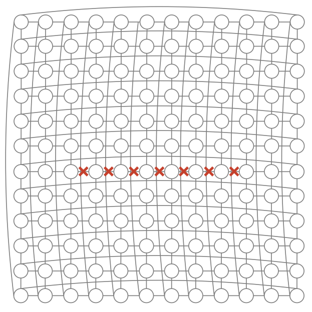
The average performance values and standard deviations for the communication patterns can be seen in Fig. 5.30. All test scenarios show average performance values above 98%. These are excellent results because spatial fault regions present complex and unique difficulties for fault-tolerant methods. This is particularly important when dealing with concave regions.

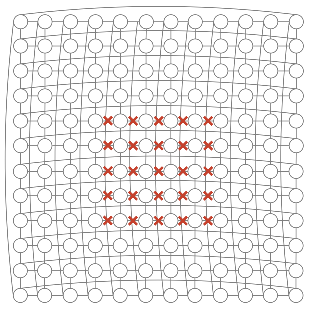
Network Parameters	Value
Network topologies	Torus 32x32
Link bandwidth	2 Gbps
Buffer size	8 KBytes
Packet payloads	256 Bytes, 512 Bytes
Packet generation rate	400 packets/sec/node
Traffic patterns	Uniform Permutation

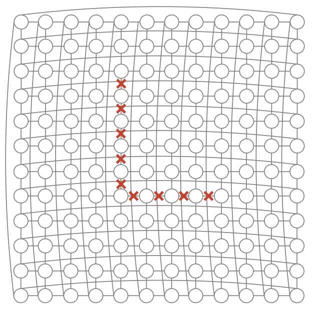
Table 5.9: Simulation parameters for permanent failures and spatial faults (part I).

FT-DRB Parameters	Value
Number of link failures	Variable (according to the fault pattern)
Start time of link failures	0
Duration of link failures	100%
Number of alternative paths	4
Availability traces	-
Spatial fault regions	Convex: line, square. Concave: L, U

Table 5.10: Simulation parameters for permanent failures and spatial faults (part II).


(a) Line (convex)


(b) Square (convex)


(c) L-shape (concave)

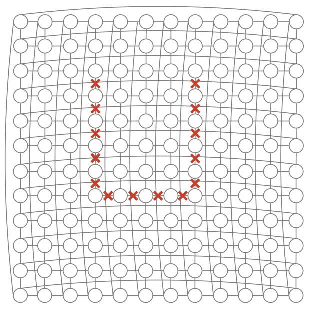

(d) U-shape (concave)

Figure 5.29: Examples of spatial fault patterns.

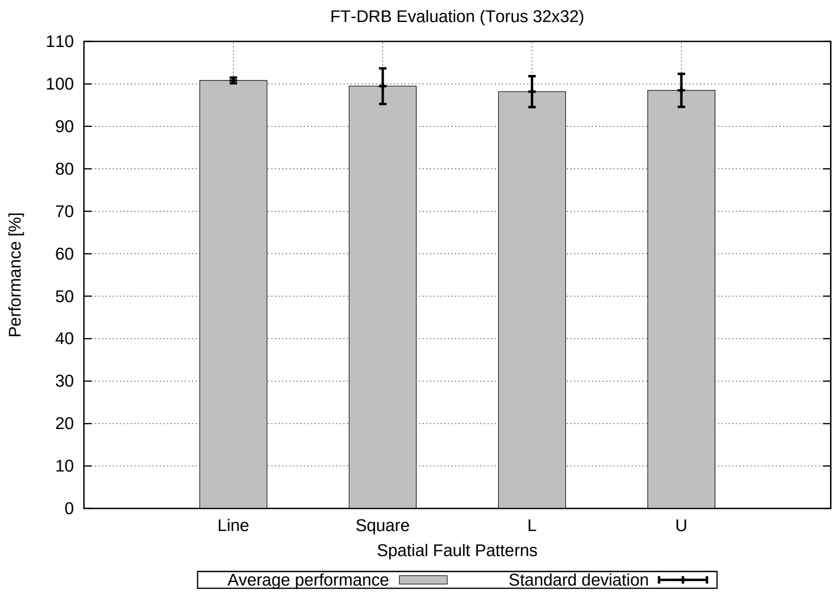


Figure 5.30: Evaluation results of spatial faults patterns.

5.6.3 Permanent Failures with Collective Communication

The evaluation of collective communication patterns is intended to analyze performance results from the viewpoint of communication libraries widely used in HPC systems, Message Passing Interface. To this end, we have conducted some experiments to study two widely used collective communications patterns: *One-to-All* (*Scatter*) and *All-to-One* (*Gather*). A MPI-like master-worker framework approach was used to evaluate the performance of the method. The *One-to-All* pattern is used in *master-worker* based frameworks when the master process sends messages to its worker processes. In this pattern, only one processing node sends different messages to the rest of the nodes in the network. While the *All-to-One* pattern is related to the information sent from the workers to the master where all the processing nodes, but the master, send messages to only one node. The set of simulation parameters used in this group of evaluation is summarized in Tables 5.11 and 5.12.

As mentioned above, the *One-to-All* communication pattern is intended to represent the master process sending messages to its workers, while the *All-to-One* pattern is intended to represent communications in the opposite direction. The implemented *One-to-All* and *All-to-One* communication patterns faithfully reproduce such behavior.

Collective communication results are shown in Fig. 5.31. As could be seen in the figure, the performance degradation for these patterns is lower than 1% (in the worst case). This means that FR-DRB is able to give support to these widely used collective communication patterns, with almost no degradation, even with 10% of network links failed (due to the multipath approach of the method).

5.6.4 Transient Failures

The evaluation approach used for transient link failures is similar to the one previously applied to permanent failures in subsection 5.6.1. The only difference is the random value assigned to the start time and duration of link failures. The set of simulation parameters used in this group of evaluation is summarized in Tables 5.13 and 5.14.

The results for 2, 4, 6, 8 and 10 link failures in the 8x8 torus topology are shown in Fig. 5.32; results of the 16x16 torus topology in Fig. 5.33; and results of the Fat-tree 4-ary 3-tree in Fig. 5.34. In all three topologies, the differential treatment of permanent and transient link failures improves the overall performance of the system. These improvements have been achieved through a better utilization of available resources over time. For the 8x8 torus topology, the lower performance value is above 99.5 %, an improvement of about 14% over the evaluation of permanent failures shown in Fig. 5.26. For the 16x16 torus topology, the improvement is about 8%; while it reaches a 12% in the Fat-tree.

Network Parameters	Value
Network topologies	Torus 32x32
Link bandwidth	2 Gbps
Buffer size	8 KBytes
Packet payloads	256 Bytes, 512 Bytes
Packet generation rate	400 packets/sec/node
Traffic patterns	One-to-all (Scatter), All-to-One (Gather)

Table 5.11: Simulation parameters for permanent failures and collective patterns (part I).

FT-DRB Parameters	Value
Number of link failures	2, 4, 6, 8, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200
Start time of link failures	0
Duration of link failures	100%
Number of alternative paths	4

Table 5.12: Simulation parameters for permanent failures and collective patterns (part II).

Network Parameters	Value
Network topologies	Torus 8x8, Torus 16x16, Fat-tree 4-ary 3-tree
Link bandwidth	2 Gbps
Buffer size	8 KBytes
Packet payloads	256 Bytes, 512 Bytes
Packet generation rate	400 packets/sec/node
Traffic patterns	Uniform Permutation, Bit Reversal, Butterfly, Perfect Shuffle, Matrix Transpose, Complement

Table 5.13: Simulation parameters for transient failures and synthetic patterns (part I).

FT-DRB Parameters	Value
Number of link failures	0, 2, 4, 6, 8, 10, 10%
Start time of link failures	Random
Duration of link failures	Random (0-10%)
Number of alternative paths	4

Table 5.14: Simulation parameters for transient failures and synthetic patterns (part II).

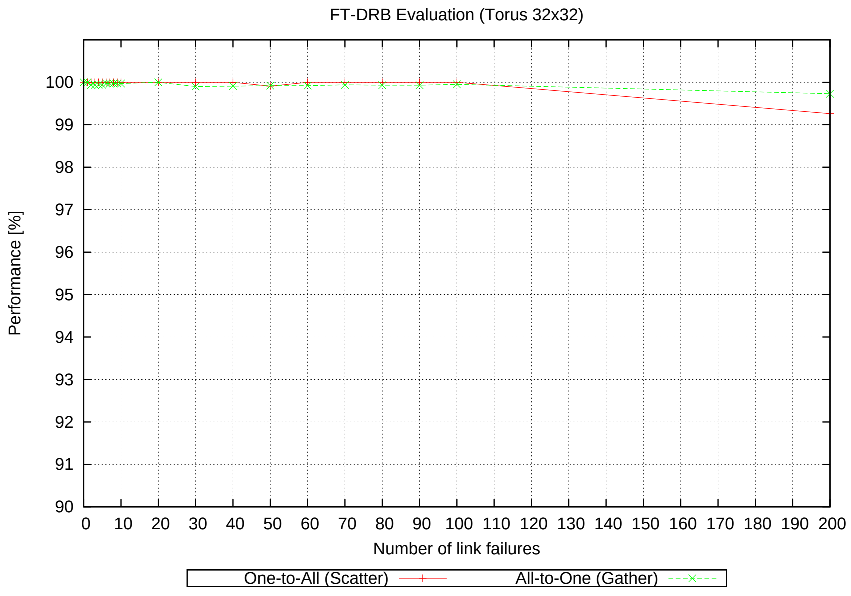


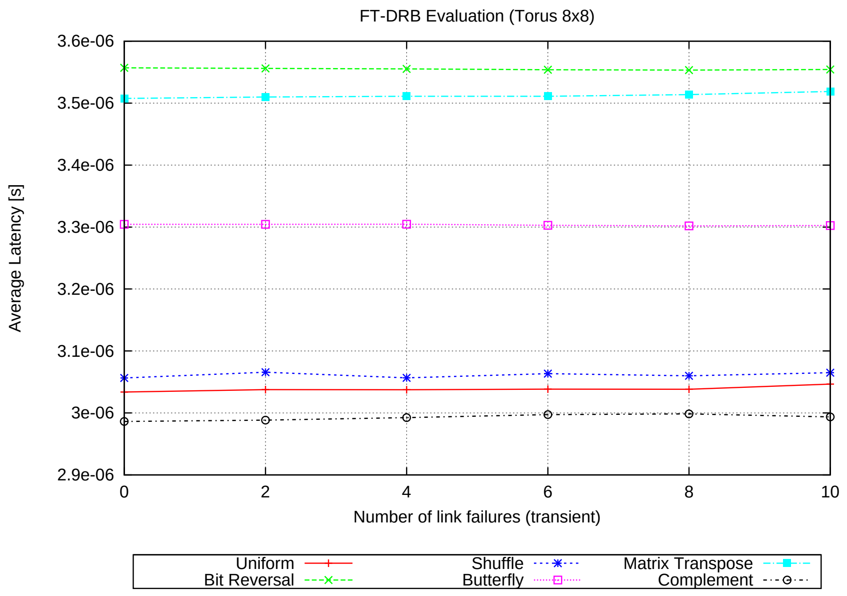
Figure 5.31: Evaluation results of collective communication patterns.

5.6.5 Real-based Failures

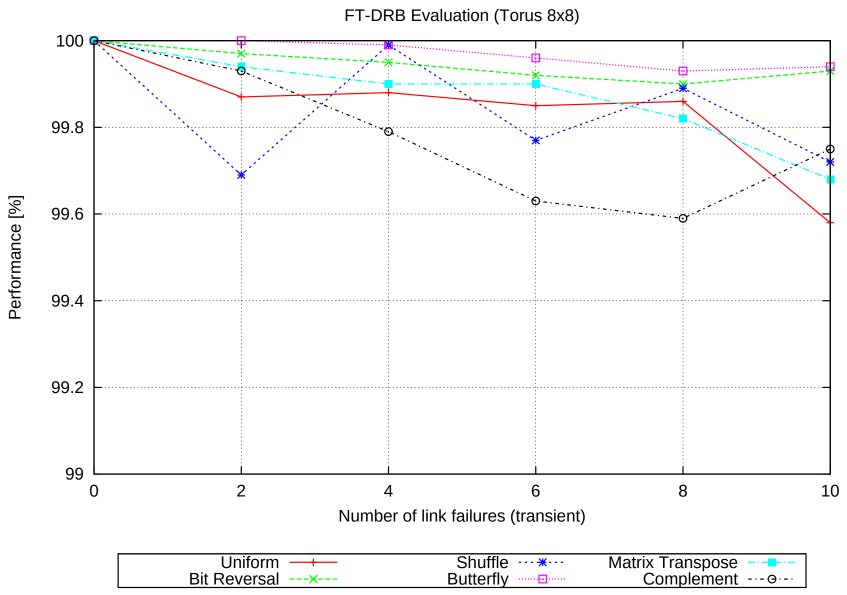
The evaluation of real scenarios is based on failure traces of real systems belonging to the *Los Alamos National Laboratory (LANL)* [49] and the *Pacific Northwest National Laboratory (PNNL)* [73]. These failure traces were obtained from the public failure data repositories CFDR [98] and FTA [29]. Four HPC systems were chosen to be simulated, taking into account their number of nodes and network failures (detailed in Table 5.2).

The number and duration of link failures varies according to the failures traces of such systems (see Table 5.2). Notice that failures lengths have been normalized to the simulation time. Since the *PNNL MPP2* trace provides no information about the failures length, they were all considered as permanent. However, as the information about the length of the fault is available for LANL systems, their failures were considered as transient.

The performance results of test scenarios based on the four systems are shown in Figs. 5.35 (LANL 12), 5.36 (LANL 18), 5.37 (LANL 19) 5.38 (PNNL MPP2). The impact of differences in the number and the duration of failures can be clearly seen the performance values of all systems, particularly for the *PNNL MPP2* machine. The degradation for this system is higher because the *PNNL MPP2* machine presents the highest number of failures and, as explained above, those failures were considered as permanent (unlike the other three scenarios).

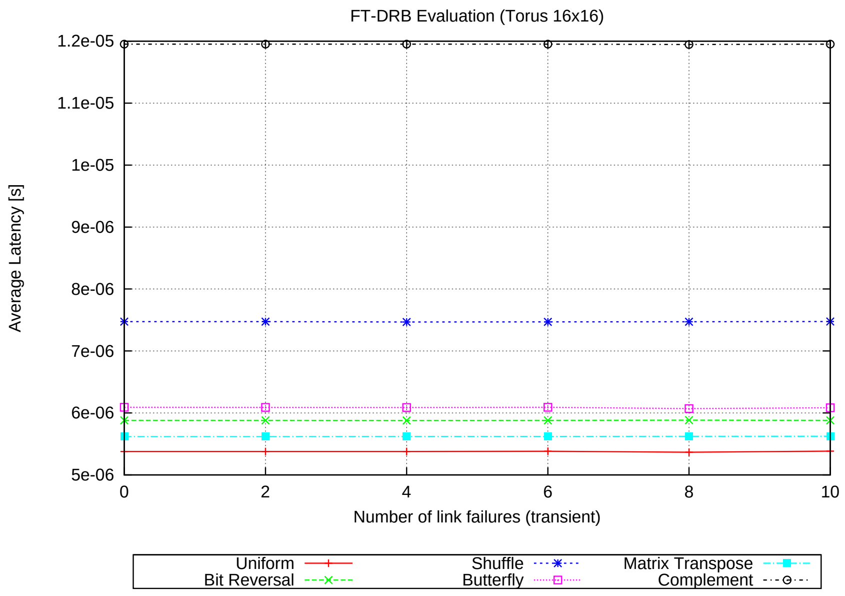


(a) Absolute values

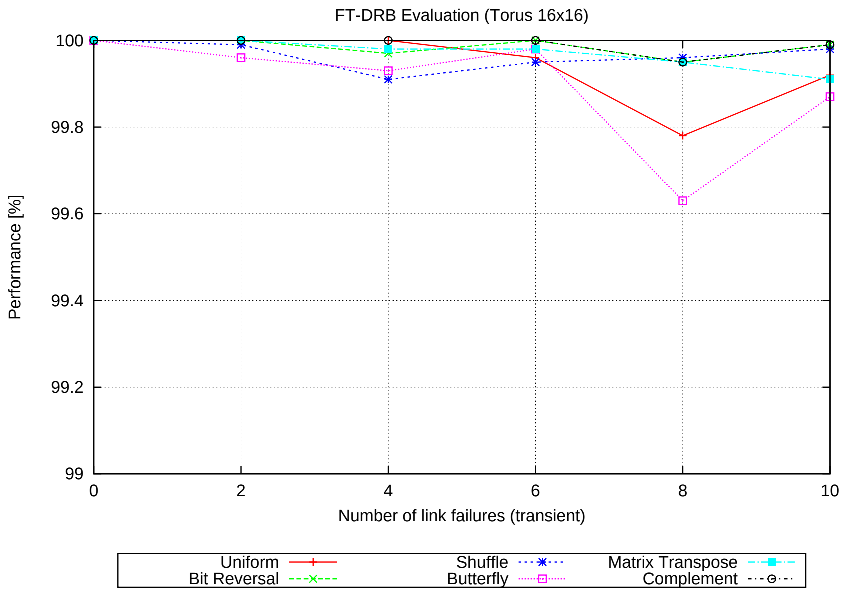


(b) Percentages

Figure 5.32: Evaluation results of transient link failures in Torus 8x8.

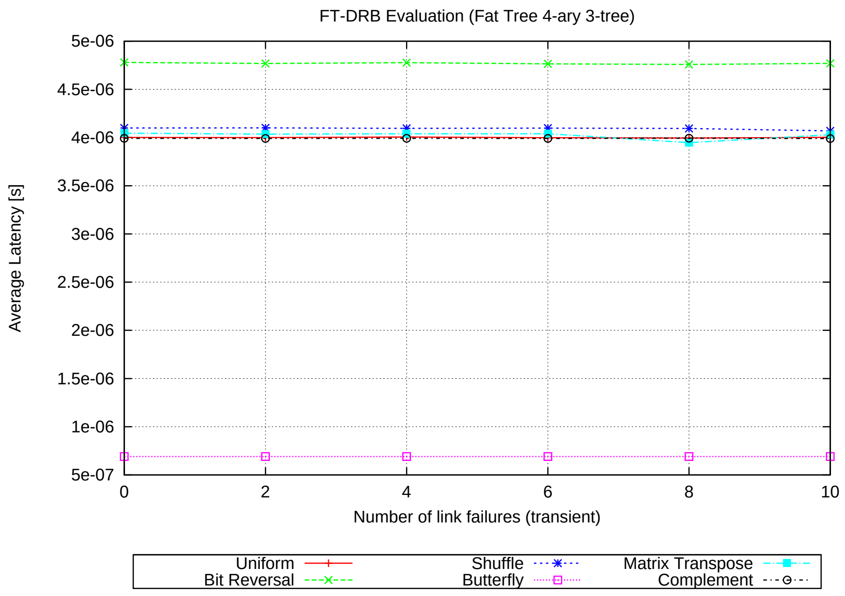


(a) Absolute values

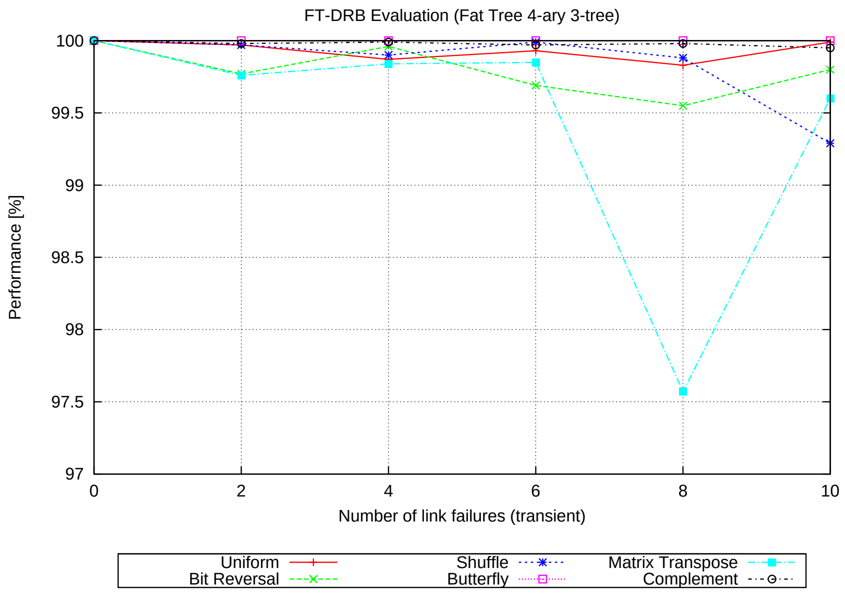


(b) Percentages

Figure 5.33: Evaluation results of transient link failures in Torus 16x16.



(a) Absolute values



(b) Percentages

Figure 5.34: Evaluation results of transient link failures in Fat-tree 4-ary 3-tree.

If considering the *Uniform* pattern, the average performance degradation for the entire set of real-based scenarios is less than 2%. In the worst case, when considering all the failures as permanent, the performance degradation is about 3%. In contrast, when failures were considered as transient, the degradation were below 1% as for the *LANL* machines.

5.6.6 Evaluation of Alternative Paths

To conclude the performance evaluation of FT-DRB, we have conducted some experiments varying the number of alternative paths but keeping the same number and duration of link failures. The simulation parameters of are summarized in Tables 5.15 and 5.16.

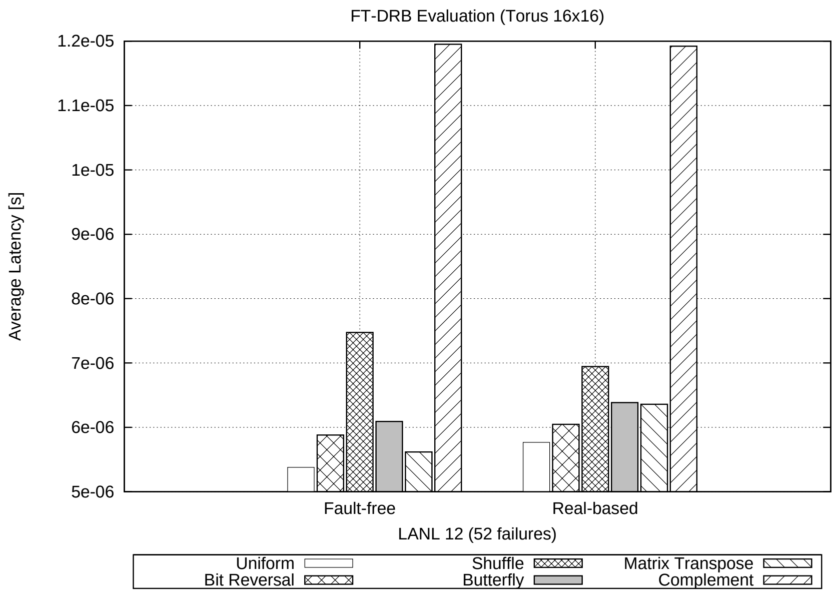
The results of this evaluation are shown in Fig. 5.39. There are two important points to highlight about these results. The first of these points is that for moderate traffic generation rates and a fixed number of link failures, the number of simultaneous alternative paths do not produce significant changes in the average latency. However, some patterns such as the *Bit reversal* and *Butterfly* show downward trends in latency when increasing the number of alternative paths. The second point to mark is the performance gap between some communication patterns such as *Matrix transpose* and *Complement*, even for situations where just one alternative path is used.

Network Parameters	Value
Network topologies	Torus 16x16
Link bandwidth	2 Gbps
Buffer size	8 KBytes
Packet payloads	256 Bytes, 512 Bytes
Packet generation rate	400 packets/sec/node
Traffic patterns	Uniform Permutation, Bit Reversal, Butterfly, Perfect Shuffle, Matrix Transpose, Complement

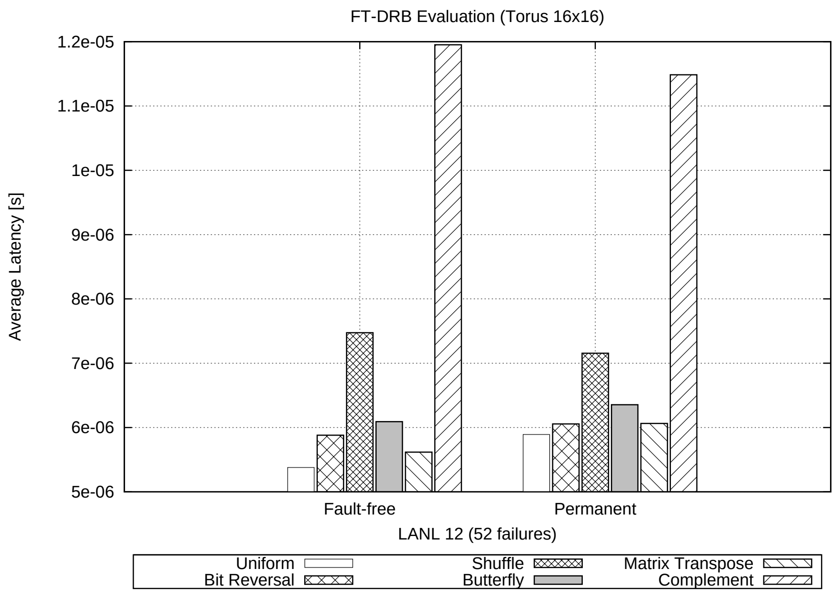
Table 5.15: Simulation parameters for the evaluation of alternative paths (part I).

FT-DRB Parameters	Value
Number of link failures	10
Start time of link failures	0
Duration of link failures	100%
Number of alternative paths	1, 2, 3, 4, 5
Availability traces	-
Spatial fault regions	-

Table 5.16: Simulation parameters for the evaluation of alternative paths (part II).

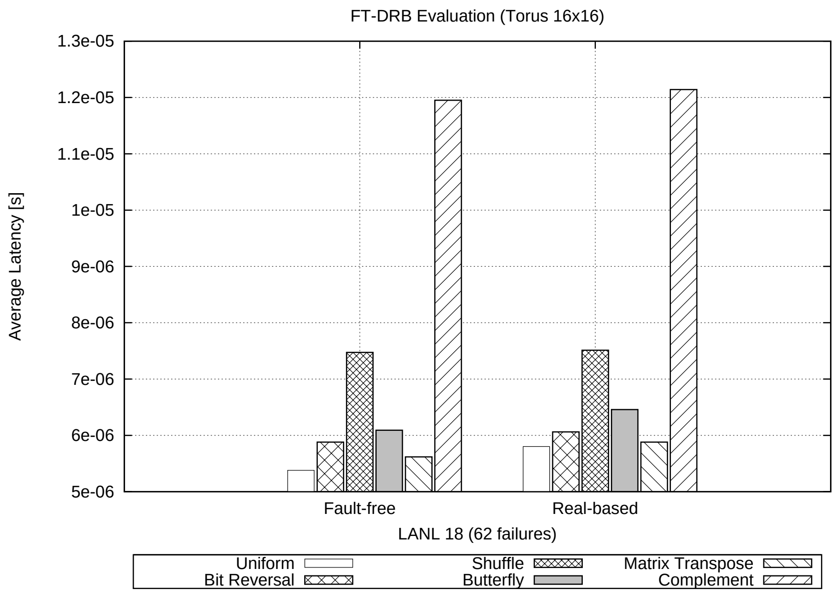


(a) Real duration

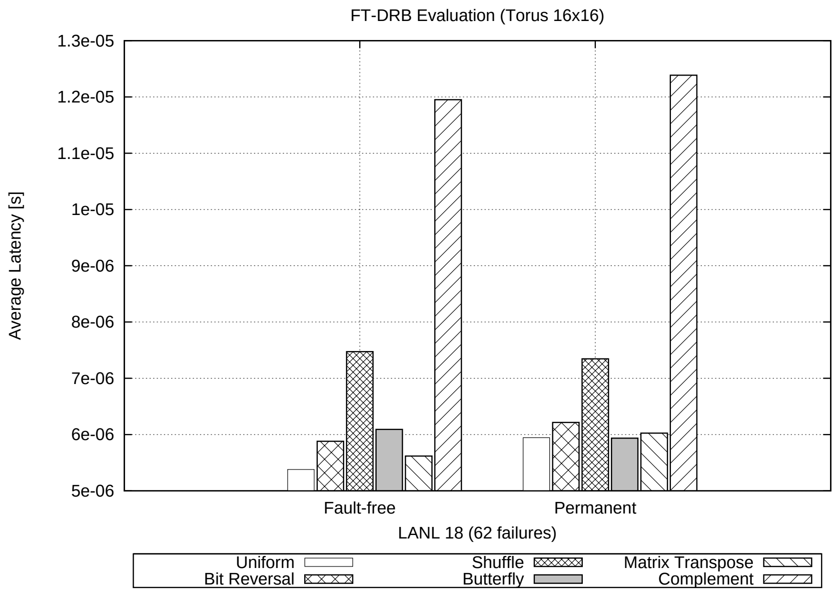


(b) Permanent

Figure 5.35: Evaluation results of real-based link failures for system LANL 12.

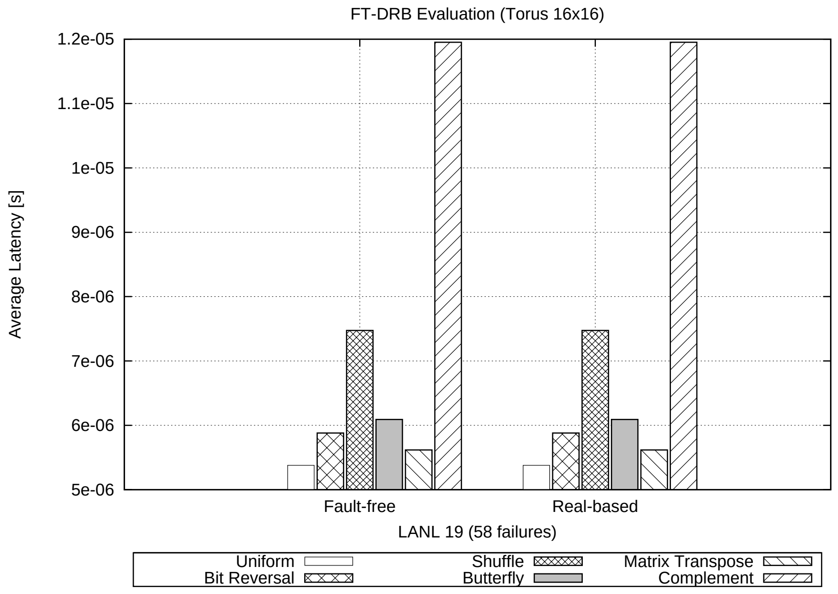


(a) Real duration

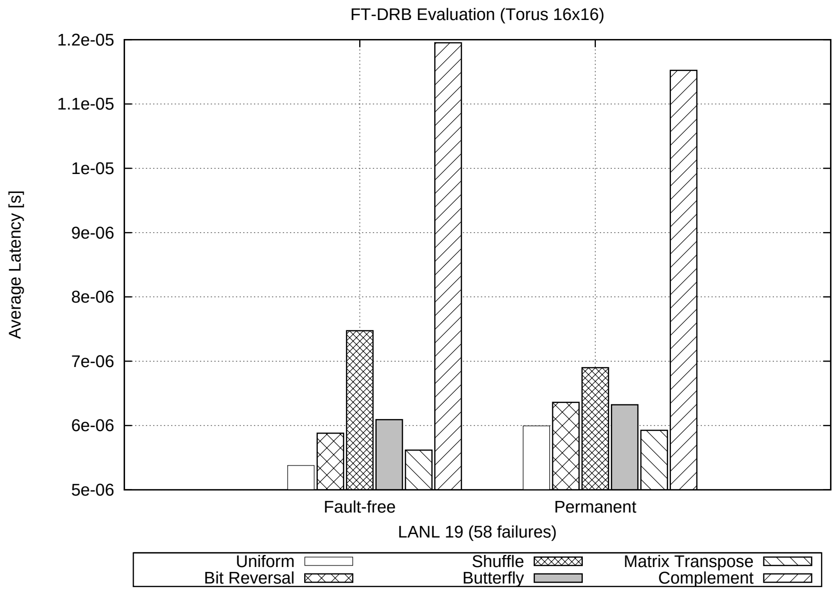


(b) Permanent

Figure 5.36: Evaluation results of real-based link failures for system LANL 18.

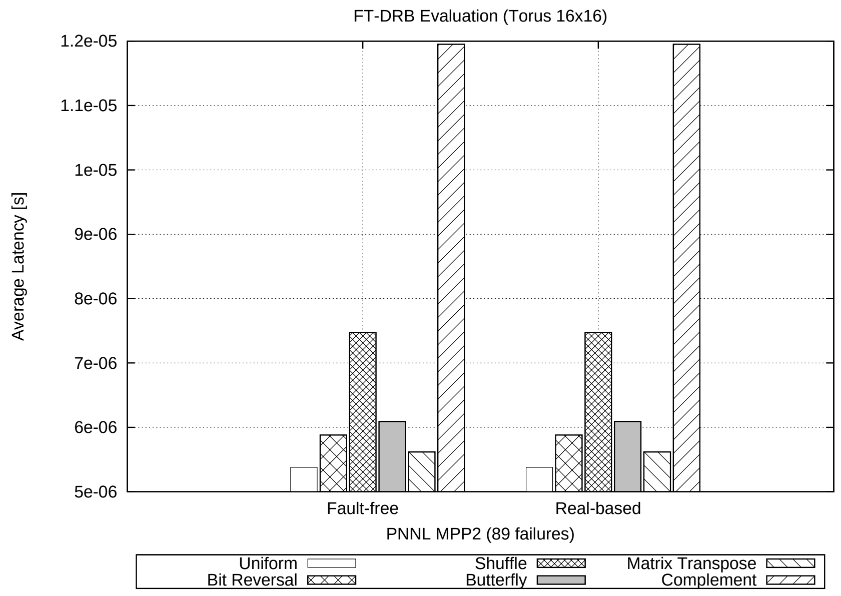


(a) Real duration

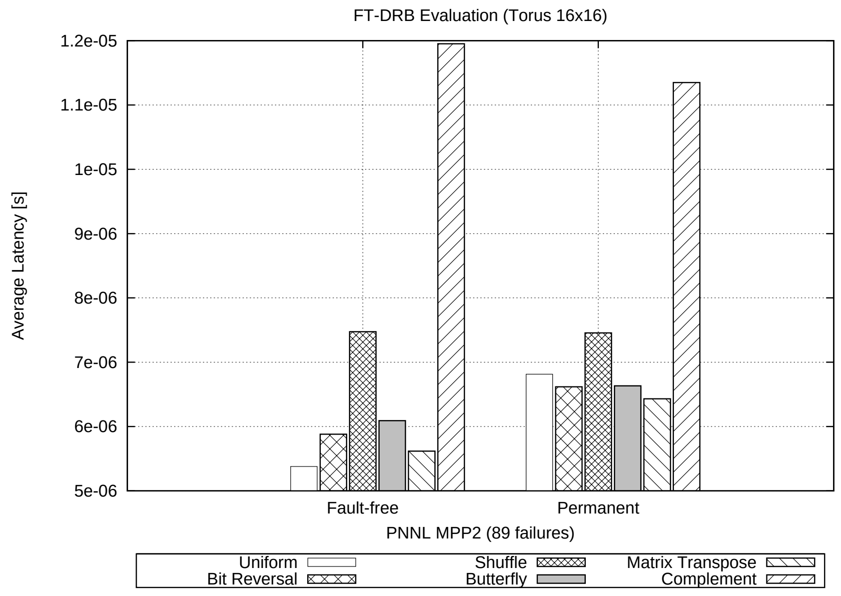


(b) Permanent

Figure 5.37: Evaluation results of real-based link failures for system LANL 19.



(a) Real duration



(b) Permanent

Figure 5.38: Evaluation results of real-based link failures for system PNNL MPP2.

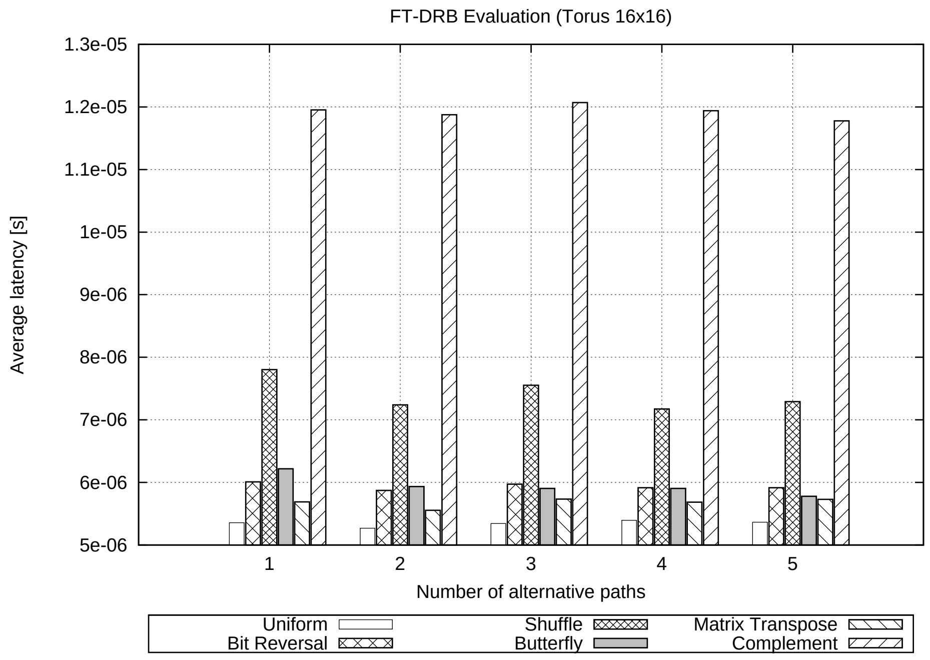


Figure 5.39: Evaluation results for different number of alternative paths with 10 permanent link failures.

5.6.7 Discussion

The strongest point of FT-DRB is that it provides a simple but sufficiently complete and satisfactory solution to the problem of dynamic fault tolerance in interconnection networks. This constitutes the major contribution of the method since the most significant information about a network's fault tolerance is whether it can function at all in the presence of faults. It is also possible to infer from the evaluation results that FT-DRB do not degrade the performance of the system in the absence of failures.

As most current routing approaches only treat static failures, it is not possible to conduct realistic performance comparisons against them. One of the only proposals dealing with dynamic failures was presented in [66] and [65]. According to their performance evaluation graphs, their method achieves average latency values of about 82%. When applying the same evaluation conditions (3 link failures in a 16x16 torus network with 90% of traffic load), the average latency value of our method is about 95%. Our method obtains an improvement of about 70% because the method presented in [66] is based on a variation of the *turn-model* routing, which uses non-faulty paths but lacks of a suitable load balancing technique [14].

Acknowledgments

We would like to thank the Computing, Communications, and Networking Division at *Los Alamos National Laboratory (LANL)* for providing us the real-systems failure data, and specifically to Gary Grider, Laura Davey and James Nunez for their efforts and help; as well as Evan Felix and David Brown from the *Pacific Northwest National Laboratory (PNNL)* for collecting the data and sharing it. The data was collected and made available using the MSC Facility in the William R. Wiley Environmental Molecular Sciences Laboratory (sponsored by the U.S. Department of Energy's Office of Biological and Environmental Research).

Furthermore, we thank *OPNET Technologies, Inc.* for providing us the *OPNET Modeler* licenses to perform the experimental evaluation of this thesis and the previous work [102], [103], [104], [106], [107], [109].

Chapter 6

Conclusions

In the previous chapters, we have presented the conception, design, implementation and evaluation of the two contributions of this thesis. The first of these contributions is the adaptive multipath routing method *Fault-tolerant Distributed Routing Balancing* (FT-DRB). This method has been designed to exploit the communication path redundancy available in many network topologies, allowing interconnection networks to perform in the presence of a large number of faults. The second contribution is the scalable deadlock avoidance technique *Non-blocking Adaptive Cycles* (NAC), specifically designed for interconnection networks suffering from a large number of failures. This technique has been designed and implemented with the aim of ensuring freedom from deadlocks in the proposed fault-tolerant routing method FT-DRB.

Throughout this thesis, we have been following the steps of the scientific research method; ranging from the planning and discussion of objectives and methods, up to the testing and validation of proposals. In the first chapter, we have introduced the objectives, motivation and challenges that have led the development of this work. Then, we have defined the theoretical background of this thesis in chapter 2. Taking these definitions as a starting point, we have been capable of performing the analysis of issues influencing the development of the proposed fault-tolerant routing policies, putting special emphasis on the fault tolerance theory. From this approach, we have been able to provide flexible solutions to the problem of fault tolerance and deadlock avoidance for high-speed interconnection networks in chapters 3 and 4, respectively. Both proposals have been implemented and widely evaluated in different network topologies through modeling and simulation tools, as described in chapter 5. Having completed all the stages that comprise our research, we are now able to present the final conclusions and further work of this thesis.

6.1 Final Conclusions

The strongest point of this thesis is that it provides a simple but sufficiently complete and satisfactory solution to the problem of dynamic fault tolerance in interconnection networks. To this end, we have designed and implemented an adaptive multipath routing method for treating a large number of network link failures in high-performance computing systems (FT-DRB). In addition, we have also proposed and evaluated a complete and scalable deadlock avoidance technique specifically designed to deal with large interconnection networks suffering from a large number of dynamic faults (NAC).

Fault-tolerant Distributed Routing Balancing (FT-DRB)

We have developed a multipath fault-tolerant routing method capable of treating a large number of dynamic link failures. The proposed method allows messages to reach their destination even in the presence of a large number of link failures. This constitutes the major contribution of the method since the most significant information about a network's fault tolerance is whether it can function at all in the presence of faults. Apart from treating link failures, the method is also able to deal with congestion problems and performance degradations caused by the occurrence of failures. It is worth noting that the method do not degrade at all the system performance in the absence of failures.

The four phases of the fault tolerance theory have been successfully integrated into the routing algorithm. These four phases are: *error detection*, *damage confinement*, *error recovery*, and *fault treatment and continued system service*. In the proposed method, monitoring and failure detection processes represent the phase of error detection. On the other hand, the injection and forwarding of link failure notifications to source nodes correspond to the phase of damage confinement. The error recovery solution consists on configuring and selecting escape paths for rerouting packets still in transit through the original faulty path. Alternative paths are configured using intermediate network nodes to circumvent dynamic link failures *on-the-fly*. Finally, fault treatment has been provided by means of configuring appropriate source-based routing paths between source-destination pairs affected by network link failures, taking into account the network condition after the occurrence of failures.

In order to maximize the use of network resources, FT-DRB detects and applies differential treatments to *permanent* and *transient* link failures. At a first stage, link failures are always considered and treated as *transient* failures. If a failure persists over time, its state is changed from *transient* to *permanent*. Thus, FT-DRB differences between *permanent* and *transient* failures.

Non-blocking Adaptive Cycles (NAC)

We have also proposed a scalable deadlock avoidance technique, *Non-blocking Adaptive Cycles*, specifically designed for interconnection networks suffering from a large number of link failures. This technique has been applied for ensuring deadlock freedom in the proposed *Fault-tolerant Distributed Routing Balancing* method. In addition, the proposed technique allows the design of fully adaptive routing methods in general terms.

This technique is based on the idea of denying the *hold-and-wait* and *circular wait* conditions to avoid deadlock occurrences. Our proposal exploits this idea by means of adding an one-slot deadlock avoidance buffer to each input buffer, and applying a simple set of actions when accessing output buffers with no free space. To this end, NAC comprises three main processes: detection of deadlock prone situations; identification of the routing cycles involved in these situations; and the application of predefined protocols to ensure the normal functioning of the system under these circumstances.

The proposed technique does not require the use of virtual escape channels thus avoiding their scalability problems. This is one of the advantages of NAC because the resources needed to avoid faults is not proportional to the number of faults to be tolerated. Additionally, the reduction of the number of virtual channels may increase routers speed. This makes it a feasible solution to the problem of deadlock avoidance for current HPC systems.

6.2 Further Work and Open Lines

This thesis, as any work covering several aspects within a certain field of science, is intended to be complete and entirely closed. However, this research also gives rise to a wide range of affordable open lines and further work. These open lines are described below, grouped according to the contribution from which they are originated.

Fault-tolerant Distributed Routing Balancing (FT-DRB)

One of the points that can be further improved in FT-DRB is the simultaneous use of source-based paths for treating both link failures and congestion problems. To this end, it is necessary to conduct a complete study of the relation between several parameters of the problem, including: the network topology; the number of failures to be treated; the maximum number of alternative paths and *latency thresholds* to be used; the communication pattern and traffic load; etc. In addition, the results of this study could be used as inputs for improving the values of thresholds applied in the treatment of transient and intermittent failures. An additional point that must be considered and treated in the future is to provide solutions to those packets already stored in output buffers affected by link failures. This is an important problem in situations where packet drops cannot be tolerated. Among others, the most important topics are:

- Contemplate non-common failure scenarios (i.e. labyrinths). This point would be based on the inclusion of some sort of *backtracking* algorithm for avoiding *livelock* situations that could arise under these circumstances.
- Design and implement a variant of the routing method that could be used for disabling network components that present low (or null) traffic loads, to enable the replacement of failed components (and also to reduce power consumption).

Non-blocking Adaptive Cycles (NAC)

For the proposed deadlock avoidance technique, further work comprises the inclusion of a reduced number virtual channels to avoid situations such as *Head-of-Line* (HoL) blocking and also to improve performance. Among others, the most important topics include:

- Evaluate the possibility of extending the technique to wormhole switching.
- Measure the time for identifying routing cycles under heavy traffic loads.
- Study the relation between deadlocks occurrences and size of network buffers.
- Study the effect of link failures in the process of routing cycle identification.

6.3 List of Publications

The work presented in this thesis has been published in the following papers.

1. **G. Zarza, D. Lugones, D. Franco, and E. Luque. *A Multipath Fault-Tolerant Routing Method for High-Speed Interconnection Networks*, In Euro-Par 2009 Parallel Processing, pp. 1078-1088, Delft, The Netherlands, August 2009. Springer Berlin/Heidelberg.** [102]
   This paper proposes a fault-tolerant routing method designed to deal with the fault tolerance problem for High-Speed Interconnection Networks. This method is able to support a dynamic fault model, while at the same time not requiring network reconfigurations or stopping packet injection at any time, using a limited number of virtual channels.

2. **G. Zarza, D. Lugones, D. Franco, and E. Luque.** ***Multipath Fault-Tolerant Routing Policies for High-Speed Interconnection Networks*****, In Proceedings of the XX Spanish Conference on Parallelism, pp. 487-492, A Coruña, Spain, September 2009.** [104]
   This work focuses on the problem of fault tolerance for high-speed interconnection networks by designing fault-tolerant routing policies to solve a certain number of dynamic failures, without sacrificing system scalability. To accomplish this the method takes advantage of communication path redundancy, by means of adaptive multipath routing approaches, fulfilling the four phases of fault tolerance.

3. **G. Zarza, D. Lugones, D. Franco, and E. Luque.** ***A Multipath Routing Method for Tolerating Permanent and Non-Permanent Faults*****, In 15th Argentine Conference on Computer Science (CACIC), pp. 251-261, Jujuy, Argentina, October 2009.** [103]
   This work focuses on the problem of fault tolerance for high-speed interconnection networks by designing a fault-tolerant routing method to solve an unbounded number of dynamic faults (permanent and non-permanent), by means of a multipath routing approach.

4. **G. Zarza, D. Lugones, D. Franco, and E. Luque.** ***FT-DRB: A Method for Tolerating Dynamic Faults in High-Speed Interconnection Networks*****, In 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP), pp. 77-84, Pisa, Italy, February 2010. IEEE Computer Society.** [106]
   This paper introduces a fault-tolerant routing method, called FT-DRB. The method is based on a multipath routing approach and is able to support a large number of dynamic faults using few additional hardware resources. Moreover, the method includes a deadlock avoidance technique that does not require the use of virtual channels.

5. **G. Zarza, D. Lugones, D. Franco, and E. Luque.** ***Deadlock Avoidance for Interconnection Networks with Multiple Dynamic Faults*****, In 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP), pp. 276-280, Pisa, Italy, February 2010. IEEE Computer Society.** [105]
   This paper introduces a new deadlock avoidance mechanism for routing algorithms designed to deal with multiple dynamic faults. The mechanism is based on adding

a small-sized buffer and applying a simple set of actions when accessing output buffers with limited free space.

6. **G. Zarza, D. Lugones, D. Franco, and E. Luque. *Fault-tolerant Routing for Multiple Permanent and Non-permanent Faults in HPC Systems*, In XVI International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), pp. 144-150, Las Vegas, USA, July 2010. CSREA Press.** [107]
   This paper introduces a fault-tolerant routing method designed to solve a large number of dynamic permanent and non-permanent link faults. As failures appear randomly during system operation, the method provides escape paths for the stalled messages and, at the same time, avoids deadlock occurrences.

7. **G. Zarza, D. Lugones, D. Franco, and E. Luque. *Non-blocking Adaptive Cycles: Deadlock Avoidance for Fault-tolerant Interconnection Networks*, In IEEE International Conference on Cluster Computing 2010 (Cluster Workshops), pp. 1-4, Heraklion, Crete, Greece, September 2010. IEEE Computer Society.** [108]
   This work presents a complete deadlock avoidance technique called Non-blocking Adaptive Cycles (NAC), specifically designed to ensure deadlock freedom in faulty networks without requiring any virtual channels. This work is a fully-enhanced three-stages version of the preliminary work presented in [105].

8. **G. Zarza, D. Lugones, D. Franco, and E. Luque. *A Multipath Routing Method for Tolerating Permanent and Non-Permanent Faults*, In XV Argentine Congress of Computer Science Selected Papers, Journal of Computer Science & Technology (JCS&T), pp. 97-107, ISBN 978-950-34-0684-7, October 2010. Pearson.** [109].
   Idem [103].

In addition, a draft of the deadlock avoidance method presented in chapter 4 has been submited to the Spanish Patent and Trademark Office:

1. **G. Zarza, D. Lugones, D. Franco, and E. Luque. *Parallel and Scalable Method for Identifying and Tracking Resource Dependency Cycles in High-speed Interconnection Networks.*** (Método paralelo y escalable de identificación y seguimiento de ciclos de dependencias de recursos en redes de interconexión de alta velocidad).

Bibliography

[1] M. Abd-El-Barr. *Design and Analysis of Reliable and Fault-tolerant Computer Systems*. Imperial College Press, London, UK, 2007. ISBN 9781860946684.

[2] N. Adiga, M. Blumrich, D. Chen, P. Coteus, A. Gara, et al. Blue Gene/L Torus Interconnection Network. *IBM Journal of Research and Development*, 49(2/3): 265–276, April 2005. doi:10.1147/rd.492.0265.

[3] K. Anjan. and T. Pinkston. An Efficient, Fully Adaptive Deadlock Recovery Scheme: DISHA. *ACM SIGARCH Computer Architecture News*, 23(2):201–210, 1995. ISSN 0163-5964. doi:10.1145/225830.224431.

[4] K. Anjan, T. Pinkston, and J. Duato. Generalized Theory for Deadlock-free Adaptive Wormhole Routing and its Application to Disha Concurrent. In *Proceedings of the 10th International Parallel Processing Symposium (IPPS)*, pages 815–821, apr 1996. doi:10.1109/IPPS.1996.508153.

[5] L. Barroso, J. Dean, and U. Holzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23(2):22–28, March-April 2003. ISSN 0272-1732. doi:10.1109/MM.2003.1196112.

[6] S. Chalasani and R. Boppana. Adaptive wormhole routing in tori with faults. *IEE Proceedings on Computers and Digital Techniques*, 142(6):386–394, nov 1995. ISSN 1350-2387. doi:10.1049/ip-cdt:19952079.

[7] S. Chalasani and R. Boppana. Communication in multicomputers with nonconvex faults. *IEEE Transactions on Computers*, 46(5):616–622, may 1997. ISSN 0018-9340. doi:10.1109/12.589238.

[8] C.-L. Chen and G.-M. Chiu. A fault-tolerant routing scheme for meshes with nonconvex faults. *Parallel and Distributed Systems, IEEE Transactions on*, 12(5): 467–475, may 2001. ISSN 1045-9219. doi:10.1109/71.926168.

[9] A. Chien and J. H. Kim. Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors. In *Proceedings of the 19th Annual International Symposium on Computer Architecture (1992)*, pages 268–277, 1992. doi:10.1109/ISCA.1992.753323.

[10] E. G. Coffman, M. Elphick, and A. Shoshani. System Deadlocks. *ACM Computing Survey (CSUR)*, 3(2):67–78, 1971. ISSN 0360-0300. doi:10.1145/356586.356588.

[11] M. Coli and P. Palazzari. An adaptive deadlock and livelock free routing algorithm. In *Proceedings of the Euromicro Workshop on Parallel and Distributed Processing (PDP)*, pages 288–295, jan 1995. doi:10.1109/EMPDP.1995.389126.

[12] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, may 1987. ISSN 0018-9340. doi:10.1109/TC.1987.1676939.

[13] W. J. Dally and C. Seitz. The torus routing chip. *Distributed Computing*, 1:187–196, 1986. ISSN 0178-2770. doi:10.1007/BF01660031.

[14] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2004. ISBN 0122007514 9780122007514.

[15] W. J. Dally., L. Dennison, D. Harris, K. Kan, and T. Xanthopoulos. The reliable router: A reliable and high-performance communication substrate for parallel computers. In *PCRCW '94: Proceedings of the First International Workshop on Parallel Computer Routing and Communication*, pages 241–255, London, UK, 1994. Springer-Verlag. ISBN 3-540-58429-3. doi:10.1007/3-540-58429-3_41.

[16] G. Dodig-Crnkovic. Scientific methods in computer science. In *Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden*, April 2002. URL http://www.mrtc.mdh.se/index.php?choice=publications&id=0446.

[17] J. Duato. A Theory of Deadlock-Free Adaptive Multicast Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(9):976–987, Sep 1995. ISSN 1045-9219. doi:10.1109/71.466634.

[18] J. Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10): 1055–1067, oct 1995. ISSN 1045-9219. doi:10.1109/71.473515.

[19] J. Duato. A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks. *IEEE Transatcion on Parallel and Distributed Systems*, 7(8):841–854, 1996. ISSN 1045-9219. doi:10.1109/71.532115.

[20] J. Duato. A Theory of Fault-Tolerant Routing in Wormhole Networks. *IEEE Transactions on Parallel and Distributed Systems*, 8(8):790–802, 1997. ISSN 1045-9219. doi:10.1109/71.605766.

[21] J. Duato and T. M. Pinkston. A General Theory for Deadlock-Free Adaptive Routing Using a Mixed Set of Resources. *IEEE Transactions on Parallel and Distributed Systems*, 12(12):1219–1235, 2001. ISSN 1045-9219. doi:10.1109/71.970556.

[22] J. Duato, S. Yalamanchili, and L. M. Ni. *Interconnection networks. An Engineering Approach*. Morgan Kaufmann, San Francisco, CA, 2003. ISBN 9780585457451.

[23] M. H. Farahabady, F. Safaei, A. Khonsari, and M. Fathy. Characterization of Spatial Fault Patterns in Interconnection Networks. *Parallel Computing*, 32(11-12):886–901, 2006. doi:10.1016/j.parco.2006.09.004.

[24] M. H. Farahabady, F. Safaei, A. Khonsari, and M. Fathy. On the fault patterns properties in the torus networks. In *IEEE International Conference on Computer Systems and Applications*, pages 215–220, 8 2006. doi:10.1109/AICCSA.2006.205092.

[25] D. Franco, I. Garcés, and E. Luque. Distributed Routing Balancing for Interconnection Network Communication. In *HiPC '98: Proceedings of the 5th International Conference On High Performance Computing*, pages 253–261, Madras, India, 1998. doi:10.1145/508791.508951.

[26] D. Franco, I. Garcés, and E. Luque. Dynamic Routing Balancing in Parallel Computer Interconnection Networks. In *VECPAR '98: Selected Papers and Invited Talks from the Third International Conference on Vector and Parallel Processing*, pages 494–507, London, UK, 1999. Springer-Verlag. ISBN 3-540-66228-6. doi:10.1007/10703040_37.

[27] D. Franco, I. Garcés, and E. Luque. A New Method to Make Communication Latency Uniform: Distributed Routing Balancing. In *ICS '99: Proceedings of the 13th international conference on Supercomputing*, pages 210–219, New York, NY, USA, 1999. ACM. ISBN 1-58113-164-X. doi:10.1145/305138.305195.

[28] D. Franco, I. Garcés, and E. Luque. Avoiding Communication Hot-Spots in Interconnection Networks. In *HICSS-32: Proceedings of the 32nd Annual Hawaii*

International Conference on System Sciences, page 10 pp., Maui, HI , USA, 1999. doi:10.1109/HICSS.1999.773072.

[29] FTA. Failure Trace Archive, April 2011 . URL http://fta.inria.fr/.

[30] C. Glass and L. Ni. The Turn Model for Adaptive Routing. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 278–287, 1992. doi:10.1109/ISCA.1992.753324.

[31] C. Gómez, M. E. Gómez, P. López, and J. Duato. An Efficient Fault-Tolerant Routing Methodology for Fat-Tree Interconnection Networks. In *Parallel and Distributed Processing and Applications*, volume 4742 of *Lecture Notes in Computer Science*, pages 509–522. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-74741-3. doi:10.1007/978-3-540-74742-0_46.

[32] M. E. Gómez, J. Duato, J. Flich, P. López, A. Robles, N. Nordbotten, T. Skeie, and O. Lysne. A New Adaptive Fault-Tolerant Routing Methodology for Direct Networks. In *Proceedings of the International Conference on High Performance Computing*, number 3296 in Lecture Notes in Computer Science, pages 462–473. Springer-Verlag, 2004. doi:10.1007/978-3-540-30474-6_49.

[33] M. E. Gómez, N. A. Nordbotten, J. Flich, P. Lopez, A. Robles, J. Duato, T. Skeie, and O. Lysne. A Routing Methodology for Achieving Fault Tolerance in Direct Networks. *IEEE Transactions on Computers*, 55(4):400–415, 2006. doi:10.1109/TC.2006.46.

[34] A. Greenberg and B. Hajek. Deflection routing in hypercube networks. *IEEE Transactions on Communications*, 40(6):1070–1081, jun 1992. ISSN 0090-6778. doi:10.1109/26.142797.

[35] H. Gu, J. Zhang, K. Wang, Z. Liu, and G. Kang. Enhanced fault tolerant routing algorithms using a concept of "balanced ring". *Journal of Systems Architecture*, 53 (12):902–912, 2007. ISSN 1383-7621. doi:10.1016/j.sysarc.2007.03.002.

[36] C.-T. Ho and L. Stockmeyer. A new approach to fault-tolerant wormhole routing for mesh-connected parallel computers. *IEEE Transactions on Computers*, 53(4): 427–438, April 2004. doi:10.1109/TC.2004.1268400.

[37] L.-H. Hsu and C.-K. Lin. *Graph Threory and Interconnetion Networks.* CRC Press, 2008. ISBN 9781420044812.

[38] IBM Blue Gene Team. Overview of the IBM Blue Gene/P project. *IBM Journal of Research and Development*, 52(1.2):199–220, January 2008. ISSN 0018-8646. doi:10.1147/rd.521.0199.

[39] InfiniBand Trade Association. *InfiniBand architecture specification: release 1.2*, volume 1. InfiniBand Trade Association, Portland, OR, 2004. URL http://www.infinibandta.org/.

[40] R. Jain. Congestion control in computer networks: issues and trends. *IEEE Network*, 4(3):24–30, may 1990. ISSN 0890-8044. doi:10.1109/65.56532.

[41] P. Jalote. *Fault Tolerance in Distributed Systems.* PTR Prentice Hall, Englewood Cliffs, N.J., USA, 1994. ISBN 9780133013672.

[42] K. Kanoun and L. Spainhower. *Dependability Benchmarking for Computer Systems.* Wiley-IEEE Computer Society Press, 2008. ISBN 9780470230558.

[43] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3(4):267–286, 1979. ISSN 0376-5075. doi:10.1016/0376-5075(79)90032-1.

[44] J. Kim, Z. Liu, and A. Chien. Compressionless routing: a framework for adaptive and fault-tolerant routing. *IEEE Transactions on Parallel and Distributed Systems*, 8(3):229–244, mar 1997. ISSN 1045-9219. doi:10.1109/71.584089.

[45] S. Konstantinidou and L. Snyder. The chaos router: a practical application of randomization in network routing. In *SPAA '90: Proc. of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 21–30, New York, USA, 1990. ISBN 0-89791-370-1. doi:10.1145/97444.97452.

[46] S. Konstantinidou and L. Snyder. Chaos router: architecture and performance. *ACM SIGARCH Computer Architecture News - Special Issue*, 19(3):212–221, 1991. ISSN 0163-5964. doi:10.1145/115953.115974.

[47] S. Konstantinidou and L. Snyder. The Chaos Router. *IEEE Transactions on Computers*, 43(12):1386–1397, December 1994. ISSN 0018-9340. doi:10.1109/12.338098.

[48] I. Koren and C. M. Krishna. *Fault-Tolerant Systems.* Elsevier/Morgan Kaufmann, Amsterdam, 2007. ISBN 9780120885251.

[49] LANL. Los Alamos National Laboratory, April 2011. URL http://www.lanl.gov/.

[50] J.-C. Laprie. Dependable Computing and Fault Tolerance: Concepts and Terminology. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing*, pages 2–11, June 1995. doi:10.1109/FTCSH.1995.532603.

[51] D. Linder and J. Harden. An adaptive and fault tolerant wormhole routing strategy for k -ary n-cubes. *IEEE Transactions on Computers*, 40(1):2–12, jan 1991. ISSN 0018-9340. doi:10.1109/12.67315.

[52] D. Lugones, D. Franco, and E. Luque. Dynamic Routing Balancing On InfiniBand Networks. *Journal of Computer Science and Technology*, 8(2):104–110, July 2008.

[53] D. Lugones, D. Franco, and E. Luque. Modeling Adaptive Routing Protocols In High Speed Interconnection Networks. In *OPNETWORK Conference*, pages 1–7, 2008. URL http://www.opnet.com/opnetwork2008/.

[54] D. Lugones, D. Franco, E. Argollo, and E. Luque. Models for High-Speed Interconnection Networks Performance Analysis. In *IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS '09)*, pages 1–4, September 2009. doi:10.1109/MASCOT.2009.5366358.

[55] D. Lugones, D. Franco, and E. Luque. Dynamic and Distributed Multipath Routing Policy for High-Speed Cluster Networks. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 396–403, Shanghai, China, May 2009. doi:10.1109/CCGRID.2009.13.

[56] D. Lugones, D. Franco, and E. Luque. Fast-Response Dynamic Routing Balancing for High-speed Interconnection Networks. In *IEEE International Conference on Cluster Computing and Workshops (CLUSTER WORKSHOPS)*, pages 1–9, September 2009. doi:10.1109/CLUSTR.2009.5289142.

[57] D. Lugones, D. Franco, and E. Luque. Adaptive Multipath Routing for Congestion Control in InfiniBand Networks. In *International Conference on Parallel Processing Workshops (ICPPW)*, pages 222–227, Sep. 2009. doi:10.1109/ICPPW.2009.38.

[58] D. Lugones, D. Franco, D. Rexachs, J. Moure, E. Luque, E. Argollo, A. Falcon, D. Ortega, and P. Faraboschi. High-speed Network Modeling for Full System Simulation. In *IEEE International Symposium on Workload Characterization (IISWC)*, pages 24–33, October 2009. doi:10.1109/IISWC.2009.5306799.

[59] O. Lysne and J. Duato. Fast Dynamic Reconfiguration in Irregular Networks. In *Proceedings of the International Conference on Parallel Processing*, pages 449–458, 2000. doi:10.1109/ICPP.2000.876161.

[60] O. Lysne, J. M. nana, J. Flich, J. Duato, T. Pinkston, and T. Skeie. An efficient and deadlock-free network reconfiguration protocol. *IEEE Transactions on Computers*, 57(6):762 –779, june 2008. ISSN 0018-9340. doi:10.1109/TC.2008.31.

[61] J. Martinez, P. Lopez, J. Duato, and T. Pinkston. Software-based deadlock recovery technique for true fully adaptive routing in wormhole networks. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 182–189, aug 1997. doi:10.1109/ICPP.1997.622586.

[62] A. Mejia, J. Flich, J. Duato, S. Reinemo, and T. Skeie. Segment-Based Routing: An Efficient Fault-Tolerant Routing Algorithm For Meshes and Tori. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–10, April 2006. doi:10.1109/IPDPS.2006.1639341.

[63] J. Montañana, J. Flich, and J. Duato. Epoch-based reconfiguration: Fast, simple, and effective dynamic network reconfiguration. In *International Parallel and Distributed Processing Symposium (IPDPS 2008)*, pages 1–12. IEEE Computer Society, April 2008. doi:10.1109/IPDPS.2008.4536298.

[64] L. Ni and P. McKinley. A Survey of Wormhole Routing Techniques in Direct Networks. *IEEE Computer*, 26(2):62–76, February 1993. ISSN 0018-9162. doi:10.1109/2.191995.

[65] N. A. Nordbotten. *Fault-Tolerant Routing in Interconnection Networks.* PhD thesis, University of Oslo, 2008.

[66] N. A. Nordbotten and T. Skeie. A Routing Methodology for Dynamic Fault Tolerance in Meshes and Tori. In *International Conference on High Performance Computing (HiPC)*, LNCS 4873, pages 514–527, 2007. ISBN 978-3-540-77219-4. doi:10.1007/978-3-540-77220-0_47.

[67] N. A. Nordbotten, M. E. Gómez, J. Flich, P. López, A. Robles, T. Skeie, O. Lysne, and J. Duato. A fully adaptive fault-tolerant routing methodology based on intermediate nodes. In H. Jin, G. R. Gao, Z. Xu, and H. Chen, editors, *NPC*, volume 3222 of *Lecture Notes in Computer Science*, pages 341–356. Springer, 2004. ISBN 3-540-23388-1. doi:10.1007/b100357.

[68] C. Nunez, G. Zarza, D. Lugones, J. Navarro, D. Franco, and E. Luque. ClusterGUI, an Application to Launch OPNET Simulations within Resource Managed Environments. In *OPNETWORK Conference*, pages 1–7, 2011.

[69] OPNET Technologies. Veriying Statistical Validity of Discrete Event Simulations. WhitePaper, 2008. URL http://www.opnet.com/whitepapers/abstracts/vsvodes.html.

[70] OPNET Technologies. OPNET Modeler Accelerating Network R&D, April 2011. URL http://www.opnet.com/.

[71] S. Park, J.-H. Youn, and B. Bose. Fault-tolerant wormhole routing algorithms in meshes in the presence of concave faults. In *Proceeding of the International Parallel and Distributed Processing Symposium (IPDPS)*, pages 633–638, 2000. doi:10.1109/IPDPS.2000.846045.

[72] T. 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, aug. 2003. doi:10.1109/TPDS.2003.1225057.

[73] PNNL. Pacific Northwest National Laboratory, April 2011. URL http://www.pnnl.gov/.

[74] V. Puente and J. A. Gregorio. Immucube: Scalable Fault-Tolerant Routing for k-ary n-cube Networks. *IEEE Transactions on Parallel and Distributed Systems*, 18(6): 776–788, 2007. doi:10.1109/TPDS.2007.1047.

[75] V. Puente, R. Beivide, J. Gregorio, J. 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 (ICPP)*, pages 58–67, 1999. doi:10.1109/ICPP.1999.797388.

[76] V. Puente, C. Izu, R. Beivide, J. Gregorio, F. Vallejo, and J. Prellezo. The adaptive bubble router. *Journal of Parallel and Distributed Computing*, 61(9):1180–1208, 2001. ISSN 0743-7315. doi:10.1006/jpdc.2001.1746.

[77] V. Puente, J. A. Gregorio, R. Beivide, and F. Vallejo. A Low Cost Fault Tolerant Packet Routing for Parallel Computers. In *Proceeding of the International Parallel and Distributed Processing Symposium*, pages 1–8, April 2003. doi:10.1109/IPDPS.2003.1213132.

[78] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide. Immunet: Dependable Routing for Interconnection Networks with Arbitrary Topology. *IEEE Transactions on Computers*, 57(12):1676–1689, 2008. ISSN 0018-9340. doi:10.1109/TC.2008.95.

[79] A. W. Roscoe. Routing Messages Through Networks: An Exercise in Deadlock Avoidance. In *Proceedings of 7th OCCAM User Group Technical Meeting: Programming of Transputer Based Machines*, pages 1–22, 1987. URL http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/21.ps.

[80] F. Safaei, A. Khonsari, A. Dadlani, and M. Ould-Khaoua. A Probabilistic Characterization of Fault Rings in Adaptively-Routed Mesh Interconnection Networks. In *International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN)*, pages 233–238, may 2008. doi:10.1109/I-SPAN.2008.17.

[81] F. Safaei, A. Khonsari, and M. Gilak. A new performance measure for characterizing fault rings in interconnection networks. *Information Sciences*, 180(5):664–678, 2010. ISSN 0020-0255. doi:10.1016/j.ins.2009.10.017.

[82] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 249–258, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2607-1. doi:10.1109/DSN.2006.5.

[83] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications*, 9(8):1318–1335, oct 1991. ISSN 0733-8716. doi:10.1109/49.105178.

[84] S. Scott and G. Thorson. Optimized routing in the Cray T3D. In K. Bolding and L. Snyder, editors, *Parallel Computer Routing and Communication*, volume 853 of *Lecture Notes in Computer Science*, pages 281–294. Springer Berlin / Heidelberg, 1994. doi:10.1007/3-540-58429-3_44.

[85] F. Sem-Jacobsen, T. Skeie, O. Lysne, et al. Siamese-Twin: A Dynamically Fault-Tolerant Fat-Tree. In *International Parallel and Distributed Processing Symposium (IPDPS 2005)*, pages 100b–100b, April 2005. doi:10.1109/IPDPS.2005.401.

[86] F. Sem-Jacobsen, T. Skeie, O. Lysne, and J. Duato. Dynamic fault tolerance in fat-trees. *IEEE Transactions on Computers*, 60(4):508–525, April 2010. ISSN 0018-9340. doi:10.1109/TC.2010.97.

[87] T. Skeie. Handling Multiple Faults in Wormhole Mesh Networks. In *Euro-Par '98: Proceedings of the 4th International Euro-Par Conference on Parallel Processing*, pages 1076–1088. Springer-Verlag, 1998. doi:10.1007/BFb0057969.

[88] T. Skeie. A Fault-tolerant Method for Wormhole Multistage Networks. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages 637–644. CSREA Press, 1998.

[89] Y. H. Song and T. Pinkston. A new mechanism for congestion and deadlock resolution. In *Proceeding of the International Conference on Parallel Processing (ICPP)*, pages 81–90, 2002. doi:10.1109/ICPP.2002.1040862.

[90] Y. H. Song and T. Pinkston. Distributed resolution of network congestion and potential deadlock using reservation-based scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 16(8):686–701, aug. 2005. ISSN 1045-9219. doi:10.1109/TPDS.2005.93.

[91] A. Tanenbaum. *Modern Operating Systems.* Pearson College Div, 2008. ISBN 9780135013014.

[92] A. S. Tanenbaum. *Page Replacement Algorithm.* Betascript Publishing, 2002. ISBN 6132107444.

[93] I. Theiss and O. Lysne. LORE - Local Reconfiguration for Fault Management in Irregular Interconnects. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1–12, April 2004. doi:10.1109/IPDPS.2004.1302916.

[94] I. Theiss and O. Lysne. FRoots: A Fault Tolerant and Topology-Flexible Routing Technique. *IEEE Transactions on Parallel and Distributed Systems*, 17(10):1136–1150, October 2006. ISSN 1045-9219. doi:10.1109/TPDS.2006.140.

[95] TOP500 Supercomputing Sites. TOP500 List, November 2010. URL http://www.top500.org.

[96] TOP500 Supercomputing Sites. Architecture share for 11/2010, November 2010. URL http://www.top500.org.

[97] TOP500 Supercomputing Sites. Interconnect family share for 11/2010, November 2010. URL http://www.top500.org.

[98] USENIX. The computer failure data repository (CFDR), April 2011. URL http://cfdr.usenix.org/.

[99] M. Valerio, L. Moser, and P. Melliar-Smith. Fault-tolerant Orthogonal Fat-trees as Interconnection Networks. In *IEEE First International Conference on Algorithms and Architectures for Parallel Processing (ICAPP 95)*, volume 2, pages 749–754, April 1995. doi:10.1109/ICAPP.1995.472263.

[100] L. G. Valiant and G. J. Brebner. Universal Schemes for Parallel Communication. In *STOC '81: Proceeding of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–277, NY, USA, 1981. doi:10.1145/800076.802479.

[101] S. Wamakulasuriya and T. Pinkston. A formal model of message blocking and deadlock resolution in interconnection networks. *IEEE Transactions on Parallel and Distributed Systems*, 11(3):212–229, March 2000. ISSN 1045-9219. doi:10.1109/71.841739.

[102] G. Zarza, D. Lugones, D. Franco, and E. Luque. A Multipath Fault-Tolerant Routing Method for High-Speed Interconnection Networks. In *Euro-Par 2009: Proceeding of the 4th International Euro-Par Conference on Parallel Processing*, volume 5704 of *LNCS*, pages 1078–1088, August 2009. doi:10.1007/978-3-642-03869-3_99.

[103] G. Zarza, D. Lugones, D. Franco, and E. Luque. A Multipath Routing Method for Tolerating Permanent and Non-Permanent Faults. In *15th Argentine Conference on Computer Science (CACIC)*, pages 251–261, October 2009.

[104] G. Zarza, D. Lugones, D. Franco, and E. Luque. Multipath Fault-Tolerant Routing Policies for High-Speed Interconnection Networks. In R. Doallo, M. Arenaz, and P. Gonzalez, editors, *Proceedings of the XX Spanish Conference on Parallelism*, pages 487–492, Sep 2009.

[105] G. Zarza, D. Lugones, D. Franco, and E. Luque. Deadlock Avoidance for Interconnection Networks with Multiple Dynamic Faults. In *18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP)*, pages 276–280, Feb 2010. doi:10.1109/PDP.2010.82.

[106] G. Zarza, D. Lugones, D. Franco, and E. Luque. FT-DRB: A Method for Tolerating Dynamic Faults in High-Speed Interconnection Networks. In *18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing (PDP)*, pages 77–84, Feb 2010. doi:10.1109/PDP.2010.65.

[107] G. Zarza, D. Lugones, D. Franco, and E. Luque. Fault-tolerant Routing for Multiple Permanent and Non-permanent Faults in HPC Systems. In *Proceedings of the*

International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), pages 144–150, Las Vegas, NV, USA, July 2010.

[108] G. Zarza, D. Lugones, D. Franco, and E. Luque. Non-blocking Adaptive Cycles: Deadlock Avoidance for Fault-tolerant Interconnection Networks. In *IEEE International Conference on Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS)*, pages 1–4, Sep 2010. doi:10.1109/CLUSTERWKSP.2010.5613085.

[109] G. Zarza, D. Lugones, D. Franco, and E. Luque. A Multipath Routing Method for Tolerating Permanent and Non-Permanent Faults. *Journal of Computer Science & Technology (JCS&T)*, 10:97–107, 2010.

[110] G. Zarza, D. Lugones, D. Franco, and E. Luque. A practical methodology for addressing the problem of resources utilization in modeling and simulation processes. Submitted to the journal Simulation Modelling Practice and Theory, July 2011.

[111] M. Zhou and M. P. Fanti, editors. *Deadlock Resolution in Computer-Integrated Systems*. CRC Press, Inc., Boca Raton, FL, USA, 2004. ISBN 0824753682.

Appendices

Appendix A

Complementary Results

In this appendix, we include additional results of the evaluation presented in chapter 5.

A.1 NAC Evaluation

A.1.1 Tornado with Variable Load

Average latency

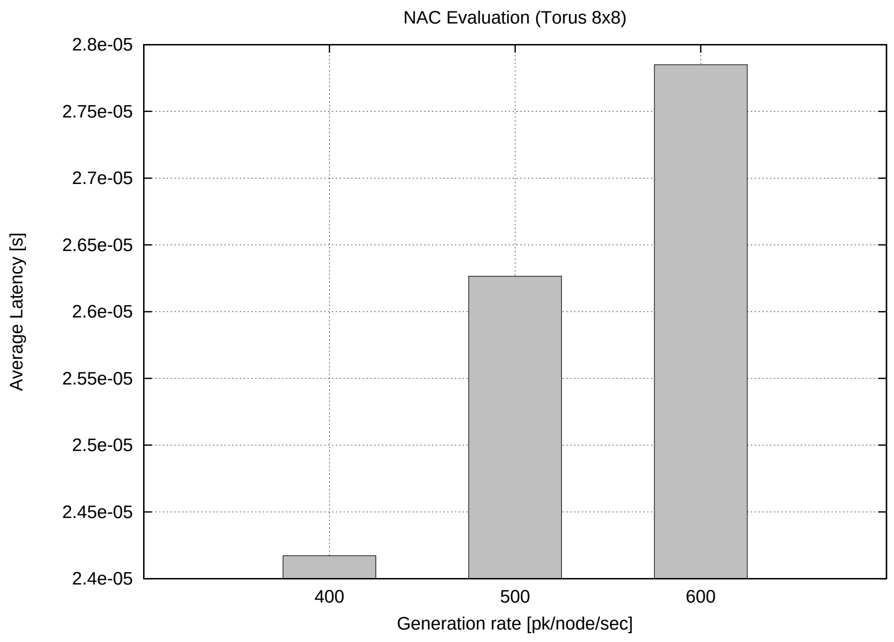
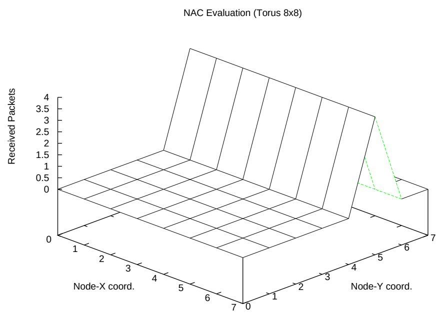
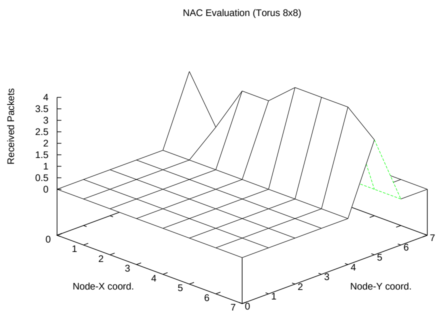


Figure A.1: Average latency for different generation rates.

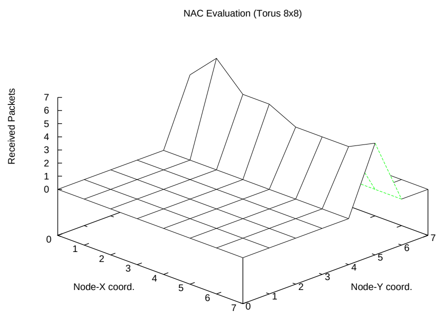
Surface-maps for generation rate 400 [pks/node/sec]



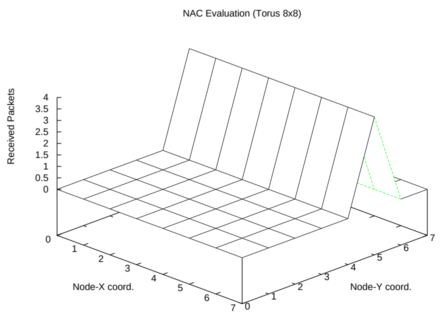
(a) Before deadlock



(b) During NAC treatment



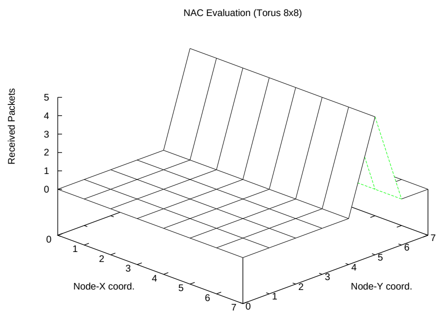
(c) After NAC treatment



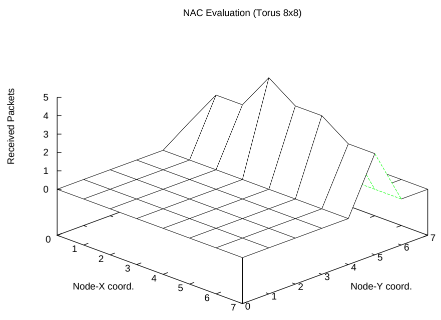
(d) After deadlock

Figure A.2: Packets received in four time slots. Tornado traffic generation rate = 400 [pks/node/sec].

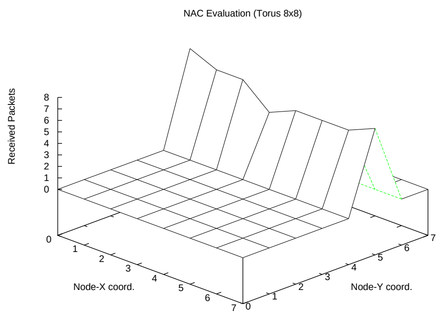
Surface-maps for generation rate 500 [pks/node/sec]



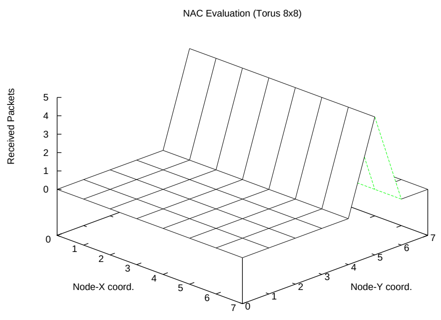
(a) Before deadlock



(b) During NAC treatment



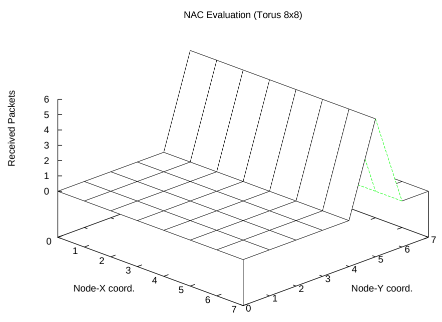
(c) After NAC treatment



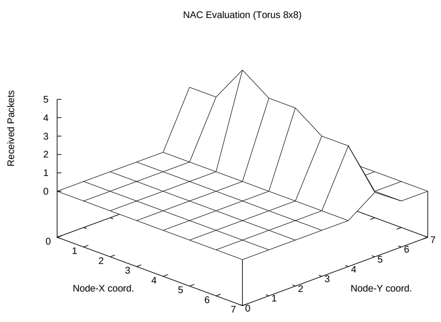
(d) After deadlock

Figure A.3: Packets received in four time slots. Tornado traffic generation rate = 500 [pks/node/sec].

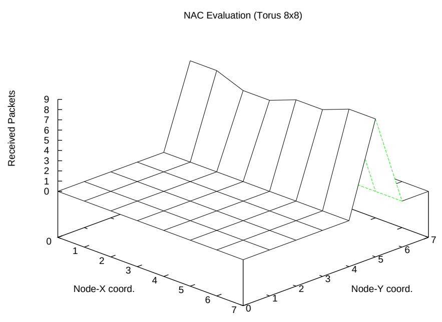
Surface-maps for generation rate 600 [pks/node/sec]



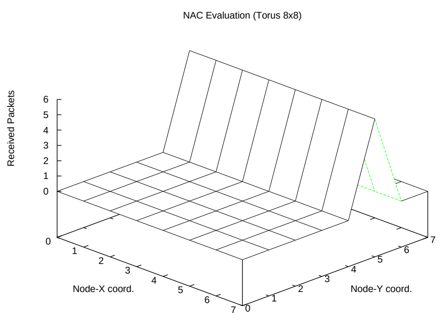
(a) Before deadlock



(b) During NAC treatment



(c) After NAC treatment



(d) After deadlock

Figure A.4: Packets received in four time slots. Tornado traffic generation rate = 600 [pks/node/sec].

A.1.2 Tornado with Additional Traffic

Average latency

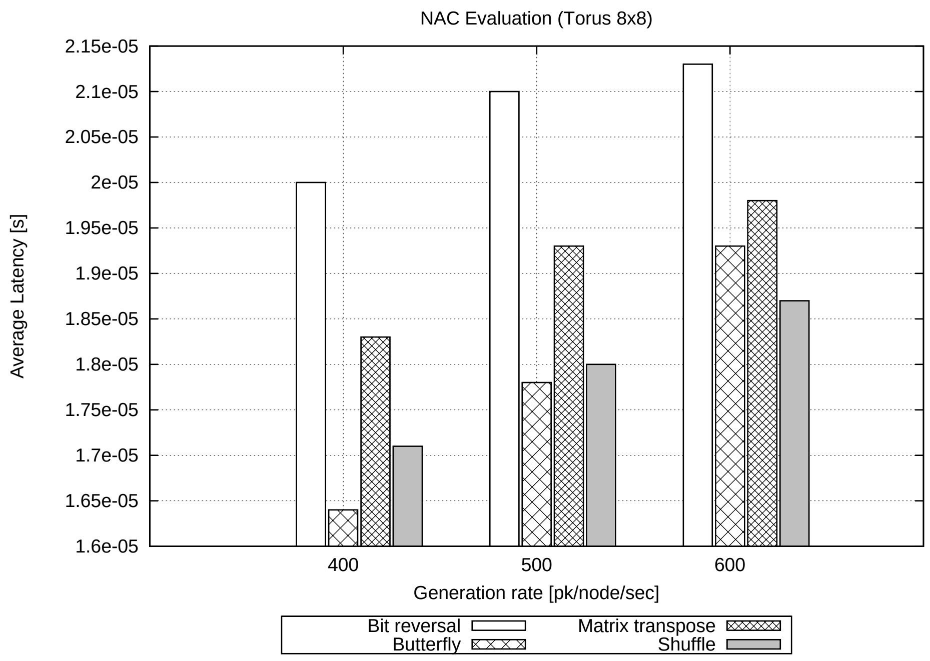
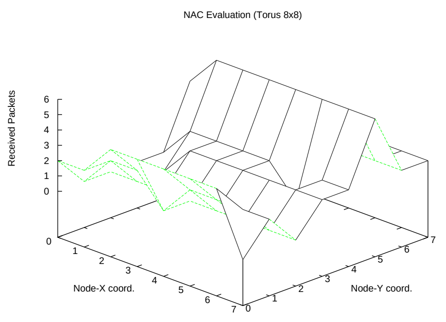
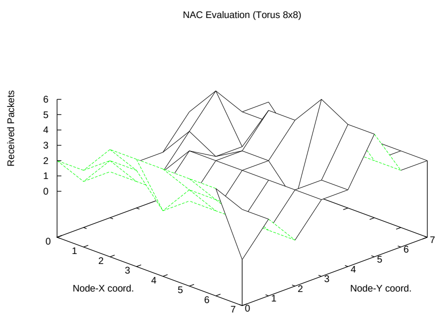


Figure A.5: Average latency with additional traffic patterns.

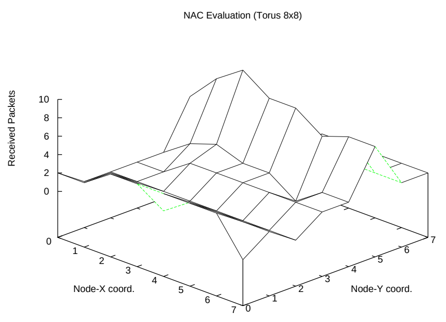
Surface-maps for generation rate 400 [pks/node/sec] with Bit reversal



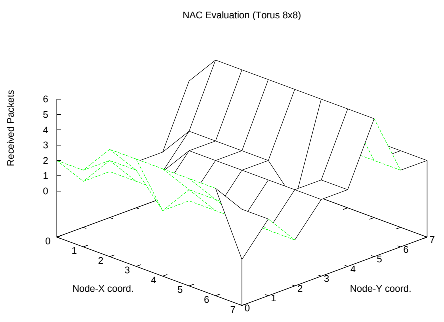
(a) Before deadlock



(b) During NAC treatment



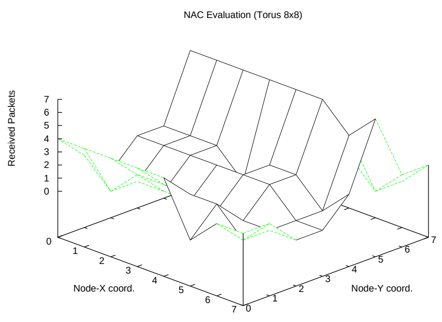
(c) After NAC treatment



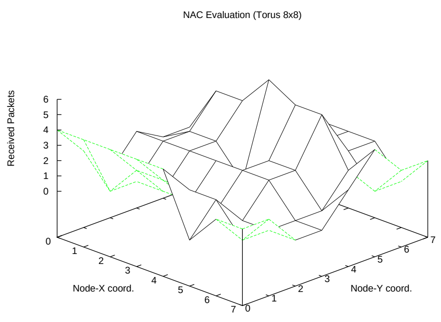
(d) After deadlock

Figure A.6: Packets received in four time slots. Tornado traffic generation rate = 400 [pks/node/sec]. Additional bit reversal traffic generation rate = 200 [pks/node/sec].

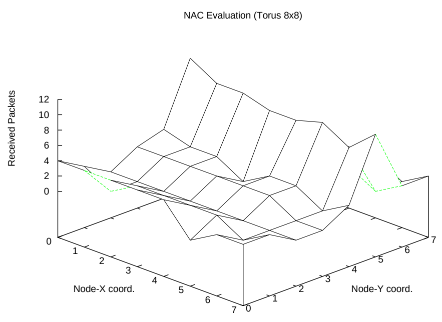
Surface-maps for generation rate 500 [pks/node/sec] with Bit reversal



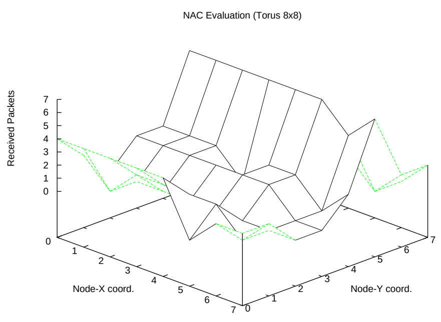
(a) Before deadlock



(b) During NAC treatment



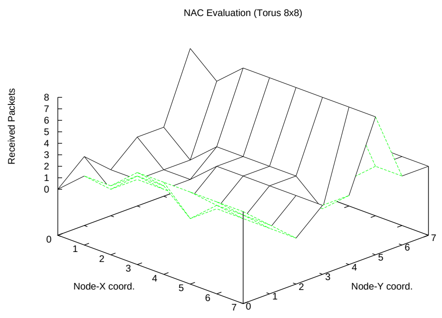
(c) After NAC treatment



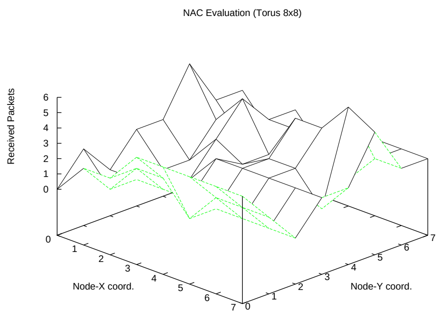
(d) After deadlock

Figure A.7: Packets received in four time slots. Tornado traffic generation rate = 500 [pks/node/sec]. Additional bit reversal traffic generation rate = 200 [pks/node/sec].

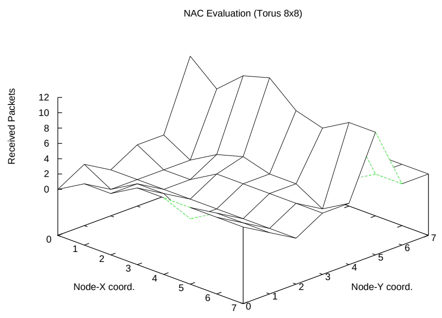
Surface-maps for generation rate 600 [pks/node/sec] with Bit reversal



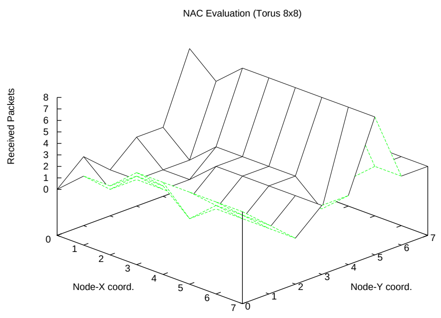
(a) Before deadlock



(b) During NAC treatment



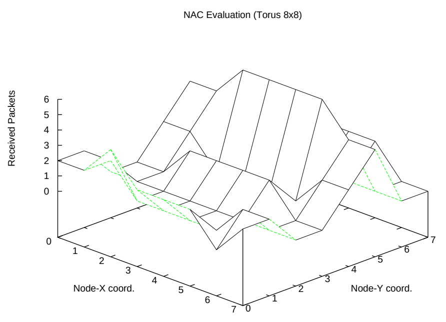
(c) After NAC treatment



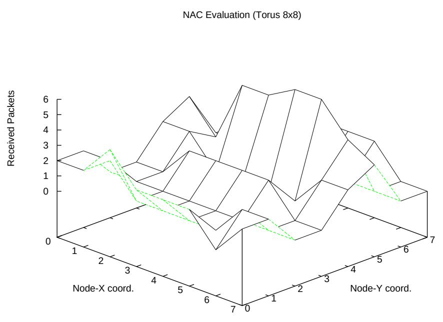
(d) After deadlock

Figure A.8: Packets received in four time slots. Tornado traffic generation rate = 600 [pks/node/sec]. Additional bit reversal traffic generation rate = 200 [pks/node/sec].

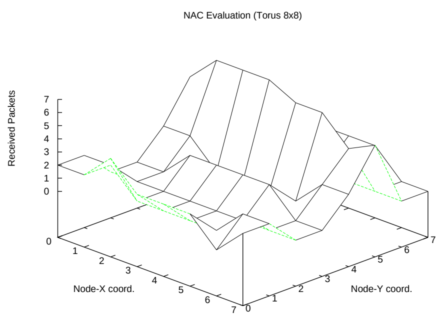
Surface-maps for generation rate 400 [pks/node/sec] with Butterfly



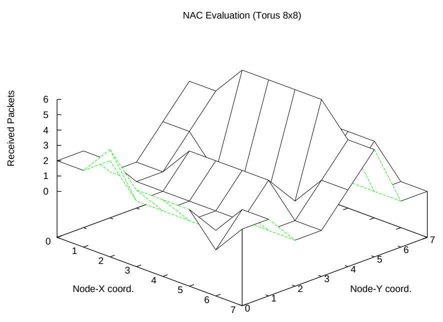
(a) Before deadlock



(b) During NAC treatment



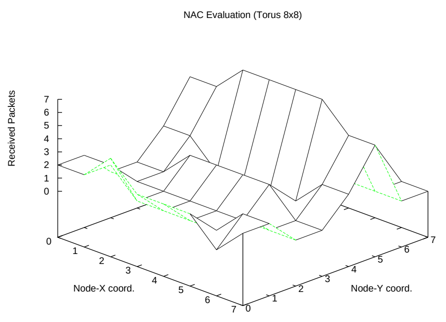
(c) After NAC treatment



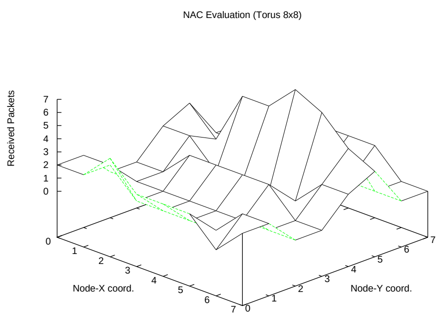
(d) After deadlock

Figure A.9: Packets received in four time slots. Tornado traffic generation rate = 400 [pks/node/sec]. Additional butterfly traffic generation rate = 200 [pks/node/sec].

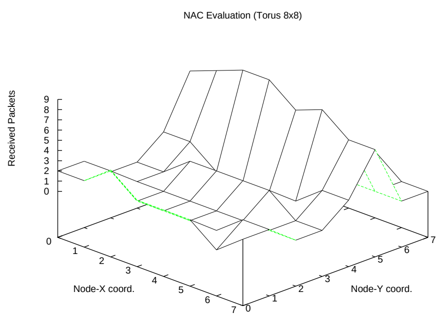
Surface-maps for generation rate 500 [pks/node/sec] with Butterfly



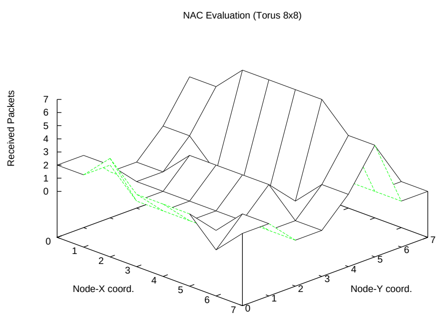
(a) Before deadlock



(b) During NAC treatment



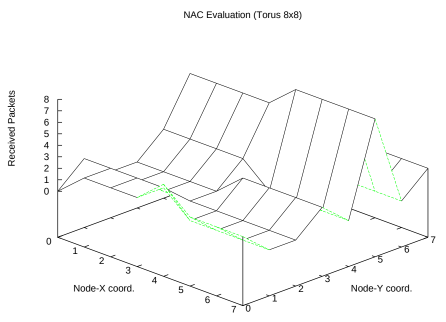
(c) After NAC treatment



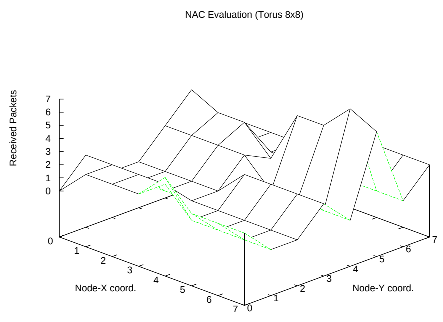
(d) After deadlock

Figure A.10: Packets received in four time slots. Tornado traffic generation rate = 500 [pks/node/sec]. Additional butterfly traffic generation rate = 200 [pks/node/sec].

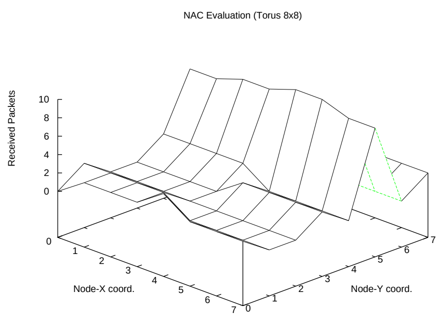
Surface-maps for generation rate 600 [pks/node/sec] with Butterfly



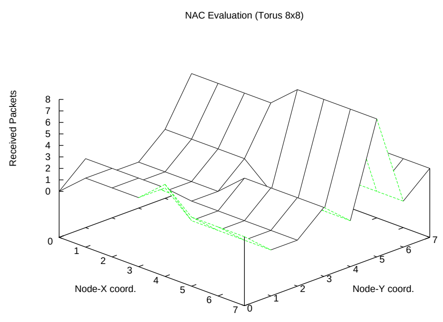
(a) Before deadlock



(b) During NAC treatment



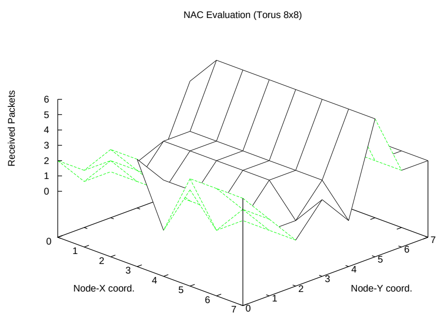
(c) After NAC treatment



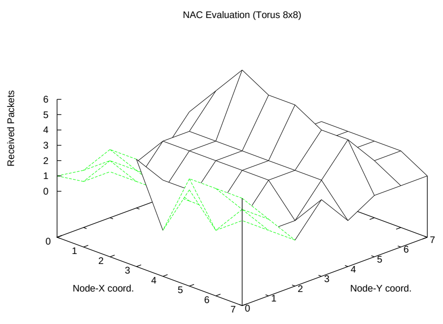
(d) After deadlock

Figure A.11: Packets received in four time slots. Tornado traffic generation rate = 600 [pks/node/sec]. Additional butterfly traffic generation rate = 200 [pks/node/sec].

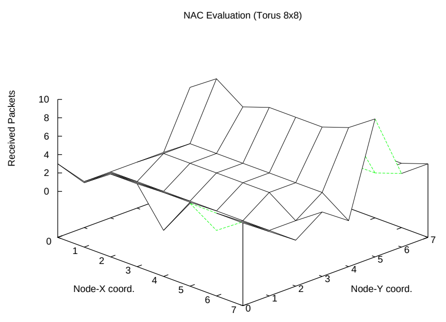
Surface-maps for generation rate 400 [pks/node/sec] with Matrix Transpose



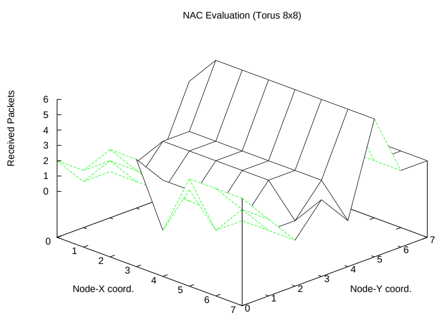
(a) Before deadlock



(b) During NAC treatment



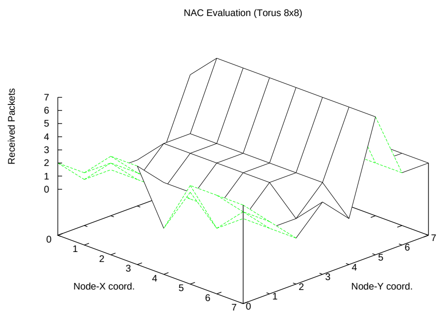
(c) After NAC treatment



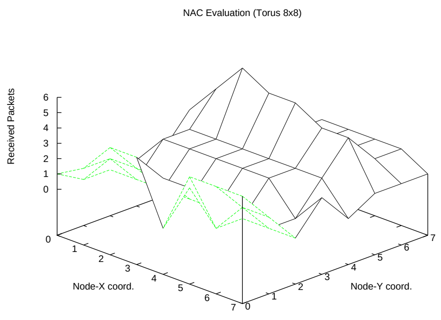
(d) After deadlock

Figure A.12: Packets received in four time slots. Tornado traffic generation rate = 400 [pks/node/sec]. Additional matrix transpose traffic generation rate = 200 [pks/node/sec].

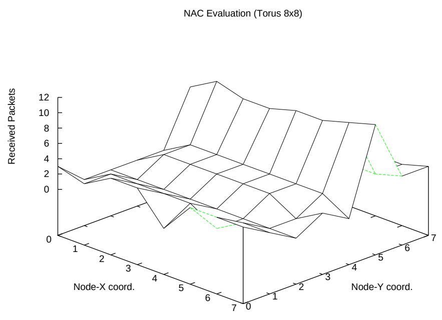
Surface-maps for generation rate 500 [pks/node/sec] with Matrix Transpose



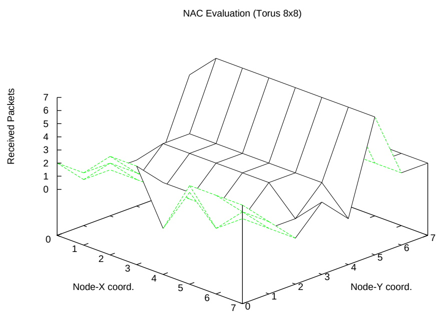
(a) Before deadlock



(b) During NAC treatment



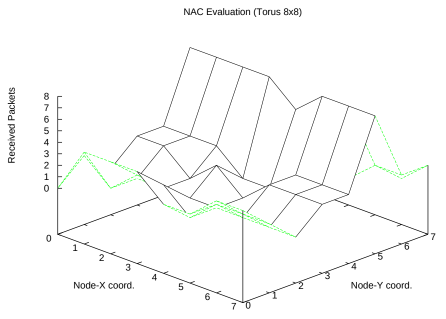
(c) After NAC treatment



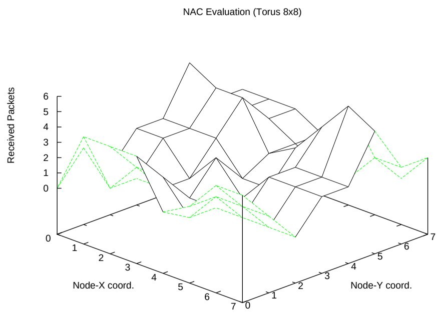
(d) After deadlock

Figure A.13: Packets received in four time slots. Tornado traffic generation rate = 500 [pks/node/sec]. Additional matrix transpose traffic generation rate = 200 [pks/node/sec].

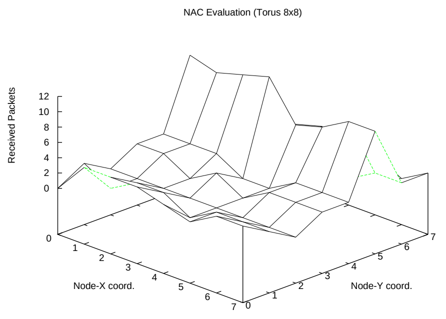
Surface-maps for generation rate 600 [pks/node/sec] with Matrix Transpose



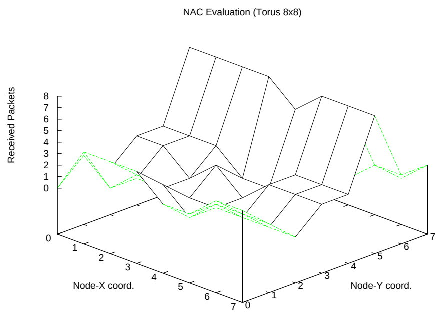
(a) Before deadlock



(b) During NAC treatment



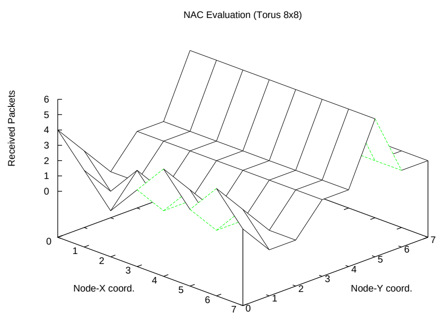
(c) After NAC treatment



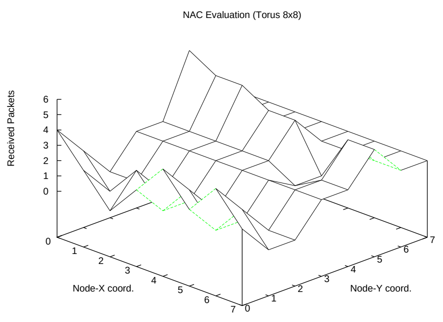
(d) After deadlock

Figure A.14: Packets received in four time slots. Tornado traffic generation rate = 600 [pks/node/sec]. Additional matrix transpose traffic generation rate = 200 [pks/node/sec].

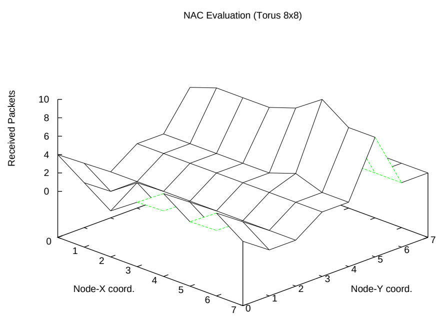
Surface-maps for generation rate 400 [pks/node/sec] with Perfect Shuffle



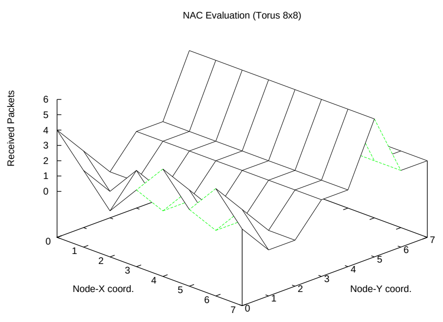
(a) Before deadlock



(b) During NAC treatment



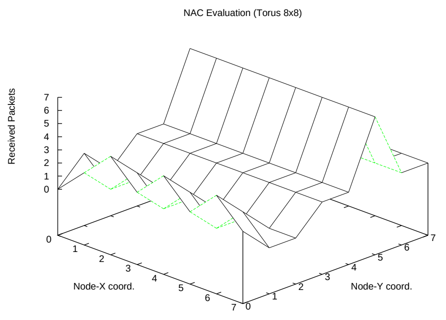
(c) After NAC treatment



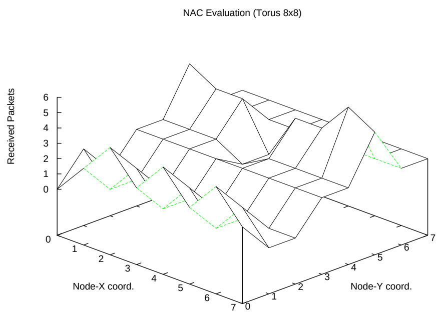
(d) After deadlock

Figure A.15: Packets received in four time slots. Tornado traffic generation rate = 400 [pks/node/sec]. Additional perfect shuffle traffic generation rate = 200 [pks/node/sec].

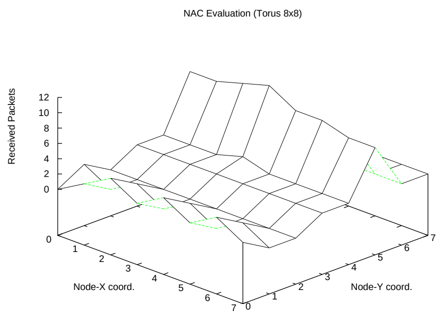
Surface-maps for generation rate 500 [pks/node/sec] with Perfect Shuffle



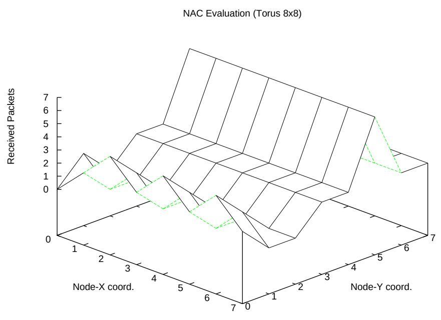
(a) Before deadlock



(b) During NAC treatment



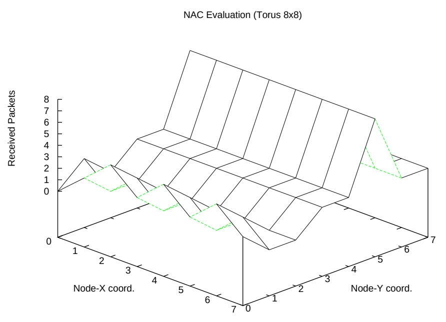
(c) After NAC treatment



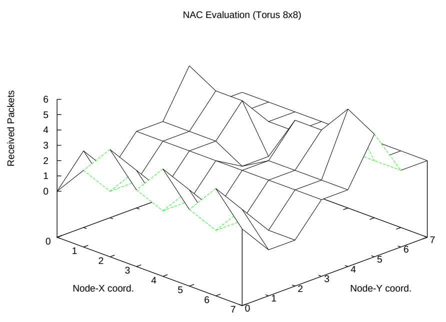
(d) After deadlock

Figure A.16: Packets received in four time slots. Tornado traffic generation rate = 500 [pks/node/sec]. Additional perfect shuffle traffic generation rate = 200 [pks/node/sec].

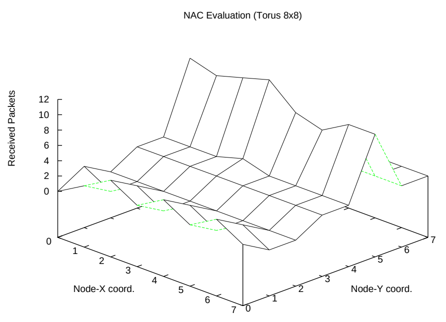
Surface-maps for generation rate 600 [pks/node/sec] with Perfect Shuffle



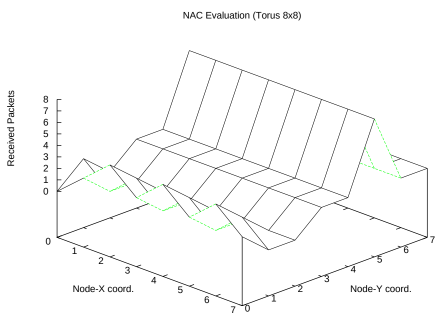
(a) Before deadlock



(b) During NAC treatment



(c) After NAC treatment



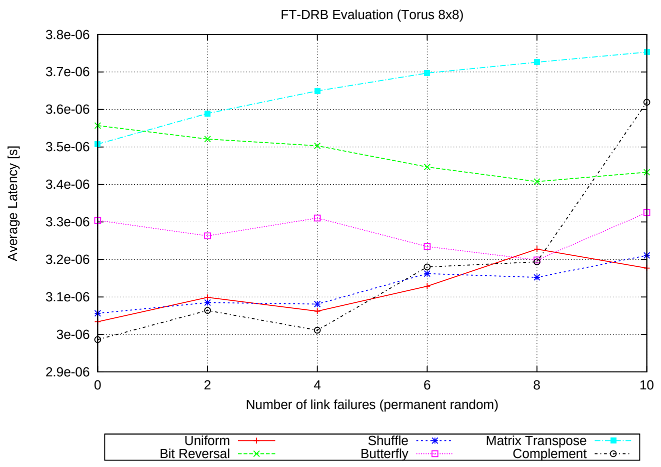
(d) After deadlock

Figure A.17: Packets received in four time slots. Tornado traffic generation rate = 600 [pks/node/sec]. Additional perfect shuffle traffic generation rate = 200 [pks/node/sec].

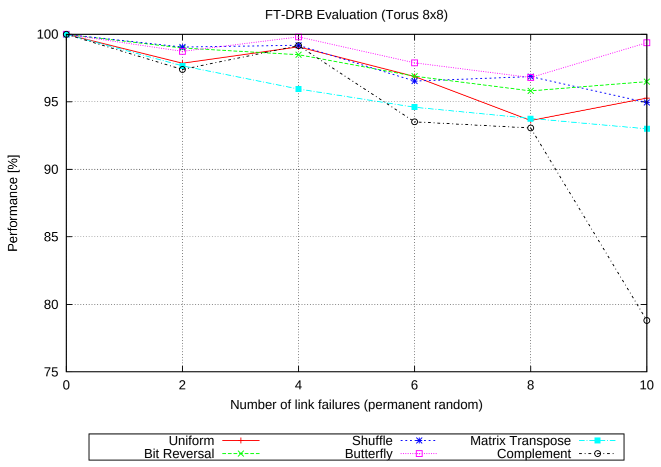
A.2 FT-DRB Evaluation

A.2.1 Permanent Failures with Synthetic Traffic

Torus 8x8. Random start time and duration.



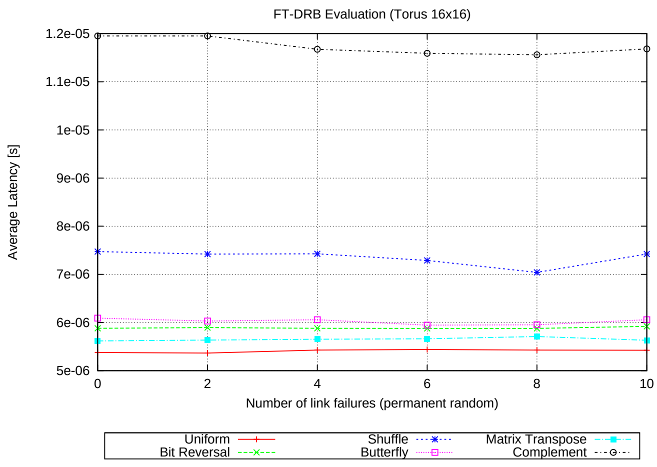
(a) Absolute values



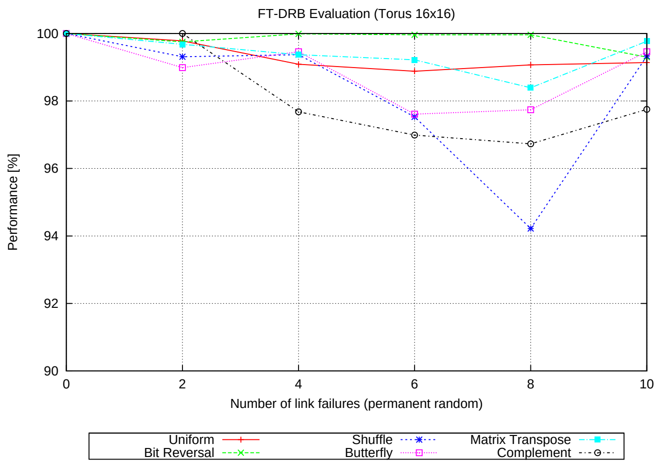
(b) Percentages

Figure A.18: Evaluation results of random permanent link failures in Torus 8x8.

Torus 16x16. Random start time and duration.



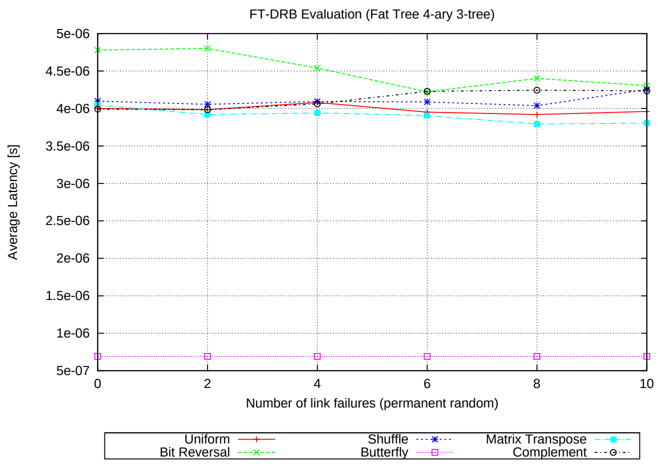
(a) Absolute values



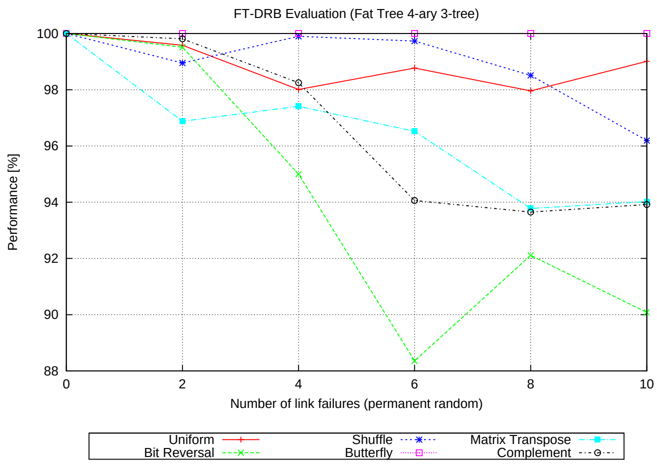
(b) Percentages

Figure A.19: Evaluation results of random permanent link failures in Torus 16x16.

Fat-tree 4-are 3-tree. Random start time and duration.



(a) Absolute values



(b) Percentages

Figure A.20: Evaluation results of random permanent link failures in Fat-tree.