Islamic University of Gaza
Deanery of Higher Studies
Faculty of Engineering
Computer Engineering Department

Congestion Management in Interconnection Networks

By

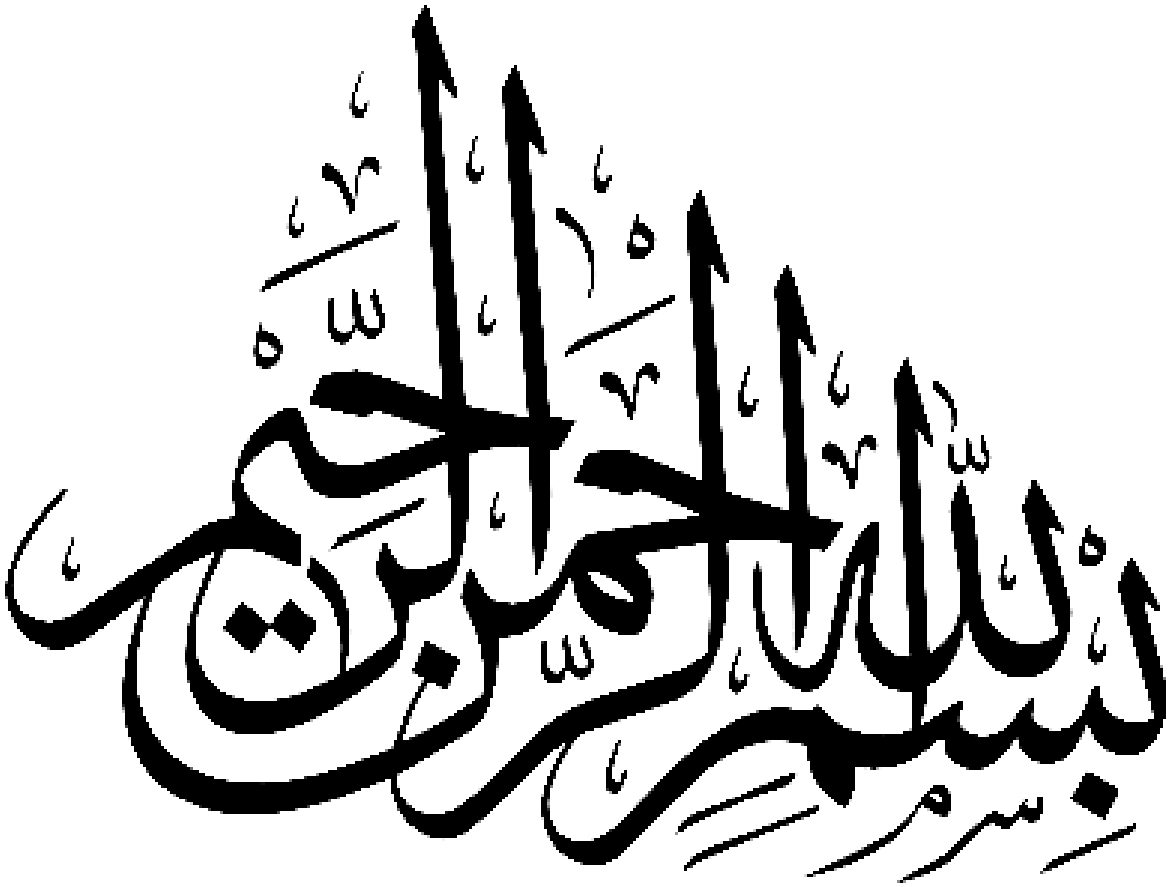
Eng. Raed Mohammed Bolbol

Supervisor

Prof. Mohammad A. Mikki

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

1433H (2012)


بسم الله الرحمن الرحيم

بسم الله الرحمن الرحيم

الجامعة الإسلامية – غزة
The Islamic University - Gaza

عمادة الدراسات العليا | هاتف داخلي: 1150

الرقم Ref: ج س غ/35/2012/10/01

التاريخ Date:

نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة عمادة الدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحث/ رائد محمد ابراهيم بليل لنيل درجة الماجستير في كلية الهندسة قسم هندسة الحاسوب وموضوعها:

Congestion Management in Interconnection Networks

وبعد المناقشة التي تمت اليوم الاثنين 15 ذو القعدة 1433هـ، الموافق 2012/10/01م الساعة الثانية عشرة ظهرًا، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

أ.د. محمد أمين مكي	مشرفاً ورئيساً	
د. أيمن أحمد أبو سمرة	مناقشاً داخلياً	
د. يوسف نبيل أبو شعبان	مناقشاً خارجيًا	

وبعد المداولة أوصت اللجنة بمنح الباحث درجة الماجستير في كلية الهندسة/ قسم هندسة الحاسوب.

واللجنة إذ تمنحه هذه الدرجة فإنها توصيه بتقوى الله ولزوم طاعته وأن يسخر علمه في خدمة دينه ووطنه.

والله ولي التوفيق،،،

عميد الدراسات العليا

أ.د. فؤاد علي العاجز

P.O. Box 108, Rimal, Gaza, Palestine fax: +970 (8) 286 0800 فاكس Tel: +970 (8) 286 0700 ص.ب 108 الرمال، غزة، فلسطين هاتف

Table of Contents

List Of Figures

List of Tables

List of Codes

Dedication

- ***To my beloved mother…***
- ***To my beloved father…***
- ***To my wife…***
- ***To my children…***
- ***To my friends…***

Acknowledgements

Praise is to Allah, the Almighty for having guided me at every stage of my life.

This thesis is the result of years of work whereby I have been accompanied and supported by many people. It is wonderful that I now have the opportunity to express my gratitude to all of them.

This work would not have been possible without the constant encouragement and support I received from prof. Mohammad Mikki, my advisor and mentor. I would like to express my deep and sincere gratitude to him. His understanding and personal guidance have provided a good basis for the present thesis.

I also extend my thanks to Dr. Aiman Abu samra and Dr. Youssef Abu Shaban the members of the thesis discussion committee.

Also, I would like to take this opportunity to express my profound gratitude to my beloved family – my mother, father and my wife- without whom I would ever have been able to achieve so much. I especially wish to express my love for my mother, my father, my brothers and my wife, they did not only endure my manifold activities but also provided inspiration and support for my inclination to perfectionism. Only they know how much I am indebted to them.

Last, but certainly not least, I want to thank my friends, for their moral support during this study.

ABSTRACT

Interconnection networks (INs) are becoming more and more powerful and hence place increasingly higher demands on the networks that interconnect their processors or processing nodes. Many of the applications running on such systems, especially embedded systems applications, with increasing application demands, high-performance networks are the hearts of these systems. These applications play an essential role in such networks. Congestion management remains yet a critical problem for interconnection networks. When traffic load is unfairly distributed across the network, some resources could be idle while others are quite congested. Imbalance of communications can produce network congestion. If congestion is not efficiently controlled, it is possible that these resources reach the saturation. This leads to a rise in message latency. In addition, global system performance is degraded due to congestion and this performance degradation is quickly propagated to the entire network. Therefore, a high speed routing scheme that minimizes congestion and avoids hot-spot areas should be included in the network components.

This thesis presents a Load-Balanced Distributed Adaptive Routing (LBDAR) algorithm that is used to manage congestion in interconnection networks. LBDAR algorithm aims to achieve high throughput, while delivering packets with low latency. LBDAR is adaptive routing mechanism that balances the communication traffic over the interconnection network routers and switches. This communication balancing is based on load-control and path expansion in order to maintain low message latency values. Experimentation shows that LBDAR achieves significant performance improvement over other algorithms used to manage congestion in INs.

Keywords: , Interconnection Networks, Congestion Managements, Load Balancing, adaptive routing.

إدارة الإزدحام في الشبكات المترابطة

م.رائد محمد بلبل

ملخص الرسالة

تعد الشبكات المترابطة (Interconnection Networks) أداء الوصل بين المعالجات والذاكرة في الحاسبات المتوازية وتتألف عادة من خطوط اتصال وعناصر متخصصة لنقل المعلومات كالمبدلات (Switches) أو الموجهات (Routers) وبفضل هذه الشبكات وتقنياتها التي تتطور بشكل مستمر فقد تمكن المصممون من بناء الحاسبات المتوازية والتي من خلالها يمكن تنفيذ برامج وتطبيقات صناعية وعلمية معقده ومن هنا جاء الاهتمام بهذه النوع من الشبكات ومع زيادة هذه التطبيقات المختلفة والتي تلعب دور أساسي في عمل هذه الشبكات حيث يتم تنفيذ التطبيق الواحد على أكثر من جهاز الامر الذي يسبب تنافس على مصادر الشبكة ويزيد الازدحام عليها ونتيجة لذلك يصبح بعض هذه المصادر مزدحمة ومصادر خاملة لا حمل عليها بسب عدم وجود أليه لإدارة المصادر في الشبكة ولذلك فان إدارة الازدحام في الشبكات الربط لا تزال موضوع بحث في الشبكات المترابطة نسعى من خلال هذه البحث إلى تصميم أليه يمكن من خلالها أن نقلل من مشكلة الازدحام وذلك من خلال تصميم خوارزميه جديدة تعتمد على توزيع الحمل في الشبكة وباستخدام اختيار المسار المناسب بالاعتماد على اقل قيمة لزمن الوصول من المصدر إلى الهدف .

في هذه الأطروحة تقدم الباحث بطريقه جديدة (LBDAR) لإدارة الازدحام في شبكات المترابطة ,تهدف إلى الوصول إلى إنتاجية عالية في الشبكة بأقل وقت زمني خلال عملية الاتصال بين المصدر والهدف ,تتميز هذه الطريقة بالتكيفيه والمرونة لإدارة الازدحام في الشبكة , لقد بينت نتائج البحث بشكل كبير إلى التحسين الذي تم الحصول عليه في إدارة الازدحام في الشبكات الربط باستخدام هذه الطريقة إذا ما قارناها بالطرق المستخدمة الأخرى .

Chapter 1
Introduction

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 [1]. 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 consist of thousands of components, including processing nodes, memory banks, disks, and other peripherals [2]. In this context, the interconnection network (IN) 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 to operate large coherent entities.

Clearly, the performance of current systems is closely related to the dependability and robustness of the congestion mechanisms on which they rely. Unfortunately, as explained

above, the steady increase in complexity and number of components of interconnection networks significantly increases congestion rates. Questions arise from the analysis of this situation such as: how do congestions affect communication systems? What kinds of congestions appear on real systems? Are those systems able to maintain their operation and performance standards in spite of congestion occurrences? If they are not, what should the solution be? What are the best options to achieve congestion management? The mere posing of these questions highlights the importance of congestion management, and the need to address this problem in current high-performance computing systems.

This thesis addresses the above questions related to congestion management of INs. It develops a distributed and adaptive congestion management protocol that solves the congestion problem in interconnection networks.

The rest of Chapter 1 describes in some detail the interconnection networks.

1.1 What are Interconnection Networks?

Interconnection networks are currently being used for many different applications. Examples of INs range from internal buses in very large-scale integration (VLSI) circuits to wide area computer networks, among others. Other examples include backplane buses and system area networks; telephone switches; internal networks for asynchronous transfer mode (ATM) and Internet Protocol (IP) switches; processor/memory interconnects for vector supercomputers; interconnection networks for multicomputer and distributed shared-memory multiprocessors; clusters of workstations and personal computers; local area networks; metropolitan area networks; wide area computer networks; and networks for industrial applications. Additionally, the number of applications requiring interconnection networks is continuously growing. For example, an integral control system for a car requires a network connecting several microprocessors and devices. The performance of most digital systems today is limited by their communication or interconnection, not by their logic or memory. In a high-end system today, most of the power is used to drive wires, and most of the clock cycle is spent on wire delay, not gate delay. As technology improves, memories and processors become smaller, faster, and less expensive. The speed of light, however, remains unchanged. The pin density and wiring density that govern interconnections between systems components are scaling at a slower rate than the components themselves. Also, the frequency of communication between components is lagging far beyond the clock rates of modern processors. These factors are combined to make interconnection the key factor in the success of future digital systems. Interconnection networks are emerging as a nearly universal solution to the system level communication problems for modern digital systems. Originally developed for the demanding communication requirements

of multicomputer, interconnection networks are beginning to replace buses as the standard system-level interconnection. They are also replacing dedicated wiring in special-purpose systems as designers discover that routing packets is both faster and more economical than routing wires [3].

1.2 Interconnection Networks Domains

Interconnection networks are designed for the use at different levels within and across computer systems to meet the operational demands of various application areas—high-performance computing, storage I/O, cluster/workgroup/enterprise systems, internetworking, and so on. Depending on the number of devices to be connected and their proximity, we can group interconnection networks into four major networking domains [4]:

- On-chip networks (OCNs)—Also referred to as network-on-chip (NoC), this type of network is used for interconnecting micro architecture functional units, register files, caches, compute tiles, processor and IP cores within chips or multichip modules.
- System/storage area networks (SANs)—This type of network is used for interprocessor and processor-memory interconnections within multiprocessor and multicomputer systems, and also for the connection of storage and I/O components within server and data center environments. Typically, several hundreds of such devices can be connected, although some supercomputer SANs support the interconnection of many thousands of devices.
- Local area networks (LANs)—This type of network is used for interconnecting autonomous computer systems distributed across a machine room or throughout a building or campus environment. Interconnecting PCs in a cluster is a prime example. Originally, LANs connected only up to a hundred devices, but with bridging, LANs can now connect up to a few thousand devices.
- Wide area networks (WANs)—also called long-haul networks, WANs connect computer systems distributed across the globe, which requires internetworking support. WANs connect many millions of computers over distance scales of many thousands of kilometers. ATM is an example of a WAN.

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. 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 [3].

1.3 Load-Balancing in Networking

Load balancing has been of interest in the networking community over the last few years. As the line rates continue to increase, load balancing plays a key role in building high-speed interconnection networks within Internet routers [5] and also guaranteeing high performance in the network backbone. In Internet routers, the worst-case throughput largely determines the capacity of the router [6] since the worst-case throughput effectively determines the maximum load that the fabric is guaranteed to handle. Load-balancing has proved to be a useful technique to ensure high guaranteed throughput delivered by the router fabric. Load-balanced switches were first introduced as Birkhoff-von Neumann (BN) switches by Chang et al [7]. These switches consist of two back-to-back crossbar stages. Similar to Valiant's load balancing scheme, the first stage load balances the incoming traffic pattern by spreading traffic uniformly and at random among the first stage's output ports. The second stage then delivers traffic to the actual destination port. Based on the BN switch, a lot of work has been done to incorporate efficient load balancing within Internet switches [8, 9]. Recently, Keslassy et. Al. [9] showed that using a fully connected graph topology with Valiant's two phase routing ensures optimal guaranteed throughput. From the point of view of worst-case throughput, this is the optimal load balancing scheme. However, this scheme does not optimize either the throughput on benign traffic patterns or the latency of packet delivery. The load balancing routing algorithm developed by the author of this thesis aims to achieve high throughput, while delivering packets with low latency.

1.4 Problem Statement

Increasing number of communication nodes interacting to transmit a packet from the source node towards a destination can lead to traffic unbalance, due to poor packets transmission strategies and inefficient mechanisms to prevent overflow of network resources capabilities.

Traffic unbalance can introduce network situations where the mentioned goals of interconnection network may not be fulfilled. For example, a routing algorithm is in charge of selecting the best routes to transmit a message over the network, but even when there may be many possible alternative paths to transmit those messages, the routing algorithm may not make proper decisions and thus cause situation where a lot of messages are being sent through some particular nodes in the network, causing a congestion situation or hotspot when other

portion of the network has enough resources to handle the traffic being locked due to poor decisions [10].

Congestion of packets in transit is also a known issue in interconnection networks. This situation appears when there are shared resources in the interconnection network, such as intermediate routers and links, and saturation can be reached if the situation is not controlled properly and in time. When traffic is not being properly handled by the network, then all messages start racing to obtain those resources. This race increases messages transit time, producing high latency values in the network in general and thus reducing the overall system performance [11]. The implication of this kind of congestion in networks where dropping of packets is not allowed is even more critical. In addition, parallel computers run this kind of networks. One solution given to this congestion problem is to combine different topological approaches, and to improve other layers of network protocol stack, such as congestion control and routing, it is considered as an affordable technique.

1.5 Thesis Motivation

Interconnection networks pay a fundamental role in many scenarios in current scientific and industrial applications, to solve practical problems and to increase human knowledge. Since interconnection networks are critical components of high-performance computing systems, congestion in network devices has terrible impact on the system. As few as a single device congests has the deadly potential of halting the entire computing system, blocking any running application. Carefully designed interconnection network routing algorithms are essential in the process of optimal utilization of communication resources, adapting to situations presented at every stage of processing, and improving the overall performance perception of the system. The routing method employed by a network determines the path taken by a packet from source terminal node to a destination terminal node. Networks with high path diversity offer many alternative paths between a source and destination. Oblivious routing algorithms choose between these paths based solely on the identity of the source and destination of the message while adaptive algorithms may base their decision on the state of the network.

A good routing algorithm makes its route selection in a manner that exploits locality to provide low latency and high throughput on benign traffic.

This thesis is intended to solve the problem of interconnection network congestion by using both distributed routing and adaptive routing. Distributed routing removes a single point of failure of the routing algorithm while adaptive routing uses a current status of congestion to

make better routing decisions. In addition, the developed routing algorithm intends to minimize routing overhead and packet delay.

1.6 Thesis Objectives

The main goal of this thesis is to design, implement and evaluate load-balance routing policy capable of serving interconnection networks. We address this problem by designing an adaptive distributed multipath routing strategy. Other objectives of the thesis include the following:

- Conduct a study on the congestion management theory, in order to analyze the potential application of these concepts to the field of interconnection networks.
- 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 congestion problem in interconnection networks.
- Improve overall system latency.
- Perform proper traffic load distribution among all communication resources.
- Avoid unnecessary hotspot situations in the network.
- keep global latency values low.
- Study and analyze specific features of an interconnection network that affect performance.
- Study dynamic features of interconnection networks using specific computing models, aimed to identify the problems during the normal operations of the network (under traffic) and their major causes.
- Study parallel applications patterns impact on network´s behavior, and how to establish a relationship between these applications and the routing mechanisms to improve performance.

1.7 Outline of the Thesis

The outline of the remaining chapters of the thesis is as follows. Chapter 2 presents thesis background and related work. It describes in detail interconnection networks fundamentals. The basis of an interconnection network is explained, and a general classification of existing topologies and their features are presented. Then, congestion problem is introduced as well as some approaches of how to solve the congestion problem. Chapter 3 presents our original Load-Balanced Distributed Adaptive Routing Algorithm (LBDAL). This chapter explains in detail the distributed adaptive routing strategy that we designed for treating load balancing in interconnection networks and for managing congestion in INs. Chapter 4 presents the evaluation and results of the proposed algorithm. This chapter describes the test scenarios and

provides the experimental results. Finally, chapter 5 concludes the paper. It concludes the thesis and presents the future work and congestion management techniques. The list of references and one appendix complete the document of this thesis.

Chapter 2

Background and Related Work

In this chapter we present some basic concepts in interconnection networks to make it easier to follow the discussion later. We introduce a general description of interconnection networks in Section 2.1, which is followed by an overview about the different techniques of congestion management in Section 2.2. In Section 2.3 we discuss some of the research methods common in the field including the method used in this thesis.

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 [13]. 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 [3]. Currently, interconnection networks are essential for systems where efficient communication technologies have a direct influence in the overall performance of the system.

These parallel systems can be classified into two main groups [14], [15]: 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 [14]. 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 [16], and represents 16.8% of the TOP500 systems [14]. These systems were the first using high-speed interconnection networks.

Many different implementations of interconnection networks exist. A bus provides a very simple interconnection network in which all terminals communicate through a common bus as shown in (a). The crossbar is another implementation of an interconnection network that is widely used in which all the terminals are connected to each other through a single switch ((b)).

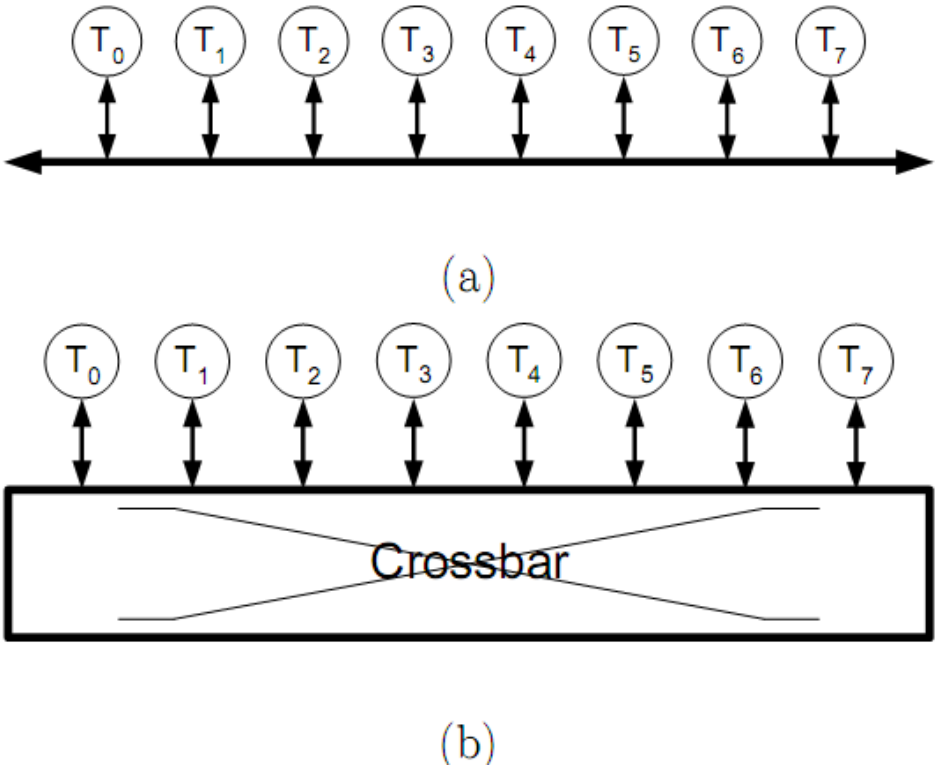


Figure 2.1: Different implementations of an interconnection network. An 8-nodenetwork realized using (a) a bus, (b) a crossbar switch

2.2 Interconnection Networks Design Factors

Interconnection networks are critical to the performance of modern parallel computer systems, and there are many factors that influence the choice of an appropriate interconnection network for a particular computer. These factors include the following, according to [17]:

• **Performance requirements**. Process executing in different processors synchronize and communicate through the interconnection network, hence performance of the network is vital. Common parameters to measure performance are latency and throughput, where latency is the time elapsed between a message is generated in the source node until it reaches its destination and is delivered. Network throughput is the maximum amount of information delivered by the network per time unit.

• **Scalability**. Through increasing the number of processors, the overall bandwidth of the system (network, I/O and CPU) increases as well. If this is not true, bottlenecks in performance appear.

• **Incremental expandability**. Parallel computer systems are purchased by parts, and the system must allow the possibility of adding more resources (network components) when needed.

• **Partitionability**. Due to the nature of applications running in parallel systems, one execution must not interfere with other execution taking place at the same time, in terms of traffic generated. This is also true when security matters arise.

• **Simplicity**. Designs made simple, help the user to understand the architecture and then configure the whole system to exploit performance at maximum level.

• **Distance span**. This factor is related to the distance between communication nodes. Distance variation can cause problems of electromagnetic noise, coupling, etc.

• **Physical constraints**. Packaging of components is related to distance span, and influence in other factors such as temperatures, latency due to distance variations, etc.

• **Reliability and repairability**. An interconnection network should be able to communicate in a reliable way. Besides, it should have a modular design and allow hot upgrades and repairs.

• **Expected workload**. Network design should be robust to adapt to different traffic conditions and always offer reasonable performance.

• **Cost constraints**. Trade-off must always be made between all the factors mentioned and the cost of an interconnection network.

• **Power consumption**. Energy-efficient interconnection networks have become and imperative matter in HPC world. This efficiency is related tightly to cost constraint. Just to mention some examples, there are costs of power and cooling in all communication devices, and as organizations look forward to reduce energy use in their data centers, computing and communications systems can be a focus of the efforts to reduce power consumption.

2.3 Interconnection Networks Topologies

The topology of the network refers to how nodes and channels are arranged in the interconnection network. This represents the path where messages must traverse during communication. Topology is modeled as a graph, where vertexes represent nodes and an edge represents a link between a pair of vertexes. A set of parameters defines an interconnection network, as defined below:

• **Degree**: Number of edges connecting one node with other nodes.

• **Diameter**: Maximum of all minimum distances between all pairs of nodes.

• **Average distance**: Average distance between any pair of nodes in the network.

• **Bisection width**: Minimum links that must be eliminated to divide the network equally into two pieces.

• **Symmetry**: A network is considered symmetric if all the nodes in the network look alike from every other node.

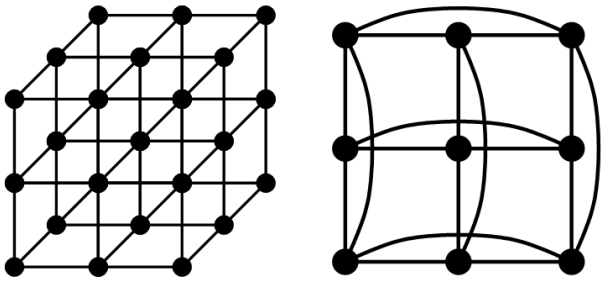
• **Expandability**: A network is expandable if the procedure of expanding its size is done in a simple way.

A topology describes the conceptual layout of a network and how the nodes in the network are connected. It is the topology that ultimately determines the set of paths a packet can take between two nodes in a network, while the routing algorithm selects a subset of these paths according to certain criteria. The choice of topology depends on the application domain, cost, and packaging restrictions. Topologies are divided into direct and indirect networks [2, 3]. Direct networks refer to topologies where every node in the network contains both a switch and a host, i.e., each node produces, consumes, and routes network traffic. This way each node can reach every other node in the network by going through one or more nodes. In an indirect network, hosts and switches are separate units connected to each other through a link. A switch can be connected to other switches, other hosts, or a combination of hosts and switches. A direct network can always be converted to an indirect network by separating the host and the switch, and then connect them through a link. This makes the distinction somewhat mute [3]. It does, however, make more sense if we view a direct network as a network where every switch is associated with a host, and an indirect network as a network where only a subset of the switches is associated with a host. Using this definition, two of the most common direct networks are the torus and the mesh, more formally called the k-ary n-cube and the k-ary n-mesh. Both these belong to a class of topologies referred to as strictly orthogonal. In a strictly orthogonal topology every node has at least one link in each dimension, a fact that makes routing in these networks simple [2].

The mesh is described by the number of dimensions and the number of nodes in each dimension. Figure 2.2 (a) shows a 3-ary 3-mesh, formally we say that an n-dimensional mesh has $k0 \times k1 \times \ldots \times kn-2 \times kn-1$ nodes with ki nodes in dimension i, where $ki \geq 2$ and $0 \leq i \leq n$ [2]. For a 3-dimensional mesh with a different number of nodes in each dimension we explicitly give the size of each dimension, e.g. a 2, 3, 4-ary 3-mesh is a 3-dimensional mesh with two nodes in dimension y, three nodes in dimension z, and four nodes in dimension x [3]. The mesh has been used in multi-computers such as the MIT M-Machine, which uses a 3-dimensional mesh [18].

The torus, also referred to as a k-ary n-cube, is shown in Figure 2.2 (b). In the k-ary n-cube all nodes have the same number of connections and the number of nodes in each dimension are the same. To get the same number of connections to each node, wrap-over connections are added at the edge nodes. A k-ary n-cube has kn nodes. Figure 2.2 (b) shows a 3-ary 2-cube.The torus has been used in machines such as the Cray T3D [19] and the BlueGene/L [20].

Indirect networks also come in several forms, but the most popular category is multistage interconnection networks (MINs). MINs consist of a set of stages where hosts are only connected to the terminal stage, and connected to each other through a set of switching stages. The Clos network described by Charles Clos at Bell Labs in 1952 [21] is an example of a MIN. Figure 2.3 (a) shows a 16-way Clos network, the number 16 refers to the number of hosts connected to the terminal stage. A Clos network consists of three or more stages where only the switches in the first and last stage are connected to hosts. The switches in the other stages are only connected to switches in the stages before and after themselves.

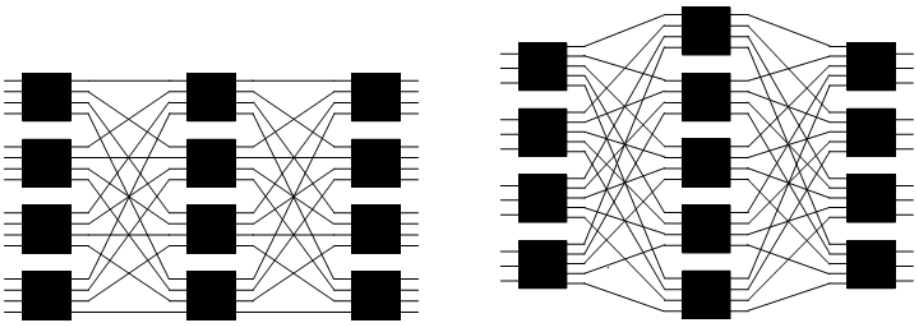


(a) A 3-ary 3-mesh. (b) A 3-ary 2-cube.

Figure **2.**2: Direct network topologies

Clos networks are categorized as either rearrangeable non-blocking or strictly non-blocking. A rearrangeable non-blocking network is a network where it is possible to find a way from a free input to a free output by rearranging the existing connections, while in a strict non-blocking network it is possible to find a way from a free input to a free output without modifying the existing connections. Figure 2.3 (a) shows a rearranging non-blocking Clos.

A strict non-blocking switch requires more switches in the center stage and fewer hosts connected to the terminal stage as shown in Figure 2.3 (b).



(a) A rearrangeable non-blocking 16 way Clos.

(b) A strict non-blocking 16 way Clos.

Figure **2.**3: Multistage interconnection network topologies

Clos networks have been used in the GF11 supercomputer from IBM [21] and have also been widely proposed as an architecture for ATM switches [22, 32, 24]. Other types of MINs are the Butterfly, the Benes, the Omega, and the Fat Tree [2]. Depending on the topology, MINs can be simple to route as deadlock cannot occur.

Another class of indirect networks in widespread use is irregular networks. An irregular network is a network without any well-defined structure, it consists of a set of randomly connected switches and hosts. Most local area networks are irregular networks, and so are most networks of workstations. Moreover, a regular network becomes irregular whenever a switch or link in the network fails. Irregular networks present the greatest challenge for routing as no assumptions can be made about the topology.

2.4 Classification of Interconnection Networks

A classification scheme of interconnection networks is shown in Figure 2.4, which categorizes the known interconnection networks into four major classes based primarily on network topology shared-medium networks, direct networks, indirect networks, and hybrid networks. For each class, the figure shows a hierarchy of subclasses, also indicating some real implementations for most of them [2].

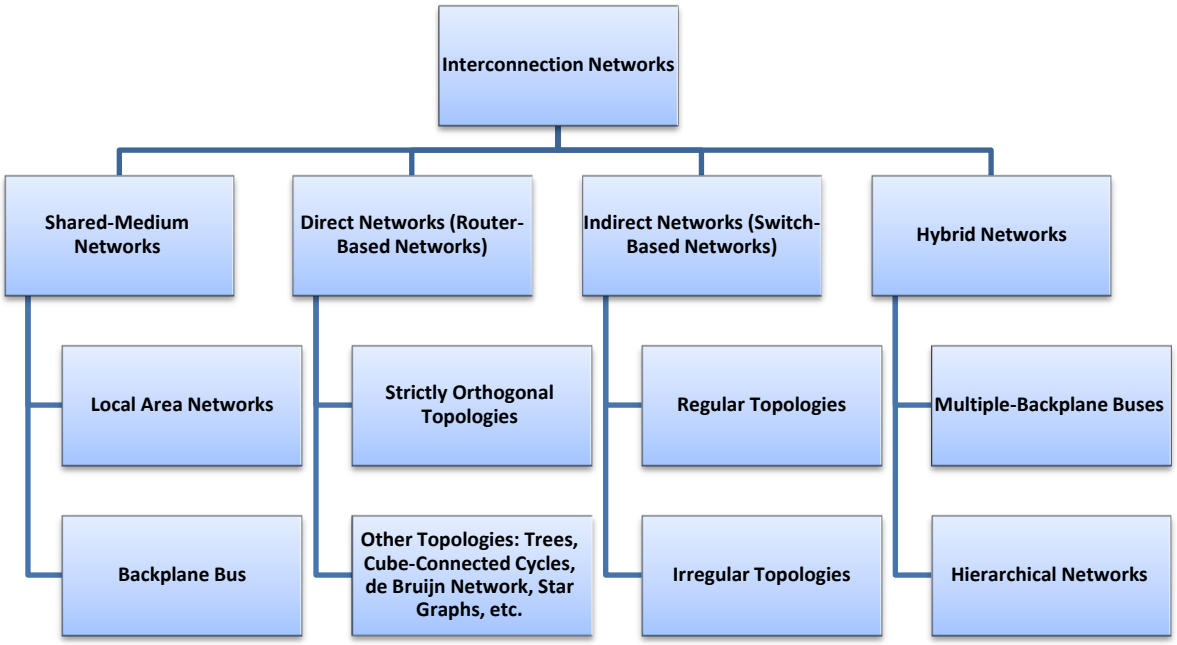
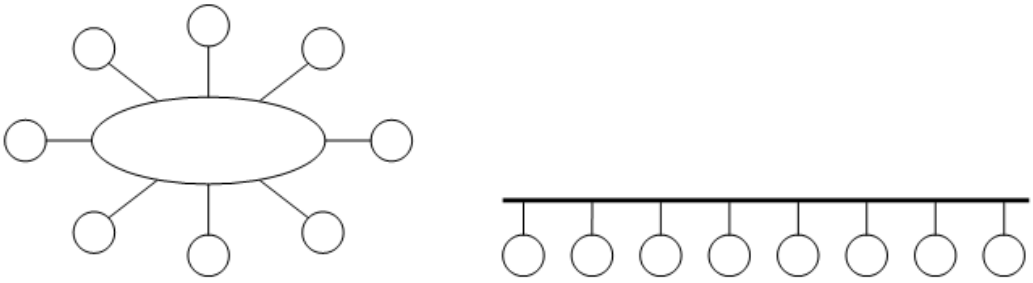


Figure 2.4: Classification of interconnection networks

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 copperwares 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. Some examples of these networks are shown in Figure 2.5.



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

Figure 2.5: Examples of shared-medium network topologies.

Direct Networks. These 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. [2], direct networks have been traditionally modeled by a graph G(N,C), where the vertices of the graph N represent the set of processing 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 has 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 Figure. 2.6.

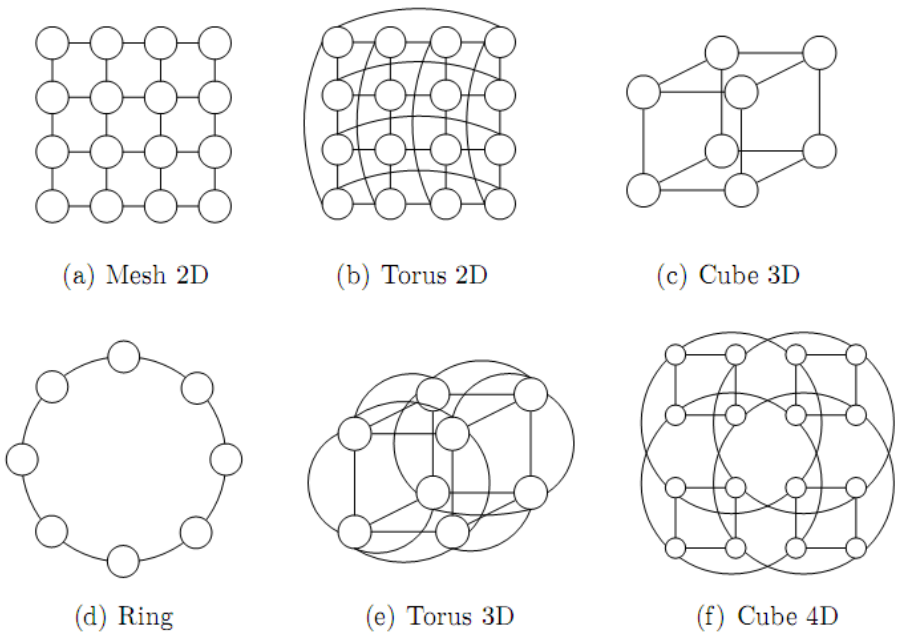


Figure 2.6: Examples of direct network topologies (orthogonal).

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 crossbars1and MINs2; 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 [2]. The classification and some examples of this kind of networks are shown in Figure 2.7.

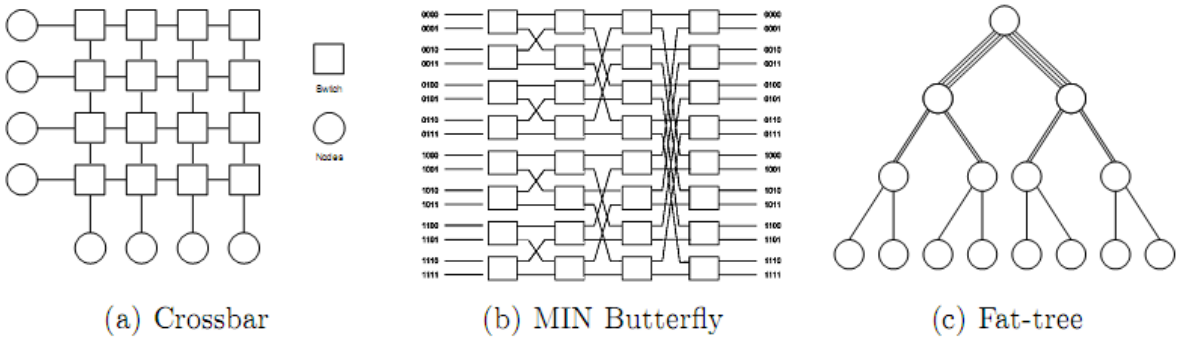


Figure 2.7: Examples of indirect network topologies

2.5 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 Dua to et al. [2]. The resulting classification is based on number of destinations, routing decisions, implementation and adaptivity. The classification is summarized in Figures 2.8 and 2.9.

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 a 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.

Some properties found in most routing algorithm are correctness, simplicity, robustness, stability, justice and optimization [26]. Correctness and simplicity implies the algorithm has to be computationally simple, because routing decisions must be made in short periods of time. Ability to adapt to topology changes because of nodes failure, is considered robustness. Convergence to a stable situation, if an adaptive technique is used, is considered stability. The property of justice is accomplished when access to resources, such as channels and links, is given under equal conditions to all demanding nodes. Optimization can be achieved in two ways; by minimizing global latency value of the interconnection network, or by global efficiency maximization. These two goals often contradict each other, because maximizing efficiency often means to insert more packets into the network, and this situation can lead to higher waiting times for packets, hence increasing latency, unless proper mechanism of congestion control are used.

Routing algorithms can be classified according to several criteria as expressed in [2]. They can be first classified by the number of destinations of a message, being unicast when there is only one destination and multicast when involves collective communications.

Routing algorithms can also be classified according to the place where routing decisions take place. If one central node is in charge of all decisions then is called centralized, source routing if decisions are taken at the source node before packet injection and distributed routing if decisions are performed while the message is traversing intermediate routers in the network. Multiphase is a concept that mixes both previous schemes. Implementation of routing algorithms can also be done in different ways. Most common approaches are the use of a

table to look up routes to be used (table lookup) or using hardware of software approaches according to a finite state machine. Both cases can fall into the categories of deterministic or adaptive.

Deterministic algorithms always choose the same path between a pair of source and destination. Adaptive algorithms take into consideration the status of the network at any time, to choose the best routes based on congestion situations, channel allocations, latency values, etc.

Adaptive routing algorithms can be categorized as a progressive or backtracking. Progressive always make decisions at each routing operation, while backtracking can go back and deallocate resources previously allocated.

At lower level, routing algorithms can be classified as according to their minimality as profitable or misrouting. Profitable only supply channel that brings the packet closer to its destination. Misrouting algorithms may supply channel that sends a packet away from its destination. At the lowest level can be classified according to the number of alternative paths as completely adaptive (also known as fully adaptive) or partially adaptive.

The performance of a routing algorithm depends on, in addition to the facts already discussed, the use of shortest-paths and a balanced use of network resources. Shortest-path routing means that we always select the path that gives the shortest number of hops between any source/destination pair, which reduces latency and improves throughput as we use the least amount of resources possible. By balanced, we mean that the routing algorithm evenly distributes traffic across the network when a uniformly distributed communications pattern is applied. Some algorithms, such as UpDown routing, are unbalanced since they rely on a spanning tree and the root of the spanning tree becomes a network hot-spot [27].

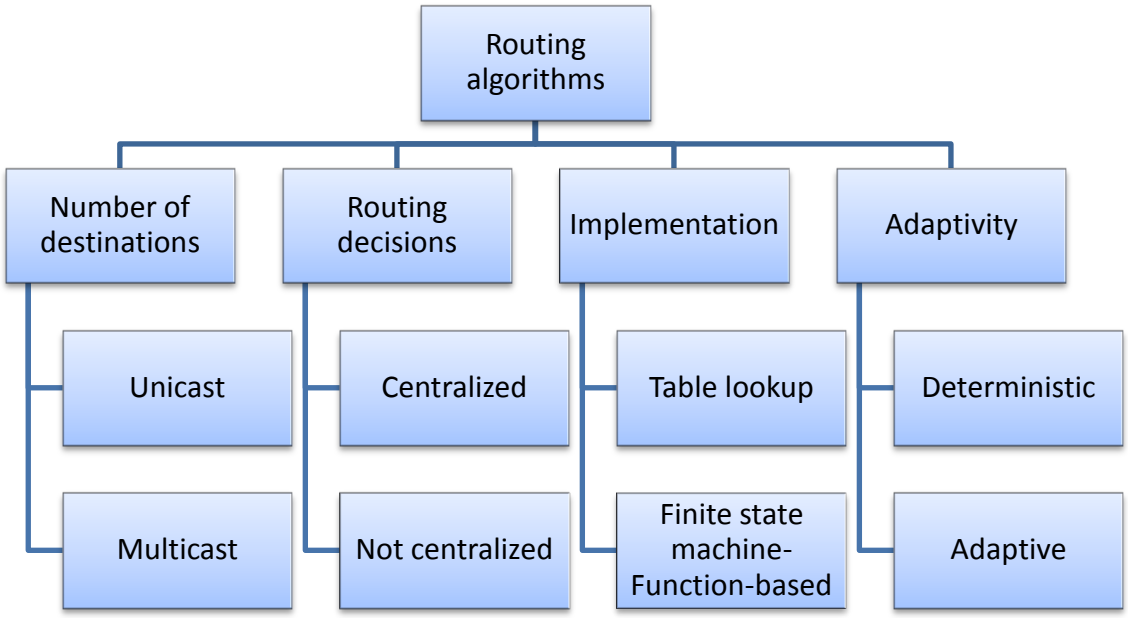


Figure 2.8: Classification of routing algorithms

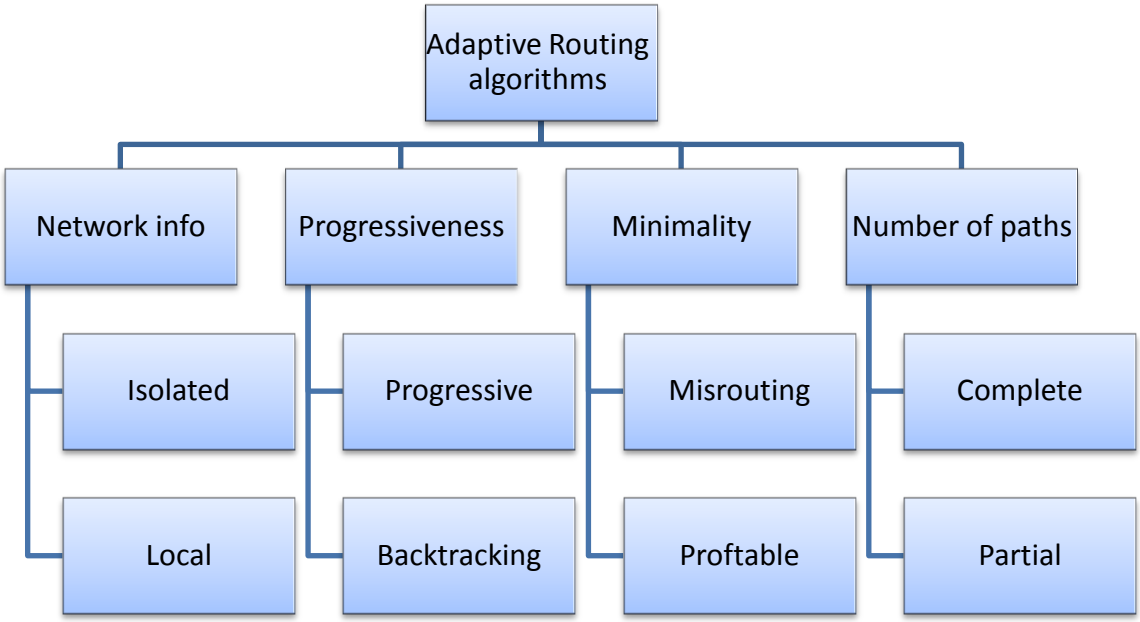


Figure 2.9: Classification of adaptive routing algorithms

2.6 Message Switching Techniques

Many services are involved at inter-process communications, and this service provides many layers that perform some specific task such as transference of bits streams, higher protocols communications between layers, packetization, compression an more. While not standardized, three layers are the most distinguished in the operations of the interconnection network: these are the physical layer, the switching layer and the routing layer. Physical layer identifies link-level communications to transfer messages and manage physical channel and other hardware components. Switching layer utilizes the physical layer protocols in order to

implement mechanisms for messages forwarding through the network. Routing decisions, selection of output channels and hence the path through the network, is the main goal of the routing layer. Actually, switching is the mechanism that removes data from an input channel and places it on an output channel [28]. At this moment, the most widely accepted techniques are Store-And-Forward, Virtual Cut-Through [29], and Wormhole [30].

Store and forward (SAF) [2] switching is a traditional switching technique used in early multi-computer networks and local area networks. It is also used in most Ethernet switches. SAF switching works by first receiving a complete packet in a buffer, then examining the header, and finally forwarding the packet out on the correct link. This happens for each switch a packet passes through and it makes it possible to check for and remove corrupt packets at every hop. The drawback is that it increases latency since each packet must be completely received by the switch before it can be routed and forwarded to its destination link. Thus, the latency is proportional to the number of hops in a path. SAF switching also puts a constraint on the maximum packet length since a complete packet must be able to fit in the buffer space of switch, i.e. if we want to support larger packets we need larger buffers.

Virtual cut-through (VCT) [31] switching is a solution to the latency problem appearing in SAF switching. Instead of transmitting a complete packet, each packet is split into a number of smaller units called flits. The first flit contains the packet header and is used to make the routing decision, while the rest of the flits belonging to the packet follow the same path as the header flit through the network. If the header flit has to be buffered due to a busy output link the rest of the flits will be buffered behind it in the current buffer. In other words, the routing decision is made as soon as the header of the packet is received and if the necessary resources are available the rest of the packet is not buffered but forwarded directly to the destination link. If the necessary resources are busy the packets are buffered in the switch just as with SAF switching. Since we don't have to do any buffering when resources are available latency is reduced compared to SAF switching. But for the worst case where a packet is blocked at every switch VCT switching is equal to SAF switching with regards to latency. VCT switching still has the drawback that the maximum packet length depends on buffer size because we must be able to buffer a complete packet when a required resource is unavailable. Furthermore, per hop removal of corrupt packets is no longer possible because we do not receive the whole packet before it is forwarded. The InfiniBand Architecture [32] and the Advanced Switching Interconnect [33]use VCT switching.

Wormhole routing (WR) [34] removes the dependency between maximum packet length and buffer size found in SAF and VCT switching. As with VCT switching we use flits, but when the header flit is blocked it is buffered at the current switch together with some of the flits following it. When the buffer in the current switch is full the remaining flits are buffered in the switches along the established path all the way to the source. At the source the flow-control halts the transmission of data until the necessary resources are available. The buffered packet now spreads like a worm through the network thus the name Wormhole routing. WR yields the same resource gain as VCT, in addition it allows for unlimited packet size as we have removed the dependency between packet size and buffer size. Unfortunately, WR is more prone to deadlock than SAF and VCT switching as the distribution of a packet as a worm through the network grabs hold of more resources giving higher deadlock probability. Wormhole routing was used in the STC104Asynchronous Packet Switch [35].

0 compares the latency of VCT/WR and SAF switching. We see that the start of a packet might be entering node 3 before the end of the packet has left node 0 when we use VCT/WR. While with SAF switching we have only reached node 1.

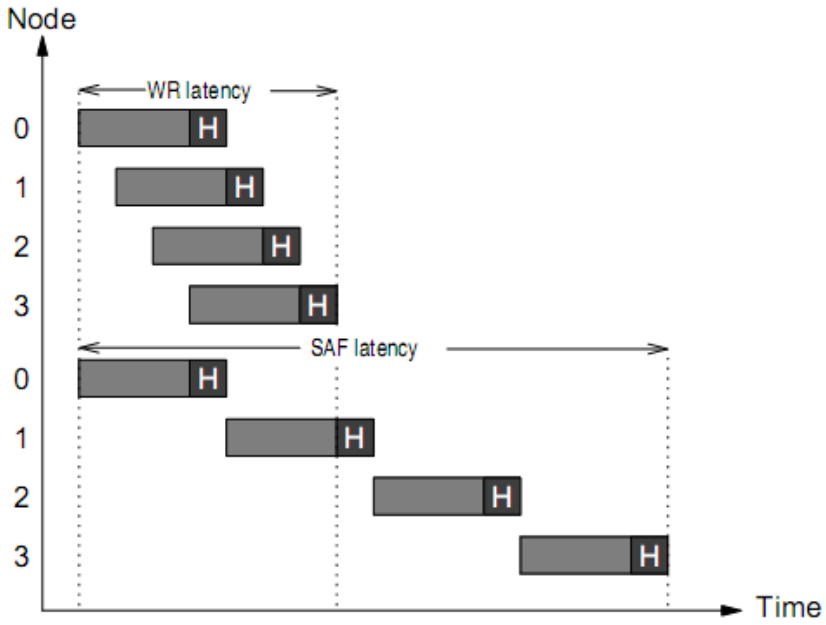


Figure 2.10: Latency for store and forward switching versus virtual cut-through switching and wormhole routing.

2.7 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 [2], [3]. The most commonly used schemes are Credit-based and On/O.

Credit-based Flow Control: This mechanism assign credits to each router in the network, in order to allow message transfer to other neighbors routers. Each transmission decrement the credit count and once the router runs out of credit, then it stops transmitting.

On/O Flow Control: In this scheme, possible buffer overflow is detected at the receiver node, and then it proceed with the notification of the situation to the source node to stop transmitting. When the receiver has enough space to resume communication, it sends a control message informing of this situation to the sender node.

2.8 Congestion Management

Congestion management is one of the most critical and challenging problems interconnect designers face today. Without suitable congestion management, network throughput may degrade dramatically when the network or part of it reaches saturation. Two major approaches exist to tackle congestion situations. These two approaches are preventive and reactive techniques. Each of these approaches has its own set of advantages and disadvantages for particular congestion situations explored.

Preventive techniques involve solutions based on the concept of increasing resources available in the system (over dimension) thus avoiding congestion by allowing communications to perform without interruption.

A necessary condition for this technique to work is that application behavior is to be known beforehand, so that resources to fulfill the task can be properly estimated. One disadvantage of this method is the inability to always determine a priori resources required for an application. Also, reservation of resources is not an optimal solution because it can lead to inefficient network usage and also cause congestion, caused by the process of search and reservation of resources itself, which is performed through packets injection within the network.

This solution, though simple, could be very difficult to implement due to current network features. Over dimensioning of resources is not a valid proposal based on communication devices cost, compared with the processing power. Another point to take into consideration, and now more institutions are paying attention to this subject, is power consumption. Deliberately including more resources can lead to highest power consumption, and

in super-computers where there are thousands of processing nodes and no less communication devices, this can be a serious item to study.

Reactive techniques embrace mechanism to use available resources properly, adapting to adverse traffic using no more than the minimum required resources. To accomplish this task, a process of analyzing network status during communications is performed, and when network situation is approaching congestion state, notification procedures are executed in order to alert injecting nodes about the situation, and let them know that some strategy to control this problem has to be done. So, three main phases can be extracted to perform the congestion control:

• Network monitoring and detection of congestion situations

• Notification about the recently found congestion to injecting nodes

• Execute corrective procedures in order to effectively control the congestion situation

During monitoring and detection phase, some features of the interconnection network are analyzed to determine congestion situation somewhere in the network. Latency values and flow speed implicit in congestion (also known as back-pressure) are metric analyzed during this phase. Decisions can be made globally or locally. Globally wide decisions are more accurate than local decisions, but introduces higher overhead into the network as well, because of synchronization between all nodes to make those good decisions. Notification phase can be implemented with variations according to the notification scheme used. One alternative is when there is no notification at all, in this case the node under congestion tries to solve the problem by itself, adjusting internal configuration parameters as internal buffers, ports, etc. Other alternatives to notify about congestion only to closer nodes in the network, neighbors, which will try to solve the congestion situation. Notification to sender nodes is another approach, based on the idea that sender nodes caused the congestion and they should be responsible for taking proper actions to regulate traffic injection, hence solving the congestion situation detected.

One last obvious approach could be notification to all nodes in the network about congestion in a particular point, expecting that every node receiving this notification takes actions to try to solve the situation unleashed previously. The problem with this technique is the overhead involved in traffic load, to communicate information about the congestion to all other nodes, and their response and actions taken to try o solve the situation.

Once congestion has been detected and notified, actions to control the situation must be executed. A technique that regulates traffic at the source node during congestion is known as Message Throttling [36], topping injection during the congestion, or slowing down new packets generation. This technique, even alleviating congestion, introduces high latency values into the network due to message generation regulation.

Another possibility is to try to organize and manage internal router buffers, re-ordering packets of different flows to avoid collision. The essence of this technique is to try to transmit packets that are not under congestion, but share resources with other that indeed are, as fast as possible. While congestion is reduced at he router applying this procedure, the problem is that congestion is not controlled at the injecting node, thus no giving a solution at the core of the problem [37],[38].

Congestion management based on path or load distribution alter original communication paths based on the status of the network, in order to avoid network areas under congestion. These techniques inject packets destined to a node through a series of alternative paths, to keep latency values low without reducing traffic load levels. Fair traffic load distribution is accomplished using this kind of algorithms because traffic is sent over idle or less burdened nodes in the network, thus avoiding congestion and as a consequence improving performance by avoiding congested spots in the network.

2.9 Related Work

Most designers today choose to reduce system cost and power consumption by reducing the number of network components. If we are to maintain a system's computational power, there are only two ways to reduce the number of network components: increasing the number of end nodes attached to each switch, or using more suitable fabric topology. However, each of these solutions leads to a higher link use level, thereby driving the network closer to its saturation point and increasing the congestion. Thus, congestion will become more common in future networks, and designers will have to implement some specific congestion management mechanism or network performance will suffer the corresponding degradation.

A lot of prior research has focused on congestion management techniques in the last years, some of them are :

2.9.1 Dimension Ordered Routing, DOR

The most published application of static or source-based routing algorithms within HPC systems is what is called Dimension Ordered Routing (DOR) [2] [3]. At the source, the node

pre-computes a set of preferred directions to route the packet within. For instance, if a source and destination pair exist within the same Z dimension, only differing within the X and Y dimensions, it would be preferred not to change coordinates within the Z dimension [2] [3]. Once the preferred directions have been calculated, this information is placed within the header of the data and it is directed into the corresponding X direction first. Once the packet has successfully reached the same X coordinate as the destination, it then routes in the Y direction (if it needs to). Following the Y direction, intermediate nodes then continue to route in the Z direction (again, if necessary) until it finally reaches the destination. So, in the case above, if the pre-computed preferred directions were [+X,-Y, 0], this packet would fully route in the positive X direction and then the negative Y direction last. In terms of all possible directions, this is the order that data is routed within the network:[+X,-X,+Y,-Y,+Z,-Z].Dimension Ordered Routing is proven to be free from message block [39] by balancing the load on virtual channels. The issue shown in [39] shows that by making this modification the structure of the network and the diversity of usable channels changes. This has shown to create non-uniform patterns. These virtual channels are only necessary on torus or wrap-around networks, so with regular mesh-type networks, the full benefits of DOR can be witnessed.

2.9.2 Direction Ordered Routing, DIR

Direction Ordered Routing (DIR) is very similar to that of Dimension Ordered Routing. The difference in these two resides in the order in which the routing occurs once the preferred directions have been calculated. These calculations are done just as they are with DOR, but instead of fully routing within the X direction before progressing to the Y direction and so on, DIR routes positively within all directions first, then routes negatively within the directions last [2] [3]. The order that data is routed within the network is as follows:[+X,+Y,+Z,-X,-Y,-Z]

2.9.3 Fully Adaptive: GOAL, GAL, CQR

The essence of the best algorithms is epitomized within the characteristics of fully adaptive routing algorithms for these networks [41]. By exploiting locality and the network's wrap-around feature, an efficient routing algorithm should be able to have the flexibility to misroute data to balance the traffic load. Fully adaptive algorithms do just this. The work that was done by Singh, et. al, [40] [41] [42], has collectively progressed into what seems like a solid attempt at finding a stable, reliable, and fully adaptive routing algorithm on tori-mesh networks for a diverse set of traffic patterns.

The first of the three algorithms, Globally Oblivious Adaptive Locally (GOAL), works very similarly to that of Minimal Adaptive with a few changes. First, it finds all possible routable

quadrants and scales the probability of selecting a particular quadrant based on how minimum it is with respect to a particular source/destination pair. Once those probabilities are assigned and a particular quadrant has been selected, routing is done minimally within it [40]. This algorithm uses a virtual-channels implementation (called *-channels) to ensure deadlock and cycle-free operation [40].

One problem with this algorithm, which was a similar problem, its inability to recognize global congestion. The answer to this comes through in the next algorithm that was developed by this same group, Globally Adaptive Load-Balanced (GAL).This algorithm is implemented by keeping a globally visible set of input queues for each node, called injection queues. Each node has as many injection queues as it has inputs, and when a packet begins its commute from source to destination, the injection queue on the destination node relative to the link the packet will be received on is increased. As with GOAL, GAL first determines the routable quadrant, then selects this quadrant based on distance (as did GOAL), but it also considers the injection queues. This way, nodes can keep a general idea of how much total traffic is being sent in any particular direction, and can attempt to balance that out globally [42]. Another versatile option for this algorithm to reduce its complexity is by subnetting the injection queues. By grouping multiple interfaces together with a single queue, thus reducing the number of queues to store globally, the authors showed that this technique had little impact on the overall performance of this algorithm [42]. Finally, a better meld of the global and local qualities that GAL and GOAL provided was integrated into a simple-to-implement and efficient protocol called Adaptive Channel Queue Routing (CQR) [41]. This routing algorithm uses local information in the form of output queues (as opposed to the GAL algorithm which implemented input injection queues which were globally accessible). CQR uses these output queues as estimators for global congestion, which was proven within the work of [41]. The work showed that even in the absence of global information, local information can provide good estimations of global congestion. also, relieving the algorithm from accessing and changing global information made it much simpler to implement (not to mention lower its overhead).

2.9.4 Valiant's Oblivious Routing Algorithms

Valiant's Algorithm came about in an attempt to balance loads within any particular topology [3]. The algorithm works by first selecting at random an intermediate node, x, and once successfully routing to that intermediate node, then routing from x to the destination. as it's stated, any arbitrary routing algorithm can be used to get data from source to the intermediate node and then from the intermediate node to the destination [3].

2.9.5 Regional explicit congestion notification (RECN)

RECN (Regional Explicit Congestion Notification)[43] is a new congestion management strategy that focuses on eliminating the main negative effect of congestion: the HOL blocking. In order to achieve it, RECN detects congestion and dynamically allocates separate buffers for each congested flow, assuming that packets from non-congested flows can be mixed in the same buffer without producing significant HOL blocking. RECN requires the use of a kind of deterministic routing that makes possible to address a particular network point from any other point in the network. In fact, RECN has been designed for PCI Express Advanced Switching (AS) [44,45], a technology that uses source routing. AS packet headers include a turnpool made up of 31 bits, that contains all the turns (offset from the incoming port to the outgoing port) for every switch in a route. Thus, a switch, by inspecting the appropriate turn pool bits, can know in advance if a packet that is coming through one of its incoming ports will pass through a particular network point. In order to separate congested and non-congested flows, RECN adds a set of additional queues at every input (ingress) and output (egress) port of a switch. These queues (referred to as Set Aside Queues or SAQs) are dynamically allocated and used to store packets passing through a congested point. To do this, RECN associates a CAM memory to each set of queues. The CAM contains all the control info required to identify the congested point and to manage the corresponding SAQ. In the aim of guaranteeing in order delivery, whenever a new SAQ is allocated, forwarding packets from that queue is disabled until the last packet of the standard queue (at the moment of the SAQ allocation) is forwarded. This is implemented by a simple pointer associated to the last packet in the standard queue and pointing to the blocked SAQ.Whenever an ingress or egress queue receives a packet and fills over a given threshold, a RECN notification is sent to the sender port indicating that an output port is congested. When congestion is detected at the egress side, the congested point is this egress port. In order to detect congestion at the ingress side, the standard queue is replaced by a set of detection queues. The detection queues are structured at the switch level: there are as many detection queues as output ports in the switch, and packets heading to a particular output port are directed to the associated detection queue. So, when a detection queue reaches a threshold, it means that the associated output port is congested. RECN notifications also include the routing information (a turnpool) to reach the congested output port from the notified port. Upon reception of a notification, each port maps a new SAQ and fills the corresponding CAM line with the received turnpool. From that moment, every incoming packet that will pass through the congested point (easily detected from the turnpool of the packet) will be stored in the newly allocated SAQ, thus eliminating the HOL blocking it may cause. If a SAQ becomes subsequently congested, a new notification will be sent upstream to some port that will react in

the same way, allocating a new SAQ, and so on. As the notifications go upstream, the included information indicating the route to the congestion point is updated accordingly, in such a way that growing sequences of turns (turnpools) are stored in the corresponding CAM lines. So, the congestion detection is quickly propagated through all the branches of a congestion tree. Apart from the SAQs allocated due to notifications, when congestion is detected at the ingress side, a SAQ is also allocated at this port, and the detection queue and the new allocated SAQ are swapped. RECN keeps track (with a control bit on each CAM line) of the network points that are a leaf of a congestion tree. Whenever a SAQ with the leaf bit set empties, the queue is deallocated and a notification packet is sent downstream, repeating the process until the root of the congestion tree is reached. Regarding flow control, RECN uses for each individual SAQ a level-based flow control (Xon/Xo). This mechanism is different from the credit-based flow control used for standard queues, that considers all the unused space of the port data memory available for each individual queue. Xon/Xo scheme guarantees that the number of packets in a SAQ will be always below a certain threshold.

2.9.6 Destination-based buffer management

Reference [46] proposes destination-based buffer management (DBBM), which uses a small number of queues in front of each link. The queue that will store an incoming packet is determined by the destination address of the packet. When the number of queues per link equals the number of final destinations, DBBM implements per-flow queueing; when it is smaller, a queue can be shared among packets heading to different destinations, thus introducing HOL blocking. The effectiveness of the scheme depends on whether, with a small number of queues per link, queue sharing among congested and non-congested flows will be infrequent or not. Certainly this depends on the (queue) mapping function, and on the traffic pattern.

2.9.7 InfiniBand congestion control

A reactive congestion management protocol for InfiniBand networks [InfiniBand] has been proposed in [46]. A switch detects congestion at one of its output ports when a new packet, p, increases the size of the corresponding output queue above a predefined threshold. The switch then sets the Forward Explicit Congestion Notification (FECN) bit in packet p. When the destination receives a packet with the FECN bit set, it responds back to the source of the packet with a Backwards Explicit Congestion Notification (BECN). Upon receiving a packet with the BECN bit set on, the source reduces its rate for the corresponding destination. The rate of a ow is controlled at its ingress point, by the flow's index in the Congestion Control Table (CCT). This table contains Inter-Packet Delay entries at an increasing order. For each BECN packet

received, the index of the corresponding flow to that table increases, pointing to larger inter-packet delays, and thus lower injection rates. Flows' rates are recovered using timers. Note that, (a) the reaction time is limited by end-to-end RTT, which is much longer than the local reaction time of RECN; (b) during the long reaction time, congestion trees spread out, thus signaling rate reductions to flows that pass through the congested area but are not responsible for congestion, because they head to other, uncongested destinations or links; (c) the absence of set-aside queues (SAQs) worsens the spreading out of rate reduction to other, unrelated flows; (d) compared to our scheme, this congestion management only acts after the bad effects of congestion have already spread out (long queues - above threshold), while our scheme maintains low queue occupancy.

2.9.8 Per-flow buffers

Sapunjis and Katevenis [47] apply per-flow buffer reservation and per-w backpressure signaling to buffered Benes fabrics; to reduce the required number of queues from O(N2) per switching element down to O(N), the authors introduce per-destination flow merging. That system provides excellent congestion management; however, the required buffer space is M. N per-switch in at least some stages, where M is the number of ports per switch. Current fabrics use switches with tens of ports, M, in order to reduce the number of hops. As shown in section 1.4.3, per-destination flow merging in a three-stage Benes, with N= 1024 and M= 32, requires switches containing 32 K FC windows in the first stage of the fabric. Furthermore, the buffers in this space are accessed in ways that do not allow partitioning them for reduced throughput (e.g. per-cross point); besides high implementation cost, this also complicates variable-size operation. Additionally, it is quite difficult for the architecture in [Sapunjis05] to provide weighted max-min fair QoS, because it merges flows in shared queues: merged-flow weight factors would have to be recomputed dynamically during system operation.

2.9.9 adaptive injection and adaptive layer selection:

adaptive injection(AI) and adaptive layer selection (AL) [48]. In AI a node adaptively selects a layer to which it injects a packet according to the current network status. After injection, the packet uses deterministic routing. In AL, a packet can change the layers during its delivery. AI is especially good when the network size is small, while AL shows better performance in ge neral. In addition, these adaptive decisions are made only when the remaining hops are less than some threshold value, or oblivious routing is selected in other cases.

2.9.10 Regional congestion awareness for load balance

To improve load balance in adapting routing, Gratz, P propose Regional Congestion Awareness (RCA)[49], a lightweight technique to improve global network balance. Instead of relying solely on local congestion information, RCA informs the routing policy of congestion in parts of the network beyond adjacent routers.

2.9.11 Dynamic Adaptive Deterministic switching

DyAD [50] Dynamic Adaptive Deterministic switching, is based on the current network congestion. Each router in the network continuously monitors its local network load and makes decisions based on this information. When the network is not congested, a DyAD router works in a deterministic mode On the contrary, when the network becomes congested, the DyAD router switches back to the adaptive routing mode and thus avoids the congested links by exploiting other routing paths; this leads to higher network throughput which is highly desirable for applications implemented using the NoC approach,

2.9.12 Injection and Network Congestion

Injection and Network Congestion (INC) [51], checks message advance speed in order to detect network congestion. Each virtual channel has an associated counter that increases each time one it is transmitted. At the end of each interval, the channel is considered congested if it is busy and the number of transmitted its is lower than a threshold *fc* (it counter). Once congestion is detected, the applied policies are different depending on whether the node is currently injecting messages towards the congested area.

2.9.13 Distributed Routing Balancing (DRB)

Distributed Routing Balancing (DRB) [52] is a method developed to uniformly balance communication traffic over the interconnection network. DRB takes a similar approach to communications as load balancing does to processes in a distributed environment. The key ideas behind DRB are to distribute communication load based on limited and load-controlled path expansion, in order to maintain low message latency.

Chapter 3

LBDAR: Load-Balanced Distributed Adaptive Routing Algorithm

In this chapter we introduce a new original Load-Balanced Distributed Adaptive Routing (LBDAR) Algorithm with the objective to improve communication performance in interconnection networks based on adaptive and distributed routing. This routing method manages and minimizes interconnection networks congestion. The proposed method is adaptive since it uses some information about the network traffic to avoid congested regions in the network. At the source each path is selected as a function of some resource condition such as packet latency. Then, by task of latency values accumulation performed at intermediate switches and registered it in packet, at destination, an acknowledge packet (ACK) with latencies is sent to the source to inform the source node about the state of latencies in the path involved in the source/destination path, new alternative paths (source-based) will be created if the congestion situation is detected as shown in Figure 3.1. ACK packet can also cause a situation of congestion, because inserting more packets would worsen the situation, we try to avoid this and hence, used distributed router locality-based selection to select new aborted path in situation the latency values registered overpass a high threshold defined. 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 proposed methods are able to avoid the path congestion, while improving the system performance by means of distributing and balancing

communications among the alternative paths. The proposed routing algorithm is adaptive because it takes into consideration the status of the network at any time, to choose the best routes based on congestion situations, channel allocations, latency values, etc. It is also distributed because decisions are performed while the message is traversing intermediate routers in the network.

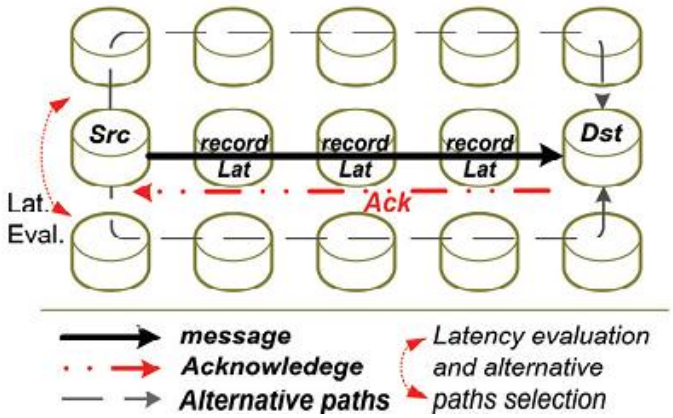


Figure 3.1: Stages of load balancing in routers of interconnection networks

LBDAR algorithm performs four basic tasks: packet latency monitoring, path configuration, distributed route management, and selection of new alternative paths (source-based) according to the congestion situation based on the high latency value that can lead to a congestion in the interconnection network.

The rest of the chapter is organized as follows. The task for monitoring of packet latency in the network, to detect current traffic conditions is explained in Section 3.1. The configuration of paths is defined in Section 3.2. Then, the selection of alternative paths and distributed route management is defined in Section 3.3. Then, the distributed route management is defined in Section 3.4. Finally, LBDAR is summarized and discussed in Section 3.5. Before describing LBDAR, it is necessary to introduce the following assumptions:

In the context of this thesis, we use the term switch to reference network nodes capable of receiving packets on inputs, determining their destination based on the routing algorithm, and then forwarding packets to the appropriate output. Consequently, the term node will be used when referencing terminal and processing nodes, each node has four paths, and the topology is 2-dimensional mesh.

3.1 Packet Latency Monitoring

Packet latency monitoring is accomplished at every intermediate switch by measuring buffer latency values, while a packet is traversing the network. The aggregate latency value will be used later at the source node to identify congested regions.

Congested regions are determined by the time a packet must wait at internal switch buffers before it is injected into the network.

Every intermediate switch executes the monitoring task when a packet goes across its internal ports. Once inside a switch, the latency of the packet is monitored. Each latency is saved, added up and moved into the packet's header where the latency is the time the packet occupies internal buffers waiting to be accepted for delivery to the next intermediate switch or the destination node.

When the packet reaches the destination node, it notifies the source node with the packet's status through a notification packet (ACK). The information in the ACK packet include packet latency values. Using information in the ACK packet, the source node is able to carry out the controlled alternative path opening based on packet latency value.

To control sending the notification packet (ACK), zero load latency is used, which gives a lower bound on the average latency of a packet through the network. Zero load assumption is that a packet never contends for network resources with other packets. Under this assumption, the average latency of a packet is its serialization latency plus its hop latency [2][3].

The pseudo code outlining the packet latency monitoring process is shown in code 3.1.

```
// At each intermediate switch accumulate queue latency
Algorithm PathMonitoring (node src, dst)
Begin
   For every intermediate switch s between src and dst
      For each input packet p of s
         Accumulate queue latency to calculate path latency
        Set  new latency value in p
      End for
    End PathMonitoring algorithm;
```

```
--At destination, the latency sent back to source via an ACK for each alternative path
calculate zero load latency= packet size/bandwidth +hops*switch dealy
If (the path aborted )OR (packet latency greater than zero load latency)
Send ACK
```

Code 3.1: Path packet latency monitoring pseudo code

3.2 Path Configuration

The main goal of path configuration task is to configure paths at each intermediate node, The first step for fulfilling this objective is to initilize the paths of each node, where path number zero is configured as an open path and the rest paths are configured as closed paths. Also the latency for every path is set equal to zero. Once the path has been configured, source nodes are able to use it in the path selection task. A simple pseudo code outlining the path configuration process is shown in code 3.2.

```
Algorithm InitilizePath( )
Begin
   path Path[];
  Path[0].status =OPENED;
  Path[0].latency =0;
  For i  from 1 to 3 do
        Path[i].status =CLOSED;
        Path[i].latency =0;
  End for
End  InitilizePath algorithm;
```

Code 3. 2:Path configuring pseudo code

3.3 Source-Based Distributed Route Management and Path Selection

To determine the appropriate number of alternative paths needed for each source-destination pair, taking into account the information obtained from path latency monitoring phase, the optimal path selection (source-based) task is performed at source nodes to select an optimal alternative path dynamically. The path selection process is invoked before any new application packet is injected into the network. When a new packet is ready to be injected into the network, one of the paths found during the path configuration phase must be selected to send that packet. Path selection is done with the idea of fair distribution of traffic in mind. Aggregate latency

values are the key point in decisions about the path to choose for a particular packet where the path with lowest latency is selected. This strategy ensures that path distribution is done properly, hence more loaded paths are going to be used less, and less loaded paths will receive as much traffic as possible to keep latency values under controlled values. This process has two major goals: avoid the use of congested paths; and distribute the communication load among the paths. The pseudo code outlining the path selection process is shown in Code 3.3.

```
At the source select a  path before inserting  packet

Algorithm PathSelection(real Thl, Tol)

   path Path[];

Begin

// ckeck if latecy for specific path greater than threshold open alternative paths

For i  from 0 to 3 do

if( i<>3)then

if (path[i].latency > =Thl ) then

if( path[i+1 ].status <> OPENED )
            path[i+1 ].status = OPENED;

EXIT FOR

if   (i==3 )   then

if (path[i].latency > =Thl ) then

         if( path[i-1].status <> OPENED )
            path[i-1].status = OPENED;

Else   if( path[i-2].status <> OPENED )
            path[i-2].status = OPENED;

End  for

// ckeck if latecy for specific path less than threshold close that path

For i  from 1 to 3 do

    if (path[i].latency < =Tol ) then

         if( path[i ].status <>CLOSED )
         path[i ].status = CLOSED;

                path[i].latency=0

End PathSelection algorithm
```

Code 3.3: Path selection process

Figure 3.2 shows the flow diagram of the selection phase in LBDAR. Tasks in the figure are performed when a new ACK packet arrives at source node. If the ACK packet has a recorded latency value showing the network is stabilized, i.e., latency values are between an acceptable high and low threshold values, then no action is needed by the selection module and new packets are injected into the network without preliminary path information. High threshold value identifies the network zone where latency switches from the working zone to a more congested zone in the network. The low threshold identifies the point where more alternative paths creation are not necessary and the set of those alternative paths is reduced. The interval between both thresholds then determines the range where a path is valid and there is no need of modifications.

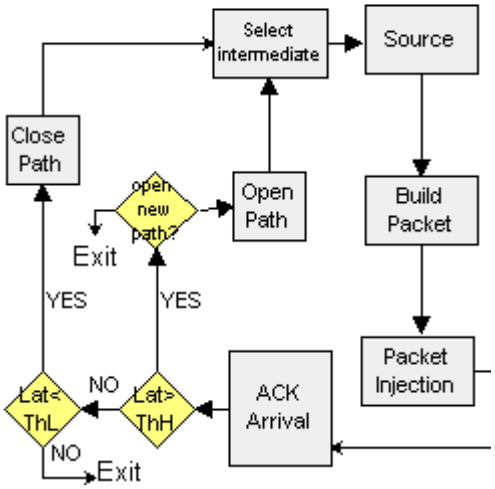


Figure 3.2: Path selection phase flow diagram

Recall that each ACK packet has information about one path latency value, as well as the information about the contending flows patterns found in the congested areas, then path latency is calculated. This calculated latency is compared with the high threshold value, and if it is greater than the threshold, then the configuration phase must proceed with path expansion procedure.

When a Good path is the path with proper intermediate nodes and other requirements is found, then the path selection phase is notified as in the previous case. This selection phase main goal is to keep trajectories working under a traffic load that produces low latency values and within threshold limits. If latency values rise, new alternative paths should be opened to satisfy bandwidth demand. If the latency value decreases and goes below minimum threshold, then the corresponding channel is using more resources than necessary. Thus algorithm must close some alternative paths to free systems resources. The process of closing

paths can be considered as a trigger or indicator to save solutions, because when the path configuration phase detects this situation, then this is a sign that global latency is stabilizing for this source/destination pair and good combination of alternative paths has been found.

Each alternative path is created through selecting two intermediate nodes in each path, one near the source which is called Intermediate Node 1 (IN1) and one near the destination which is called Intermediate Node 2 (IN2). The basic idea is based on the presence of intermediate nodes between the source and the destination. The number of intermediate nodes chosen is two to act as bridges towards building the new paths. This task of setting up the intermediate nodes chooses one node, from a set situated close or centered to the source node, for the first intermediate node. The second intermediate node selection task takes similar approach to find its corresponding node, but related to the destination node. The selected two nodes close to either source or destination or centered are called super-nodes. The set of all paths that can be generated between the source super-node and the destination super-node is defined as a SubPath, as shown in Figure 3.3.

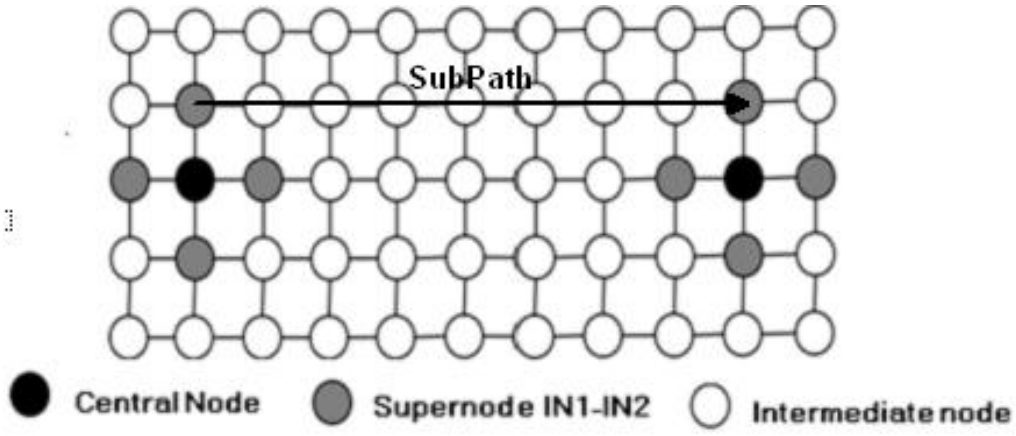


Figure 3.3: Supernode and subPath example

As mentioned above, the set of intermediate network nodes that can be used in alternative paths is called super-node [25]. Intermediate nodes are chosen according to their distance to the switch that has detected the congestion 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 are considered and so on. An example of the distance-based selection of intermediate nodes is shown in Figure. 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.

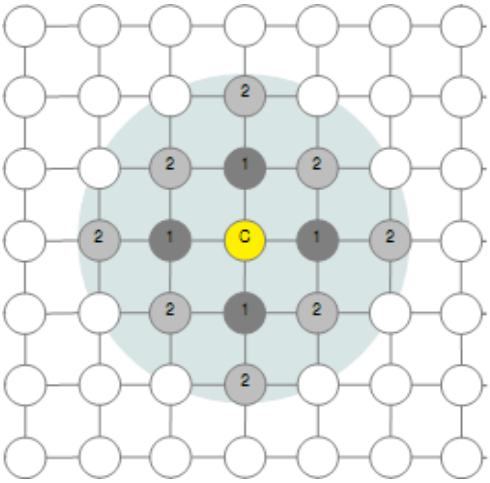


Figure 3.4: Example of a super-node with 1-hop and 2-hop intermediate nodes

According to this approach each packet must traverse several stages with several intermediate Nodes (INs) to reach its destination. These INs are created to define a controlled area where new packets would traverse in their path from source to destination, and serve as scattering and gathering areas for the new alternative fault-free paths.

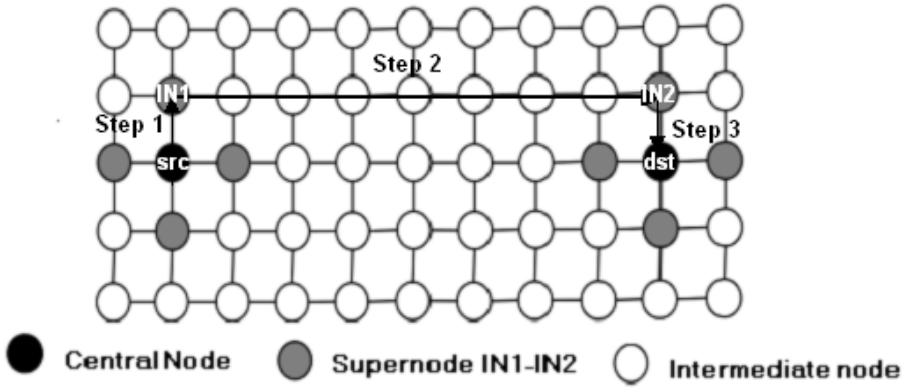
Each alternative path is created, using three steps routing as in fiuger 3.2. These steps are summarized as follows:

Step 1: (src – IN1): From the source node (src) to the first intermediate node (IN1), which is part of the ones centered around the source.

Step 2: (IN1 – IN2): From the first intermediate node (IN1) to the second intermediate node (IN2), which is part of the ones centered around the destination.

Step 3: (IN2 – dst): From the second intermediate node, to the destination.

An example of the steps of the path selection is shown in Fig. 3.5.



Fiuger 3.5: An example of the steps of the path selection

3.4 Distributed Route Management

In section 3.3, we showed how the selection of new paths according to latency values, could detect some cases of congestion before the packets reach the destination as shown in Figure. 3.3.

In order to avoid this, we incorporate adaptively into the LBDAL routing, i.e., we permit routing decisions to be made based on the congestion in intermediate switches. However, such decisions need to be made based on local queues latency information. In order for such local router decisions to be effective, we incorporate a realistic flow control in our routing scheme. This means that when latency buffers reach an overpass threshold, this situation is intended to provide solutions for rerouting packets to alternative aborted paths.

In the process of selecting alternative aborted paths, the switch that has congestion, will first select a set of intermediate network nodes to configure an alternative path to the destination. Alternative paths are basically composed of three path segments grouped in three main stages. The first stage corresponds to the segment between the source and the first intermediate node (close to the congestion node); the second stage comprise the set of segments between the first intermediate node (IN1) and the last intermediate node (IN2) (close to the destination node); finally, the third stage corresponds to the segment from the last intermediate node to the final destination.

An example of an alternative aborted path based on the use of two intermediate network nodes is shown in Figure 3.6.

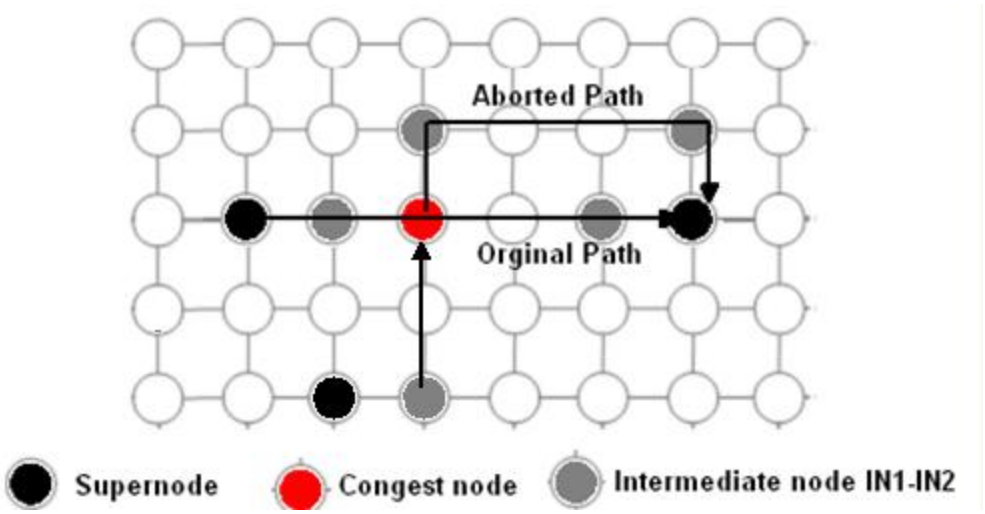


Figure 3.6: Aborted path selection example

A simple pseudo code outlining the adaptive distributed routing process is shown in code 3.4:

```
Algorithm AbortedPathSelection (Node s, Node D)
Begin
  At each intermediate switch between S and D
  begin
     If path between S and D is not aborted  then
     If path  latency > threshold
       If path=path0 then
          Change path locally=YES;
          set path aborted
         try  Open first alternative path1
         or
         try open second alternative  path2
        set new IN1 close to congest node and IN2 close to destination node in sent packet
            trough the path
  end
end algorithm
```

Code 3.4: Aborted path selection pseudo code

3.5 LBDAR Summary and Code

The four tasks of the proposed LBDAR routing algorithm are summarized and integrated as shown in Figure 3.7. To this point, each task was described separately, but all of these tasks are integrated and in most cases they run concurrently. When a new packet is ready for injection, a new path is chosen from current available paths.

Latency values are considered during the selection task. Paths with lesser latency values are going to be selected more often.

Once a packet has been injected into the network, the time it takes to pass each of the intermediate switches in its paths is accumulated and recorded. The time of a packet inside a switch is defined as the time it must wait in the switch's buffers with other packets if this time overpass threshold, Adaptive distributed path monitoring, management, selection and configuration tasks are triggered.

This process of monitoring latency is performed in each intermediate switch, to register and add up latency values until reaching the destination node.

When a packet reaches its destination, it would have all the information about the network situation in its header, and then the destination node proceeds with the notification back to the source node about what the packet has learned during its journey so far, Each destination also monitor the latency if it's value overpass ACK generation threshold to make decision and send ACK to source or not. When ACK packet reaches the source node and the process of packet information analysis is performed, new alternative paths will be created if the congestion situation is detected, If latency values are controlled, then the normal injection of new packets is guaranteed. If congestion is detected, then, before opening a new alternative path, the configuration of new path should take place. When the congestion is controlled by the alternative paths created some paths would be closed. ACK packets will confirm that with latency values under a normal low value, path selection task will proceed to close those paths that are no more necessary to the communication. Moreover, the aborted path selection mechanism is invoked only when congestion is detected. Similarly, latency updates are performed while messages are waiting in switch queues. All the tasks are performed concurrently, thus packets delivery is performed while the tasks of configuring new paths are executing. At the intermediate switches this condition is also true, while packets are waiting in the internal switch's buffers, the task of accumulating and saving latency and checking it are being carrying out, so no extra overhead is included. Path configuration and path selection are performed at source nodes and not at intermediate switches. So intermediate switches are not more complex than normal ones found under other implementations. A simple pseudo code outlining the all tasks of LBDAR routing algorithm as an integrated single program are shown in code 3.5.

```
Algorithm ABDAR( )
Begin
   At source when a new packet is to be injected in network:
      Invoke source based  route selection task;
     If received packet type ==ACK   then
         If Latency>=ThHThen
            Opne  path;
        Else if Latency<=ThL Then
            Close path;
   At destination:
      send back ACK if latency of recived packet is less than Ack generation threshold
      ack generation threshold=zeroload latency;
     zero load latency=hops* packet size/bandwidth +hops*switch dealy;
```

```
    At the intermediate switch
  Accumulate queue latency to calculate path latency;
     Set  new latency value in packet;
    If  latency is greater than threshold
       Trigger adaptive locally task;
    Else
       Continue to next intermediate switch or destination;
end algorithm
```

Code 3.5: LBDAR routing algorithm summary pseudo code

Figure 3.7 shows the LBDAR routing algorithm tasks that correspond to those in code 3.5.

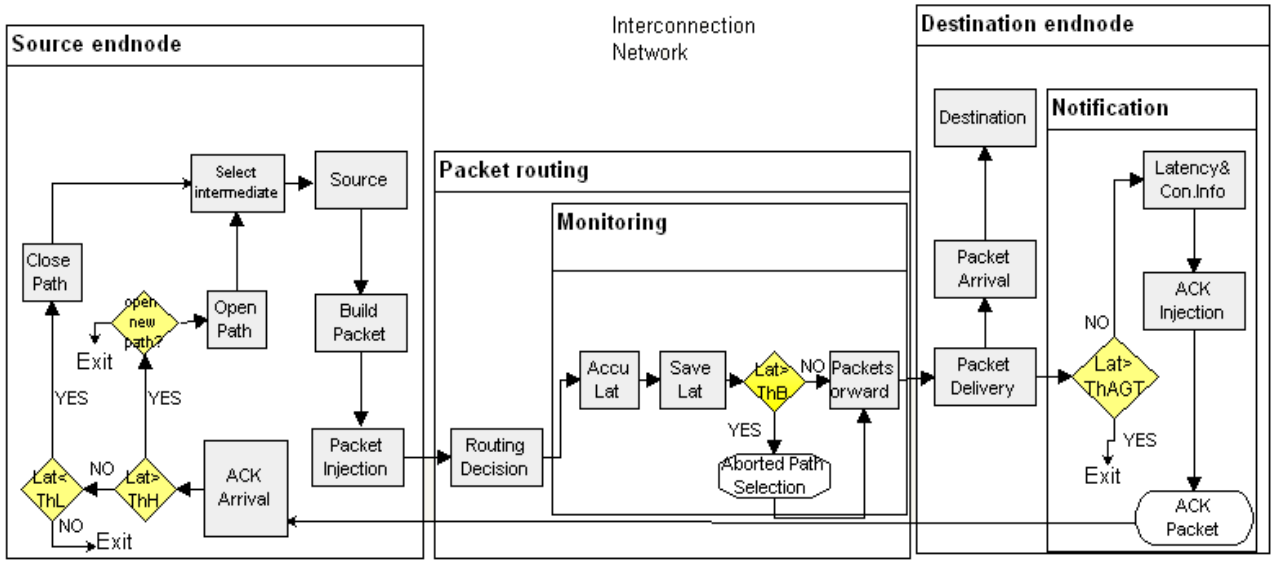


Figure 3.7: LBDAR routing algorithm tasks that correspond to those in code 3.5

3.6 Message blocking avoidance:

To avoid message blocking in our network we apply a flow control method to allocate the appropriate paths for the packet resources. Since we use finite buffers, we incorporate virtual channel flow control to avoid message blocking in the network. Our implementation of LBDAR employs four virtual channels (VCs) per unidirectional physical channel (PC) to achieve message blocking freedom in the network.

As an analogy to the real world, we should provide alternative pathways when congestion occurs in a highway road, so that the incoming traffic can overcome the accident and continue its way. Having this analogy in mind, at hardware level, if a packet gets blocked in a buffer while expecting other resources to get free, incoming packets should not get blocked by this packet. The flow control mechanism should provide them an escape path in the form of an

alternative buffer, so that the packet can proceed. The implementation of this in hardware is the partition of the used buffer in several pieces that we call virtual channels.

If we consider that the buffer is a First In First Out (FIFO) queue, virtual channels are the partitioned representation on several smaller parts of memory, called sub-queues. These sub-queues are those that are used as escape paths for the packets. The implementation can be in hardware or software level. In hardware, the implementation can be in the form of separated buffers with a circuit flow control mechanism. In software level, the unique buffer is treated as partitioned, applying the flow control policy through a software implementation over the virtual channels. To make our job easier for this purpose we used the virtually a predefined model of a FIFO buffer queue as shown in Figure 3.8.

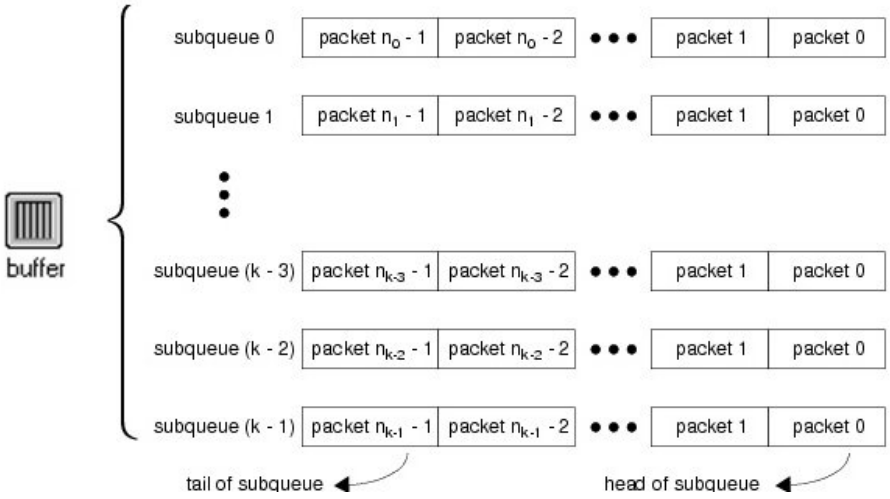


Figure 3.8: Buffer with virtual channels

Each queue module contains a definable number of sub-queues as we see in Figure 3.8. A sub-queue is an object which is subordinate to the queue object and which has its own attributes used to configure it. The capacity of each sub-queue to hold data is unlimited by default, but a limit may be set on the number of packets or the total size of all packets (or both) within a sub-queue. It is up to the processes in the queue to determine what action to take when sub-queues become full: packets may be removed to create space for new arrivals, or the new arrivals may be discarded. Because the user controls the process model executed by a queue, it is possible to model any queuing protocol by defining the manner in which the sub-queues are accessed and managed.

3.7 Discussion

Throughout this chapter, we have described the proposed congestion management adaptive routing method Load-Balanced Distributed Adaptive Routing (LBDAR) algorithm. Let us now provide a brief comparison with the most relevant related works in literature.

Although LBDAR 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 the Routing Methodology Based on Intermediate Nodes [3], [41]; and the Routing Methodology for Dynamic congestion[36].

LBDAR is based on a multipath adaptive routing approach and needs only four virtual channels for message blocking avoidance. These are the biggest differences between these methods and our proposal.

LBDAR has been designed to deal with both static and dynamic congestion. the strongest point of LBDAR is that it provides a simple but sufficiently complete and satisfactory solution to the problem of dynamic congestion management in interconnection networks. This constitutes the major contribution of the method since the most significant information about a network's congestion management is whether it can function at all in the presence of congests.

Chapter 4
Evaluation And Results

In this chapter, we present the experiments carried out to probe the idea presented in this work proposal, and the results obtained. Also, simulation techniques, tools, and the environment of the tests are introduced and described.

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, to accomplish these goals comparison of the performance is carried out, where the main metric used is latency of the entire network for each algorithm.

In the following sections, we describe the three main aspects of the evaluation process; simulation models in section 4.1; workload models in section 4.2; and metrics in section 4.3. Finally, we present the evaluation method and results of each proposal separately.

4.1 Simulation Models

A simulation model provides mechanisms to analyze performance of a real system or behavior under different configurations when the system is not really implemented yet. Even if the system is already implemented, a simulation tool can be very useful because it allows many configurations to analyze specific components or perform different execution under distinct environmental variables to study or stress some particular feature of the system. Simulation and modeling are very useful because they can provide insight knowledge about a system [56].

To implement our proposed algorithm, the commercial simulation tool OPNET modeler was used [57]. This tool allows the simulation of distinct communication network protocols, distributed systems and other technologies under different configurations, also providing the tools to analyze those results.

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.

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.

Process level: behavior of modules is programmable via their process models. They consist of finite state machines (FSMs) containing blocks of enhanced C/C++ user code and OPNET Kernel Procedures (OKPs). 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.

Architecture of Network Components

4.2 Processing Node

The internal structure of the node model is shown in the Figure 4.1. The node model is separated in the node processor and the network interface. These two parts are connected through statistic and packet streams, and with the intermediate action of the AMR handler.

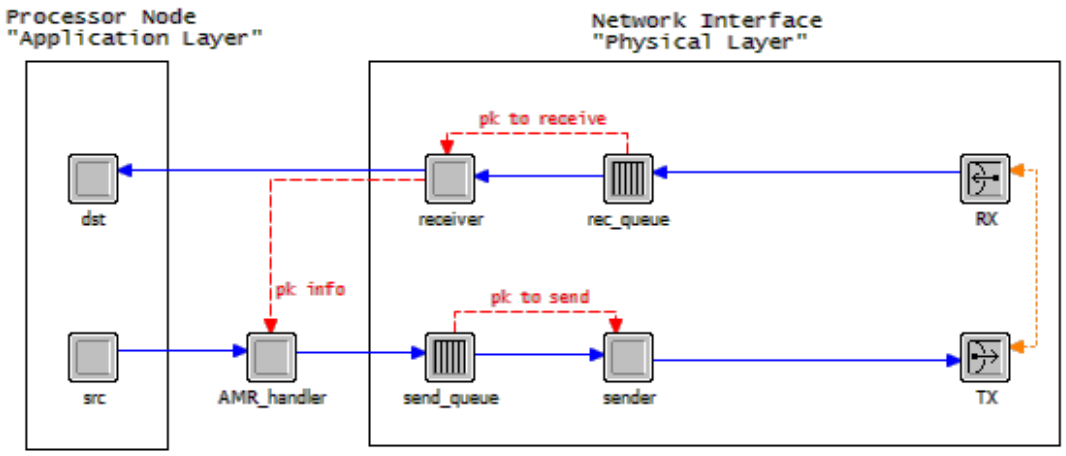


Figure 4.1: Processing nodes model

The processor consists of a packet source (src) and a packet consumer (dst). The source module (src) generates link layer data packets according to a specific synthetic traffic pattern. This module is shown in Figure 4.2. Additionally, processing nodes are provided into several attributes related to packet generation such as: injection rate, start and stop time, packet format and length, workload characterization, etc.

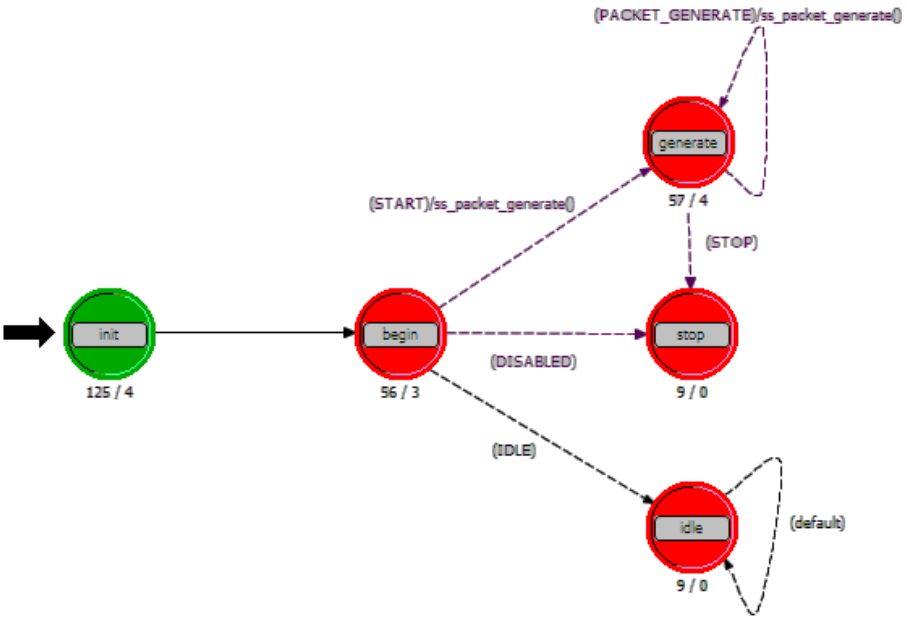


Figure 4.2: Finite state machine of src processing nodes

A component (dst) for the destination node is also modeled, where packets are received and simulation of packet delivery to higher layers is performed. The consumer module (dst) shown in Figure 4.3 receives the incoming packets and is responsible for the calculation of the offered and received load that has travelled inside network and also for the average global latency of those packets.

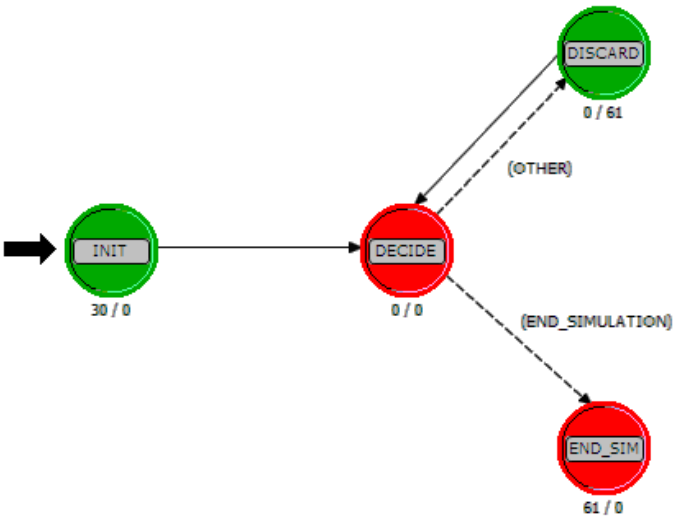


Figure 4.3: Finite state machine of dst processing nodes

The AMR_handler(Adaptive mesh router) module is situated between the node processor and the network interface of the node. AMR_handler is responsible for receiving the packets from the input port and recognize their type and also receive the packets that come from the src module and forward them to the sender module that will insert them to the network. In Figure 4.4 is shown the finite state machine of the AMR_handler component of the processing node.

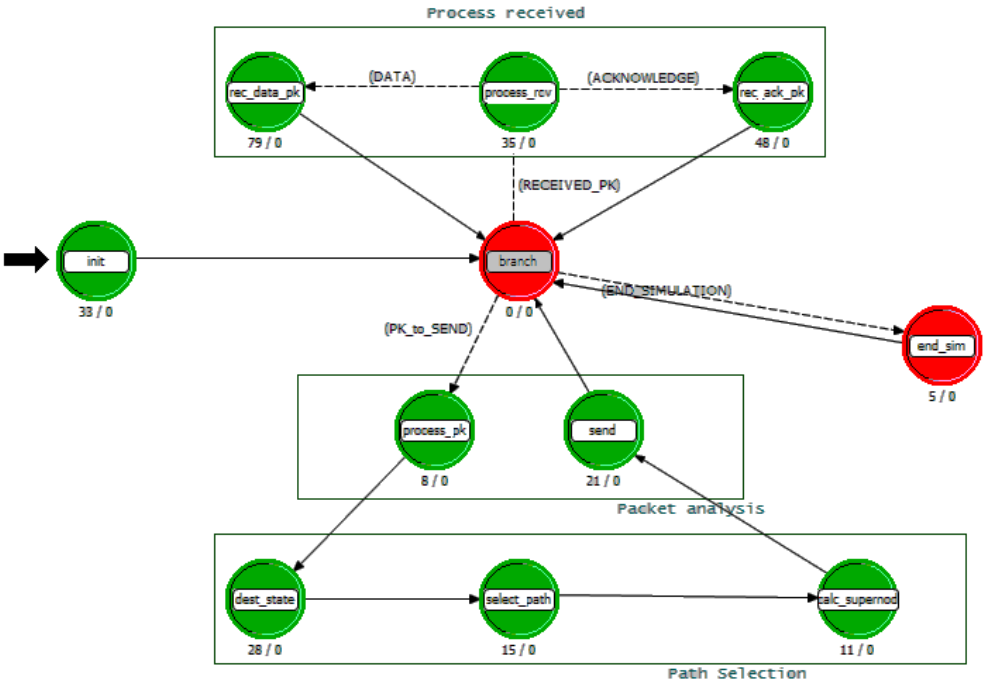


Figure 4.4: Finite state machine of AMR_handler processing nodes

The network interface of the node is composed of six modules. These are the transmitter (TX) and receiver (RX) units that give access to the physical layer, the input and output buffers which are rec_queue and send_queue modules respectively and the receiver and sender modules which are settled after the buffers. They are modeled in Figures 4.5, 4.6, 4.7 and 4.8 respectively.

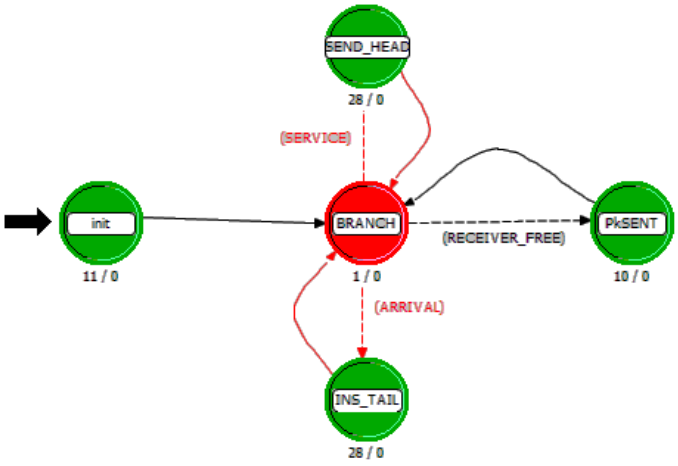


Figure 4.5: Finite state machine of rec_queue processing nodes

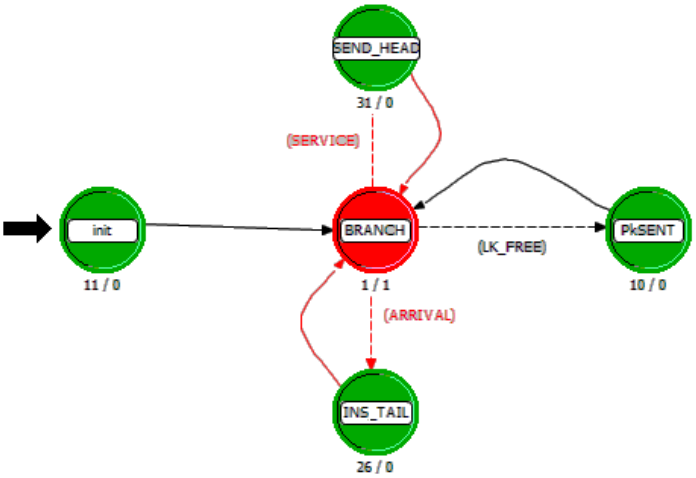


Figure 4.6: Finite state machine of send_queue processing nodes

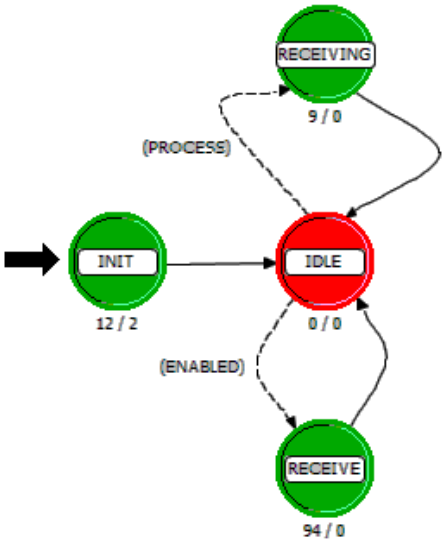


Figure 4.7 : Finite state machine of receiver processing nodes

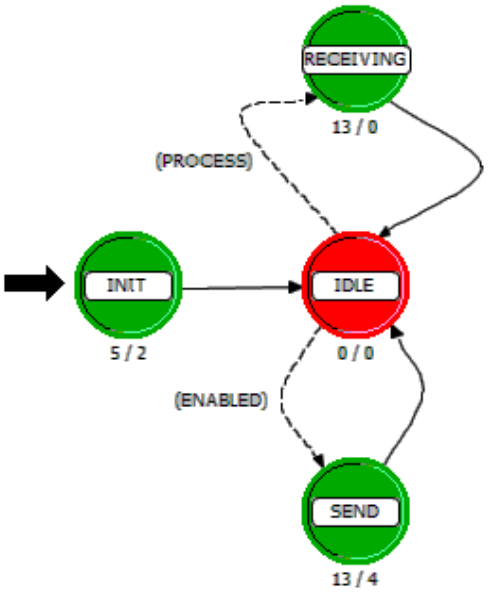


Figure 4.8: Finite state machine of sender processing nodes

When a packet is received by the network from the RX unit, it is passed into the rec_queue module tail, which works similarly to the switch buffers, and informs through a statistic wire the receiver module that a packet is inside the queue. When the packet reaches the head of the queue then, it is sent through a packet stream to the receiver module. The receiver module after a small time delay receives the incoming packet and depending on the type of the packet, data or ACK, gets the latency values for each one of the packets. After that it informs the AMR_handler through a statistic wire and sends the packet to its destination, the dst unit, through a packet stream. The same happens also, when a packet needs to exit from the node. The packet is received by the send_queue and stored in the tail of the queue. The send_queue informs the sender module with a statistic and when the packet reaches the head of the queue it is sent to the sender module where after a small delay is sent to the physical layer.

4.3 Switch Node

The internal structure of the implementation of the 8-ports switch node is shown in Figure 4.9. This model 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: switch info, routing unit, the arbitration unit, and the crossbar unit.

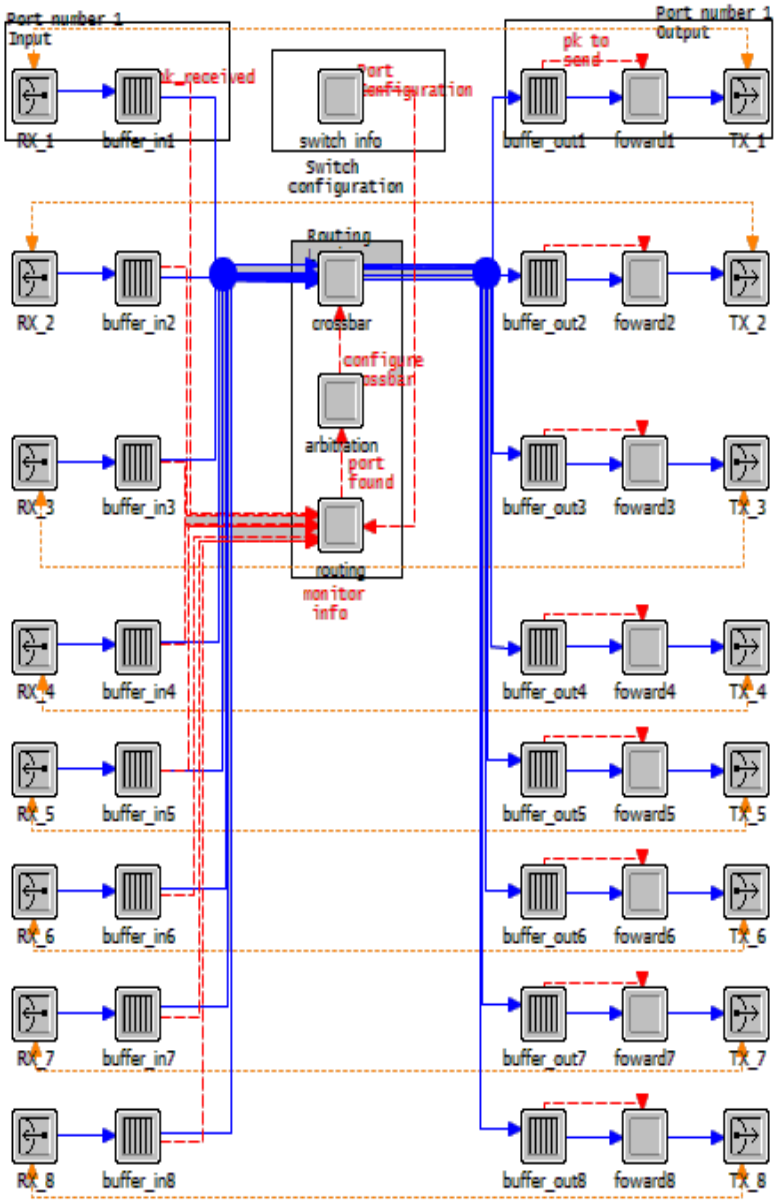


Figure 4.9: Switch model implementation

The switch info unit shown in Figure. 4.10 is the first unit that is accessed inside the switch. It has the biggest priority from all the modules of the switch. Switch info in first step initializes the network. It receives information for the number of the nodes and characterizes them in nodes and switches giving their names and coordinates. Also the switch info allocates memory for the nodes that it has found. For the initialization of the simulation process, it finds the ports in each network node and gives them a port number and checks the connected links in the switch and names them also. The unit also discovers the neighbors and constructs the topology while understanding the logical position of the switch and the geographical position of the neighbors. Finally switch info unit informs the routing unit, through a statistic wire, for the ports and it de-allocates the memory that has been used.

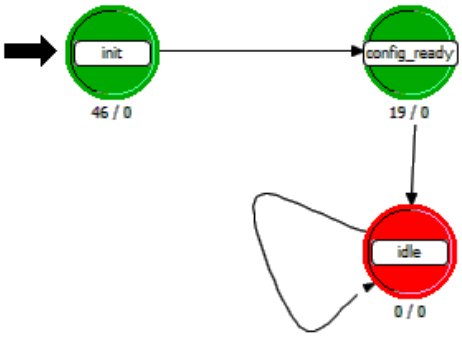


Figure 4.10: Finite state machine of switch info unit

The routing unit shown in Figure 4.11 is the first step for the routing mechanism that the switch uses. It receives information through statistic wires firstly from the switch info unit that informs the routing mechanism with information about the switch situation. In addition, it receives information about each one of the input buffers, that represent equal input ports. Unit is also connected through statistic wires to send information, with the arbitration unit. The routing process starts by receiving the number of the input ports that it is connected with and allocates the appropriate memory space, for equal number of packets that are waiting to be routed. Then, it registers the statistics with which it is going to inform the arbitration unit. Here also, it initializes variable for a round robin approach search between the input channels.

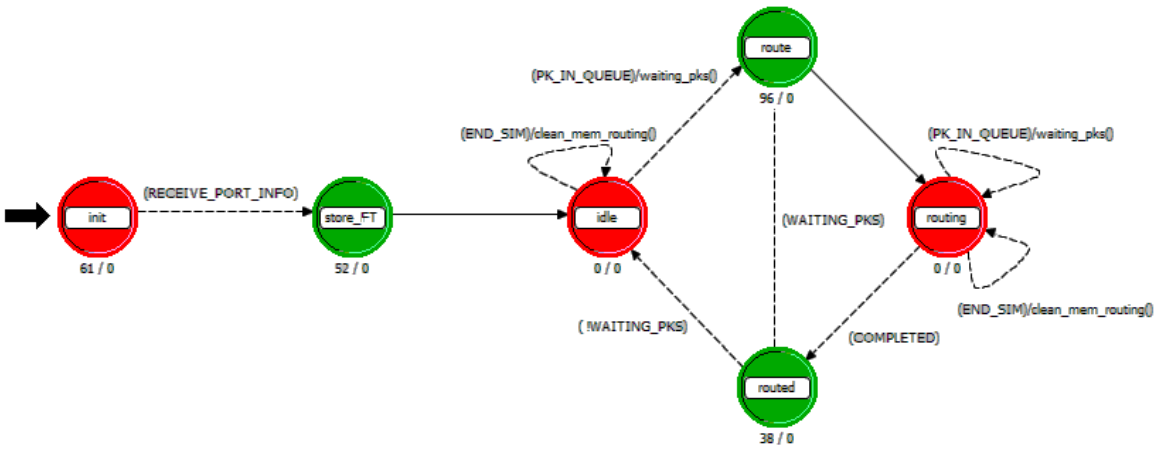


Figure 4.11: Finite state machine of routing unit

In the next state, the routing unit receives from the switch info information, through the statistic wire, in order to arbitrate. The information received is the port configuration, with which it checks if the switch has a valid logical position, the routing algorithm that is used for the packet traversal and the low and high values of the threshold. In this part also the unit makes pairing between the input port number and the equivalent input buffer. Now the routing unit comes in a ”pause” situation. Here it waits to clean the memory for waiting packets if simulation is

terminated, and also waits for incoming interrupt by one of the connected input buffers. When an interrupt occurs, the unit receives all the information from the interrupt, increases the number of waiting packets by one and goes in the next state to route the packet. The unit now searches in all ports, using round robin, to find a waiting packet which needs to be routed. When it finds it, it applies the routing algorithm that gives the appropriate output port, clearing the waiting packets memory for the specified input and decreasing the number of the packets that are waiting to be routed by one. Before it exits from the state, it makes some checks for the received output port, and sets that routing as completed, continuing with a self interrupt it proceeds to the next state. Here the routing unit comes once more in a "pause" situation where it waits for interrupt of a packet that is waiting to be routed. If from the previous state it is declared that the routing has been done, it proceeds to the next state where it prepares information to be sent in the arbitration unit, informing that the sender module has a packet in the queue.

The arbitration unit shown in Figure 4.12 is a connection between the routing unit and the crossbar unit. Arbitration receives information from the routing unit and passes it to the crossbar. Given the input and the output port, the unit finds the correct stream to forward the packet to that port.

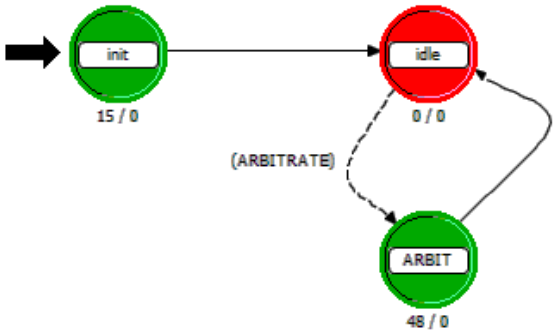


Figure 4.12: Finite state machine of arbitration unit

The Crossbar unit shown in Fig. 4.13 is responsible to receive incoming packets from the input buffers and forward them to the requested output buffers that will lead them to a predetermined output port. The crossbar unit is informed from the arbitration unit by an interrupt, which declares that a specific input port has requested an output port. The unit checks in the information received by the arbitration unit. If it is necessary, it updates specific packet headers and forwards the packet to the requested output port.

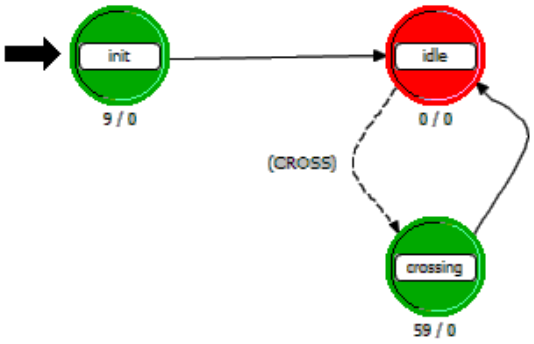


Figure 4.13: Finite state machine of Crossbar unit

The receiver unit represents the input port of the bidirectional channel. The receiver passes the incoming packet from the physical layer of the channel to the input buffers of the switch.

The input buffer is connected between the receiver unit, from where it receives any incoming packet, with the routing unit which informs that a packet needs to be routed and with the crossbar unit where its sends the packet so that it can find its requested output buffer and port. Input buffer starts by initializing the statistic that will inform the routing unit, receives the input port number and the internal bandwidth of the buffer. Before exiting the state it declares that it has no out-coming packet. When the unit receives an interrupt for incoming packet, it receives the packet and sees if the buffer is empty. If it is, then it inserts the packet in the tail of the FIFO queue, and informs the routing unit by a statistic wire that it has a packet waiting to be routed, in the specified port. When the module has an access to the interrupt that has appeared it continues to the next state. Now the module searches in the head of the queue, and if the queue is not empty and contains a packet, module has access to that packet. The module next gets the latency of the packet and updates the average occupation of the buffer, while removing the packet from the head of the queue and receiving its size. In sequence, the module sends the packet through a packet stream to the crossbar without causing an interrupt, thus earlier it has informed the routing mechanism through a statistic wire. The module computes a delay based on the packet size, the internal bandwidth and delay of the switch and when this time passes it creates a self interrupt which makes the module proceed to the last state. In this last state the module checks for the buffer if its empty, to find if any new packet has reached the head of the queue while the last packet was exiting from the queue. If a new packet has appeared the module gets once more access to the head of the queue. Before it exits from the state it takes the information from the new arrived packet and informs once more the routing unit through a statistic wire.

The output buffer module is connected with the crossbar unit from which it receives packets through a packet stream. The unit is also connected and sends information, with the forward unit, through a statistic and a packet stream. Output buffer module, after initializing the necessary variables, proceeds to the next state to receive a packet. Here the unit receives through an interrupt stream the packet arrived. It checks if the tail of the queue is empty, and if it is, it inserts the packet inside the tail. While there are no other packets that are waiting to exit or occupying the link, the unit informs the sender module that there is a packet inside the queue. In the next state, the unit receives a request to access the queue. If the queue is not empty and contains packets, then the module receives the first packet that is in the head of the queue and calculates the buffer latency. After that it removes the packet from the queue, receives its size and sends it to the forward unit without causing an interrupt because it has already informed the unit by a statistic wire. Thus the packet will occupy the link for some time, the unit calculates that time based on the bandwidth of the switch and the packet size. After that time expires the unit causes a self interrupt that makes it proceed to the next state. In this last state after the last packet has completely left from the queue, the unit searches in the head of the packet once more if it has a new packet, and if so, it informs once more the forward unit through a statistic wire.

The transmitter unit represents the output port of the bidirectional channel. The transmitter passes the incoming packet from the output buffers to the physical layer of the channel.

4.4 Workload Models

The evaluation of interconnection networks requires the definition of representative workload models. The workload model is basically defined by three parameters: distribution of destinations, injection rate, and message length [2].

Recently, several specific communication patterns between pairs of nodes have been used to evaluate the performance of interconnection networks: bit reversal, perfect shuffle, butterfly, matrix transpose, and complement. These communication patterns take into account the permutations that are usually performed in parallel numerical algorithms [58, 59, 60]. In these patterns, the destination node for the messages generated by a given node is always the same. Therefore, the utilization factor of all the network links is not uniform. However, these distributions achieve the maximum degree of temporal locality. These communication patterns can be defined as follows:

Bit reversal. The node with binary coordinates $a_{n-1},a_{n-2},...,a_1,a_0$ communicates with the node $a_0,a_1,...,a_{n-2},a_{n-1}$.

Perfect shuffle. The node with binary coordinates $a_{n-1},a_{n-2},...,a_1,a_0$ communicates with the node $a_{n-2},a_{n-3},...,a_0,a_{n-1}$ (rotate left 1 bit).

Butterfly. The node with binary coordinates $a_{n-1},a_{n-2},...,a_1,a_0$ communicates with the node $a_0,a_{n-2},...,a_1,a_{n-1}$ (swap the most and least significant bits).

Matrix transpose. The node with binary coordinates $a_{n-1},a_{n-2},...,a_1,a_0$ communicates with the node $a_{n/2-1},...,a_0,a_{n-1},...,a_{n/2}$.

Complement. The node with binary coordinates $a_{n-1},a_{n-2},...,a_1,a_0$ communicates with the node $a_{n-1},a_{n-2},...,a_1,a_0$.

The mathematical descriptions of the above communication patterns are shown in Table 4.1.

Table 4.1: Mathematical descriptions of synthetic traffic patterns

Pattern	**Destination**	
Bit reversal	$d_i = s_{n-i-1}$	$\forall n: 0 \leq i \leq n-1$
Perfect shuffle	$d_i = s_{(n-i) \bmod n}$	$\forall n: 0 \leq i \leq n-1$
Butterfly	$d_{n-1} = s_0 \quad , \quad d_0 = s_{n-1}$	$\forall n: 0 \leq i \leq n-1$
Matrix Transpose	$d_i = s_{\left(i+\frac{1}{2}\right) \bmod b}$	$\forall n: 0 \leq i \leq n-1$
Complement	$d_i = s'_i$	$\forall n: 0 \leq i \leq n-1$

In our simulation experiments we have included two different kinds of synthetic workloads: Perfect shuffle and matrix transpose in order to evaluate our proposed routing protocol.

4.5 Evaluation Metrics

To evaluate the performance of the network, 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 [2]. The average latency for each packet *x* reaching a destination node*i* is given in Eq. 4.1, where $l_i[x]$ is the latency value of the packet *x* at the node *i*.

$$L_i[x] = \frac{1}{x}(l_i[x] + (x-1) * L_i[x-1]), \forall x \neq 0 \qquad (4.1)$$

The global average latency is calculated by averaging the latencies of every packet, and is measured in seconds as defined in Eq. 4.2, where n is the number of destination nodes.

$$L = \frac{1}{n}\sum_{i=1}^{n} Li \qquad (4.2)$$

We have measured throughput as the data rate in bits per second 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.

4.6 Packets Format

In LBDAR protocol, packets include mechanisms for identifying alternative paths (either aborted based or source-based) and also for carrying a path latency value.

Data packets include a multiple header for storing the additional information about the intermediate nodes as shown in Figure 4.14. In order to simplify the packet format, LBDAR configures both escape and source-based alternative paths using only two intermediate nodes. If a switch detects congestion along the path, that switch 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 order to route packets correctly, each switch 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. Also 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 .Also LBDAR uses ACK packets for purpose reporting path latency values as shown in Figure. 4.15.

source	Intem1 node	Intem2 node	destination	Path latency	Escape path E	Hops	path	Data id

Figure 4.14: LBDAR data packet format

source	Intem1 node	Intem2 node	destination	Path latency	Escape path E	Hops	path	Data id

Figure 4.15: LBDAR ACK packet format

4.7 LBDAR Validation Experiments

In order to validate Validation, experiments were done over an 8x8 2D mesh network topology, distributed across 64 intermediate routing nodes, and one processing node connected to each of the switches as shown in Figure 4.16.

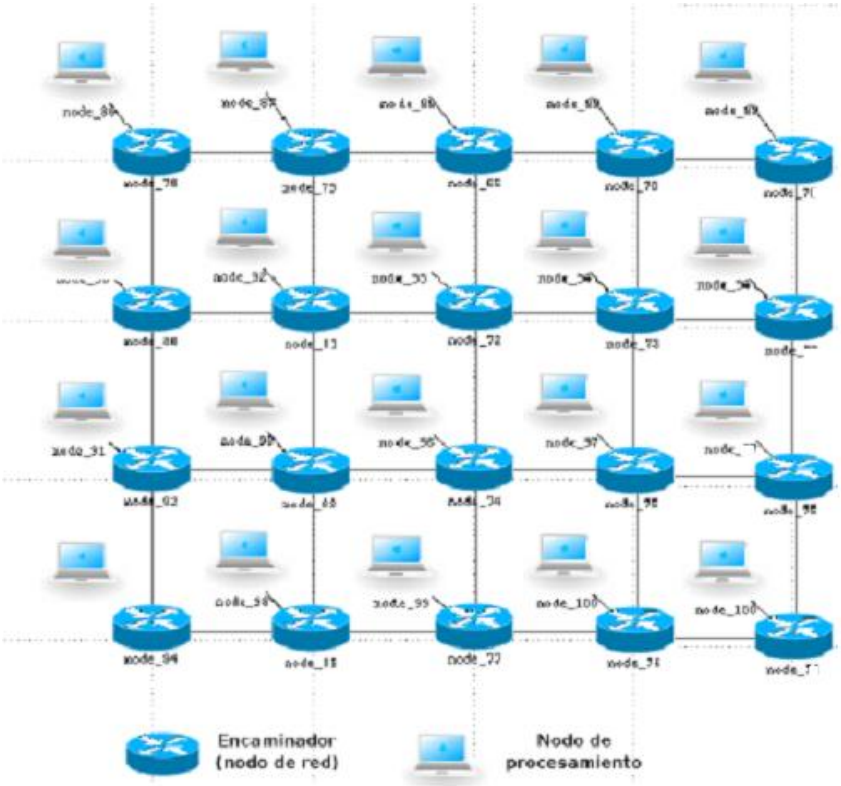


Figure 4.16: Network model of validation experiments of LBDAR

Simulations under different test scenario were performed as shown in Table 4.2. The mesh network topology is selected for simulation experiments, because it offers one essential feature for our proposed protocol, which is the great number of alternative paths that can be found to reach a destination node. Besides, it works well in many experiments in the area of interconnections networks.

Table 4.2: different simulation parameters of LBDAR

Parameters	Value
Time	1.02 seconds
Traffic generation	poisson
Routing Algorithms	LBDAR, DOR, GOAL
Network size	8x8 (64 nodes)
Packet size	1024 bytes
Flow control	Virtual cut-through
Link bandwidth	2 Gbps
Network topologies	Mesh
Buffer size	4 KBytes
Traffic patterns	Perfect shuffle, matrix transpose

4.8 LBDAR Evaluation

The main goal of this evaluation is to visualize the effect of the LBDAR, and then compare the LBDAR performance with other algorithms to make some final conclusions. Figure 4.17 shows the latency behavior of LBDAR, DOR [3] and GOAL [46] algorithms.

We see from this figure that both DOR and GOAL algorithms practically behave the same way, while LBDAR is continuously opening and closing paths to try to stabilize global latency value.

With at most four alternative paths for these experiments, LBDAR incurs a remarkable lower latency than DOR and GOAL algorithms. Figure 4.17 shows results with 8X8 communicating nodes connected using shuffle traffic pattern. Latency reduction under the perfect shuffle pattern is 30%. Figure 4.18 shows latency under the matrix transpose pattern which shows a latency reduction of around 20% of LBDAR compared to DOR and GOAL algorithms. LBDAR uses less network resources for a given load, because those resources are efficiently handled.

We see that latency values of the different algorithms in the rising area of Figures 4.17 and 4.18 is diverging from each other. This is because LBDAR has the best routing solution, and no extra time to find the routes is necessary.

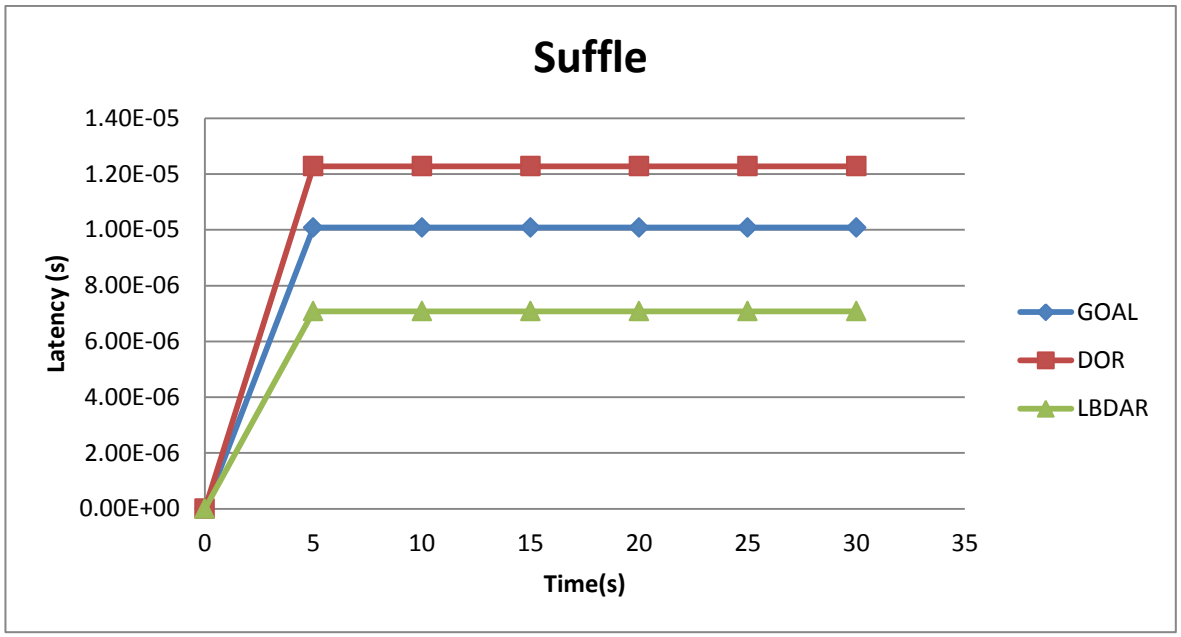


Figure 4.17: Average latency against simulation time for the shuffle traffic pattern

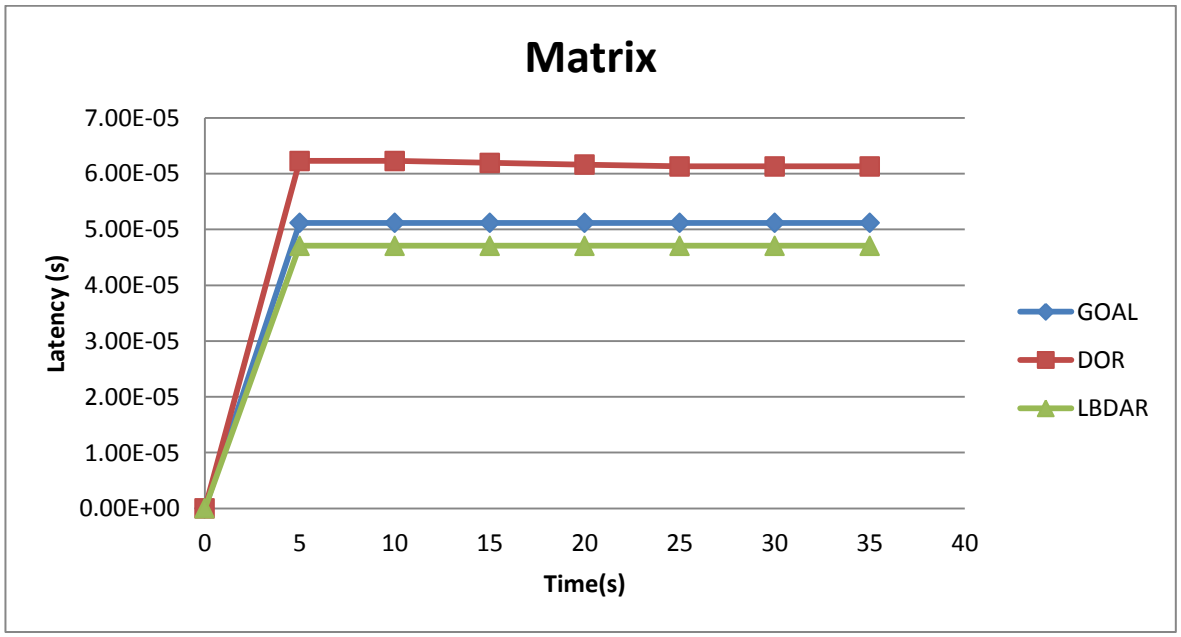


Figure 4.18: Average latency throughout simulation time for the matrix traffic pattern

4.9 Latency Distribution

Latency distribution across the entire set of switches is presented in Figures 4.19, 4.20 and 4.21. This kind of 3D Figures is known as surface-maps, where axis x and y represent the coordinate of a node, and the z axis represents some specific values for that node, such as the average latency. The surface in the surface-maps is the average contention latency at buffers in our experiments. We use surface-maps with different values in the z-axis to ease the visualization of the evaluation results.

These experiments establish some fixed destinations to produce network congestion. Remaining network nodes inject uniform load to create "noisy" traffic. DOR AND GOAL response to the traffic is always the same, due to their use of the same solution for different cases of congestion.

The idea of these graphs is to show how traffic is distributed across the network and how LBDAR strategy influence the gain in latency values expressed so far.

Figure 4.19 depicts LBDAR behavior, where we see a high peak of latency, compared to DOR and GOAL behavior shown in Figures 4.20 and 4.21. It is clearly visualized that LBDAR has less congested areas in the whole network. The reason of this behavior, is related to the fact that DOR uses only one path and GOAL uses short paths during their route search. Another interesting aspect shown in these figures, is that LBDAR has its maximum peak of latency below the maximum registered for DOR and GOAL, as a result of less congested paths introduced by using the optimum solution. All this situation leads to a better distribution of traffic load across a correct number of communication nodes, avoiding situations where intermediate routers are overloaded due to handling traffic, while other routers may be idle or underutilized.

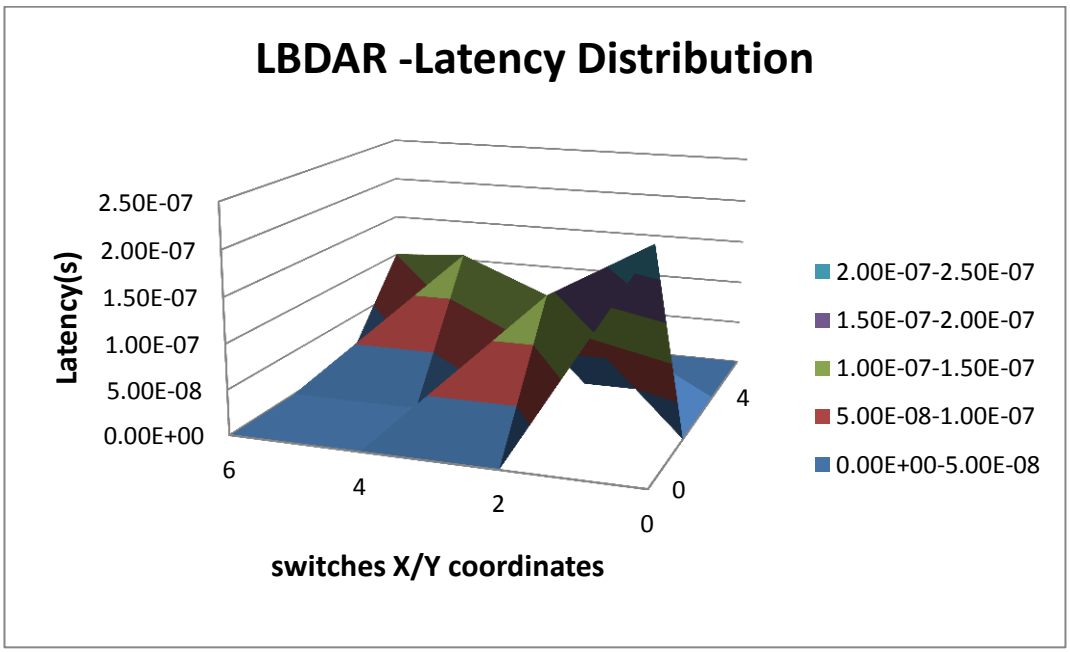


Figure 4.19: Mesh network latency map for LBDAR algorithm

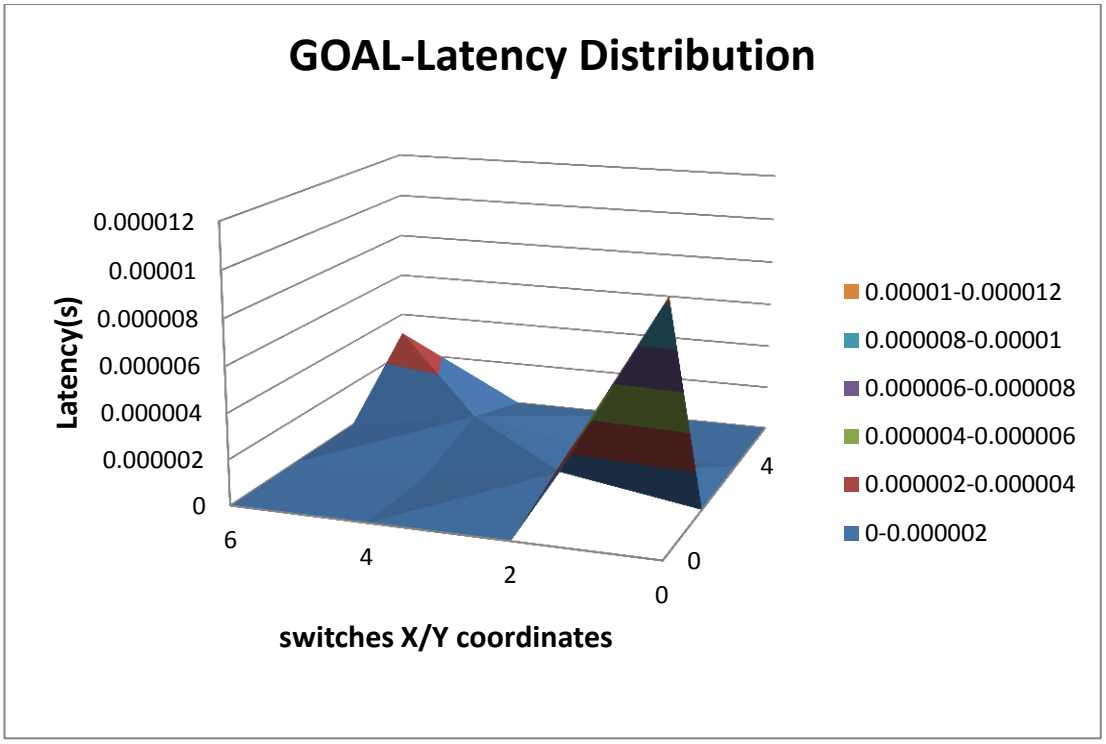


Figure 4.20: Mesh network latency map for GOAL algorithm

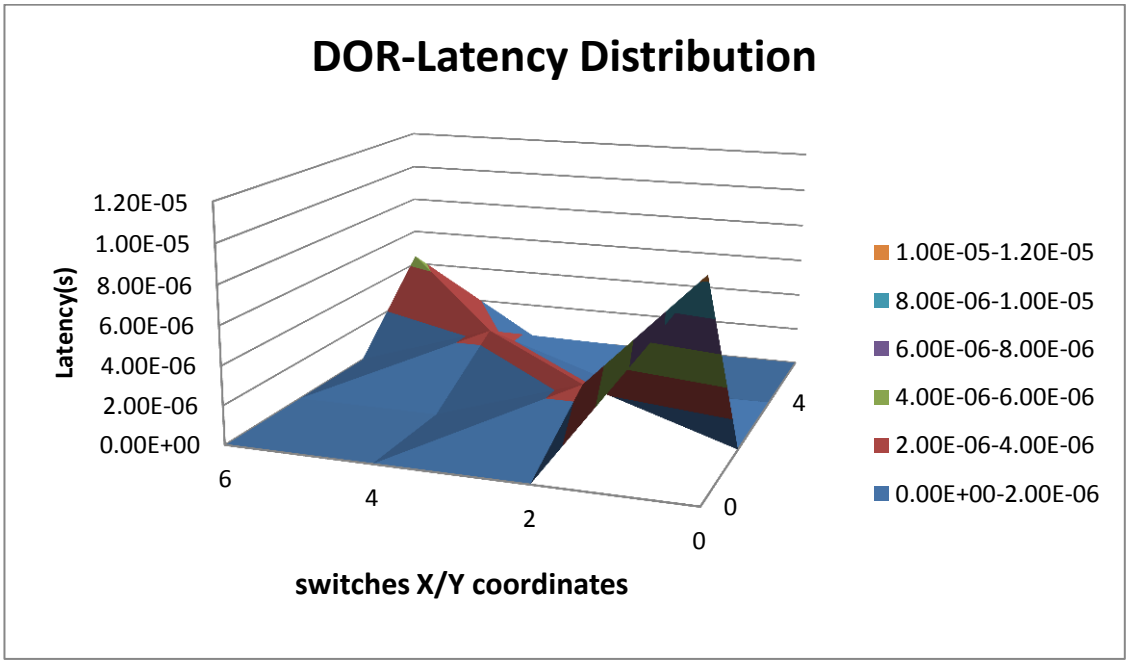


Figure 4.21: Mesh network latency map for DOR algorithm

Chapter 5

Conclusions

5.1 General conclusions

The design of an interconnection network has three aspects — the topology, the routing algorithm used, and the flow control mechanism employed. The topology is chosen to exploit the characteristics of the available packaging technology to meet the bandwidth, latency, and scalability [61] requirements of the application, at a minimum cost. Once the topology of the network is fixed, so are the bounds on its performance. For instance, the topology determines the maximum throughput (in bits/s) and zero-load latency (in hops) of the network. This thesis has demonstrated the significance of the routing algorithm used in the network towards achieving these performance bounds. Central to this thesis is the idea of load-balancing the network channels. A naive algorithm that does not distribute load evenly over all channels, stands to suffer from sub-optimal worst-case performance. However, unnecessary load-balancing is overkill. Spreading traffic over all channels when there is no uneven distribution of traffic, leads to sub-optimal best-case and average-case performance. This thesis explores routing algorithms that strive to achieve high worst-case efficiency without sacrificing performance in the average or best-case.

This work proposes a new method of routing in an interconnection network, called Load-Balanced Distributed Adaptive Routing (LBDAR). This new strategy is based on path distribution under congestion situations across many alternative paths, to keep latency values controlled and enough bandwidth available to applications running in the interconnection network.

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 two. Taking these definitions as a starting point, we have

been capable of performing the analysis of issues influencing the development of the proposed congestion management routing policies. From this approach, we have been able to provide flexible solutions to the problem of congestion for high-speed interconnection networks in chapter three. finally modeling and simulation tools, described in chapter four. Having completed all the stages that comprise our research, we are now able to present conclusions and further work of this thesis.

5.2 Results Conclusions

In this dissertation we have proposed a load balanced routing approach, called LBDAR, which is targeted to improve interconnection network latency values under congestion or hotspot situations. Results obtained from the experiments carried out show that our proposal is valid as a mechanism of load balanced routing in interconnection networks. We also observed that by applying our mechanism, we can obtain relevant improvements in latency values under specific traffic loads. the proposal algorithm can extract relevant information about the agents involved in the congestion situation at any particular intermediate node. The information gathered during the process is notified to involved source node, so that it can take proper actions to control congestion in the network. LBDAR use all these collected information, in a particular source node, to start the procedure of selection. selection is done opening new alternative paths to alleviate high latency values registered during hotspot situations.

After our experiments we saw that LBDAR performance was better than the another algorithms, through a series of latency improvements. We also observed that latency values in LBDAR raises smoothly and stabilizes sooner. In addition, and as a result of the load balanced scheme applied, the global load distribution in the network is also optimized.

This implies that intermediate switches are less loaded because traffic is flowing through the best available paths right away, avoiding unnecessary communications and keeping these resources available to other communications requests.

This work basically presents the methodology of a load balanced routing approach, and after observing initial experiments results, in depth analysis of proposed approach is necessary, in order to extend the idea and make solid proposals about performance gain. During the development of this work, many other related ideas emerged as well, and can be considered as promising.

5.3 Future Work

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

To continue this work, short term goals and activities can be described. First, more experiments must be executed, in order to finely tune load balanced module behavior and scope. This fine tuning can lead to full routing load balanced modules implementation, based on already existing models developed with OPNET simulator tool. Having the load balanced module partially or completely developed, then new experiments may be carried out in order to analyze different conditions or features to enforce load balanced scheme methodology, such as:

-Use different interconnection network topologies, such as k-ary n-cube, 3D meshes, among others

-Execute experiments under different traffic loads, to study LBDAR behavior under different load scenarios and its response in time and latency values

-Study the effect of scientific applications in the interconnection network

-Analyze parallel applications and its communications patterns

- Predict future traffic conditions and be able to detect congestion situation before it effectively appears in the interconnection network

References

[1] 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.

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

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

[4] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, 4th edition, Morgan Kaufmann 2007.

[5] Cheng-Shang Chang, Duan-Shin Lee, and Yi-SheanJou. Load balanced birkhoff-von neumann switches, part 1: one-stage buffering. Computer Communications, 25:611622, 2002.

[6] Brian Towles and William J. Dally. Worst-case traffic for oblivious routing functions.In Proc. of the Symposium on Parallel Algorithms and Architectures, pages 1–8,Winnipeg,Manitoba, Canada, August 2002.

[7] Cheng-Shang Chang, Duan-Shin Lee, and Ching-Ming Lien. Load balanced birkhoffvon neumann switches, part 2: multi-stage buffering. Computer Communications,25:623–634, 2002.

[8] Isaac Keslassy. The Load-Balanced Router. Ph.D. Dissertation, Stanford University,June 2004.

[9] Isaac Keslassy, Cheng-Shang Chang, Nick McKeown, and Duan-Shin Lee. Optimal load-balancing. In Proc. of IEEE INFOCOM, Miami, Florida, 2005.

[10] Wang, Mu-Cheng and Siegel, Howard Jay and Nichols, Mark A. and Abraham, Seth, Using a MultipathNetwork for Reducing the Effects of Hot Spots, IEEE Trans. Parallel Distrib. Syst., 1995, pp. 252—268

[11] Dandamudi, Sivarama P., Reducing Hot-Spot Contention in Shared-Memory multiprocessor Systems,IEEE Concurrency, 1999, pp. 48—59

[12] D. Franco, I. Garc_es, 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, pp. 210-219, New York, NY,USA, 1999

[13] 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.

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

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

[16] 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.

[17] D. Franco, Balanceo Balancing Distributed routing interconnection networks 2000

[18] Marco Fillo, Stephen W. Keckler, William J. Dally, The m-machine multi-computer. In Proceedings of the 28th Annual International Symposium on Microarchitecture, pages 146–156, 1995.

[19] R.E. Kessler and J.L. Schwarzmeier. Cray t3d: a new dimension for cray research. In Compcon Spring '93, Digest of Papers, pages 176–182, February 1993.

[20] N.R. Adiga, M.A. Blumrich, D. Chen, P. Coteus, A. Gara, M.E. Giampapa, P. Heidelberger, S. Singh, B.D. Steinmacher-Burow, T. Takken,M. Tsao, and P. Vranas. Blue gene/l torus interconnection network. IBM Journal of Research and Development, 49, 2005.

[21] Charles Clos. A study of non-blocking switching networks. The Bell System Technical Journal, March 1953.

[22] J. Beetem, M. Denneau, and D. Weingarten. The gf11 supercomputer.In Proc. 12th Annu. Int. Sym. Computer Architecture, pages 108 – 115,June 1985.

[23] K. Hajikano, K. Murakami, E. Iwanbuchi, O. Isono, and T. Kobayashi.Asynchronous transfer mode switching architecture for broadband isdn.In Proc. IEEE ICC 88, pages 911–915, June 1988.

[24] Y. Sakurai, N. Ido, S. Gohara, and N. Endo. Large scale atm multistage switching network with shared buffer memory switches. In Proc. ISS 90, volume 4, pages 121–126, May 1990.

[25] H. Suzuki, H. Nagano, T. Suzuki, T. Takeuchi, and S. Iwasaki. Output buffer switch architecture for asynchronous transfer mode. In Proc.IEEE ICC 89, pages 99–103, June 1989.

[26] AS. Tanenbaum, Computer Networks, Prentice-hallH, Englewood Cliffs,3rd,1997, pp. 775—785

[27] Olav Lysne and Tor Skeie. Load balancing of irregular system area networks through multiple roots. In Proceedings of the International Conference on Communication in Computing, pages 165–171, June 2001.

[28] 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.

[29] 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.

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

[31] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. Computer Networks, 3(4):267–286,September 1979.

[32] InfiniBand Trade Association. Infiniband architecture specification, 1.2 edition, October 2004.

[33] Advanced Switching Interconnect Special Interest Group. Advanced Switching Core Architecture Specification, revision 1.1 edition, August 2004.

[34] W. J. Dally and C. L. Seitz. The torus routing chip. J. Distributed Computing, 1(3):187–196, 1986.

[35] SGS-Thomson Microelectronics. STC104 Asynchronous Packet Switch - Data sheet, June 1996.

[36] Shihang Yan and Geyong Min and Irfan Awan, An Enhanced Congestion Control Mechanism in InfiniBand Networks for High Performance Computing Systems, Advanced Information Networking and Applications, International Conference on, 2006, pp. 845-850

[37] Tamir, Yuval and Frazier, Gregory L., Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches, IEEE Trans. Comput., 1992, pp. 725—737

[38] M. E. Gómez and J. Flich and A. Robles and P. López and J. Duato, VOQ_SW: A Methodology to Reduce HOL Blocking in InfiniBand Networks , [online]. Available.

[39] W. Dally, H. Aoki, and C. MIT. Deadlock-free adaptive routing in multicomputer networks usingvirtual channels. Parallel and Distributed Systems, Jan 1993.

[40] A. Singh, W. Dally, A. Gupta, and B. Towles. Goal: a load-balanced adaptive routing algorithm for torus networks. Computer Architecture, Jan 2003.

[41] A. Singh, W. Dally, A. Gupta, and B. Towles. Adaptive channel queue routing on k-ary n-cubes. Proceedings of the sixteenth annual ACM symposium on Parallel Algorithms and Architectures, Jan 2004.

[42] A. Singh, W. Dally, B. Towles, and A. Gupta. Globally adaptive load-balanced routing on tori. Computer Architecture Letters, Jan 2004.

[43] Duato, J. and Johnson, I. and Flich, J. and Naven, F. and Garcia, P. and Nachiondo, T., A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks, HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture , 2005, pp. 108—119

[44] "Advanced Switching for the PCI Express Architecture". White paper. Available at http://www.intel.com/technology/pciexpress/devnet/AdvancedSwitching.pdf

[45] "Advanced Switching Core Architecture Specification". Available at http://www.asi-sig.org/specifications for ASI SIG.

[46] J. Duato, J. Flich, T. Nachiondo: \A Cost-E_ective Technique to Reduce HOL Blocking in Single-Stage and Multistage Switch Fabrics", Euromicro Conference on Parallel, Distributed and Network-Based Processing, February 2004,pp. 48-53.

[47] G. Sapountzis, M. Katevenis: \Benes Switching Fabrics with O(N)-Complexity Internal Backpressure", IEEE Communications Magazine, vol. 43,no. 1, January 2005, pp. 88-94.

[48] Kyung Min Su and Ki Hwan Yum, Simple and Effective Adaptive Routing Algorithms inMulti-Layer Wormhole Networks, IEEE International Performance, Computing and Communications Conference, 2008, pp. 176-184

[49] Paul, Gratz. and Boris, Grot. and Stephen, Keckler, Regional Congestion Awareness for Load Balance in Networks-on-Chip, Proceedings of the 14th International Symposium on High Performance Computer Architecture , 2008, pp. 203-214

[50] Hu, Jingcao and Marculescu, Radu, DyAD: smart routing for networks-on-chip, DAC '04: Proceedings of the 41st annual Design Automation Conference , 2004, pp. 260—263

[51] Baydal, Elvira and Lopez, Pedro and Duato, Jose, A Family of Mechanisms for Congestion Control in Wormhole Networks, IEEE Trans. Parallel Distrib. Syst., 2005, pp. 772—784

[52] D. Franco, I. Garc_es, 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.

[53] Franco, D. and Garces, I. and Luque, E., A new method to make communication latency uniform: distributed routing balancing, ICS '99: Proceedings of the 13th international conference on Supercomputing ,1999, pp. 210—219

[54] Glass, Christopher J. and Ni, Lionel M., The turn model for adaptive routing, SIGARCH Comput. Archit. News, 1992, pp. 278—287

[55] Valiant, L. G. and Brebner, G. J., Universal schemes for parallel communication, STOC '81:
Proceedings of the thirteenth annual ACM symposium on Theory of computing , 1981, pp. 263—277

[56] Top500 Supercomputers site, Interconnection family share for 06/2010, 2010 [online]. Available: http://www.top500.org.

[57] OPNET Technologies, "Opnet Modeler Accelerating Network R&D," June 2008, http://opnet.com. 2008.

[58] J. H. Kim and A. A. Chien, "An evaluation of the planar/adaptive routing," Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing, pp. 470–478, December 1992.

[59] F.T. Leighton, Introduction to Parallel Algorithms and Architectures:Arrays, Trees,Hypercubes, Morgan Kaufmann, San Francisco, 1992.

[60] P. R. Miller, Efficient Communications for Fine-Grain Distributed Computers,Ph.D. Dissertation, Southampton University, U.K., 1991.

[61] Amit K. Gupta, William J. Dally, Arjun Singh, and Brian Towles. Scalable optoelectronic network (SOEnet). In Proc. of Hot Interconnects, pages 71–76, Stanford,CA, August 2002.